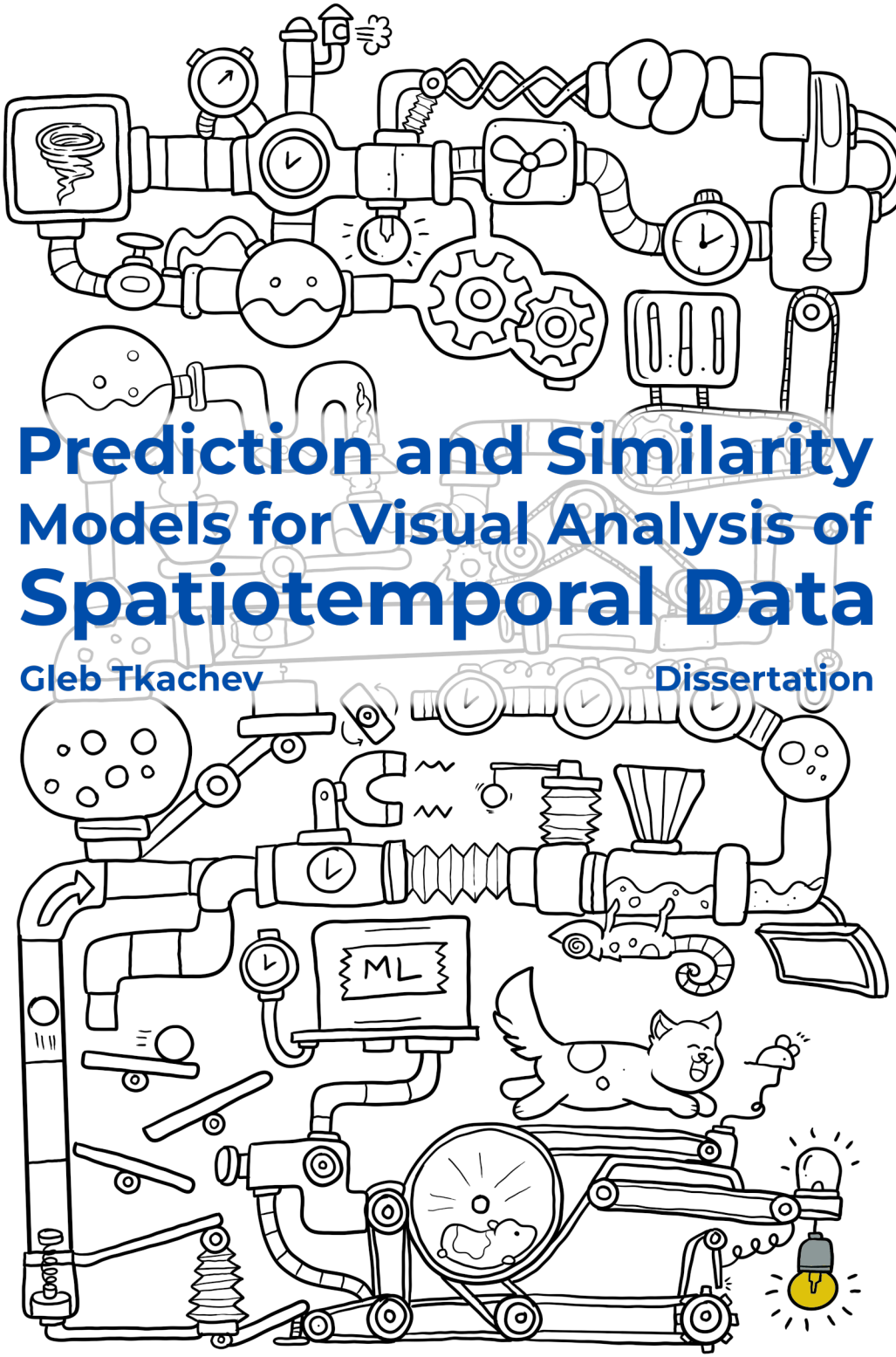


Prediction and Similarity Models for Visual Analysis of Spatiotemporal Data

Gleb Tkachev

Dissertation

Prediction and Similarity Models for Visual Analysis of Spatiotemporal Data

Von der Fakultät Informatik, Elektrotechnik und
Informationstechnik der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Gleb Tkachev

aus Kaliningrad

Hauptberichter:	Prof. Dr. Thomas Ertl
Mitberichter:	Prof. Dr. Steffen Frey
	Prof. Dr. Chaoli Wang
Tag der mündlichen Prüfung:	30 Juni 2022

Visualisierungsinstitut
der Universität Stuttgart

2022

Contents

List of Figures

List of Tables

Acknowledgments

Despite what one says during job interviews, behind any finished PhD thesis stands the work of many people. In fact, their work is everywhere: behind, in front, above and, most importantly, beside. Here I would like to thank all those who stood beside me during this long and winding journey to write this long and winding thesis.

First of all, my gratitude goes to Thomas Ertl, my advisor, who gave me the opportunity to pursue my doctoral degree and join the wonderful crowd at VIS/US. Thank you for your guidance and for making sure we all have everything we need to succeed. I would also like to thank Steffen Frey, who helped me throughout my studies, and without whom I would defend neither my PhD thesis nor cannon rushes. Furthermore, a special thanks to Chaoli Wang for reviewing the thesis and taking part in the defense.

A big thank you to all my co-authors, in particular to Alexander Straub, Christoph Müller, Fabian Beck, Filip Sadlo, Hamid Gadirov, Ilyass Tabiai, Michael Sedlmair, Michael Wermelinger, Moritz Heinemann, Patrick Diehl, Rene Cutura, Shivam Aragwal and Valentin Bruder. And thank you to all the other great people at VIS/US, who made these years fly by: Alex, Cristina, David, Dominik, Frank, Franzi, Johannes, Kuno, Marcel, Maurice, Moataz, Quynh, Stefan, Tobi and many, many more. I would like to explicitly exclude my office-mates Katrin, Michael B. and Sergej from this list, because without our engaging discussions I could have finished a year earlier. Sad. Finally, my thanks to the negative \vspace function for making my publications possible.

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2075 – 390740016 – with projects in the SimTech Clusters of Excellence EXC 310 and EXC 2075.

Summary

Ever since the early days of computers, their usage have become essential in natural sciences. Whether through simulation, computer-aided observation or data processing, the progress in computer technology have been mirrored by the constant growth in the size of scientific data. Although the improvements in scale and resolution of scientific outputs are critical to the advancement of many fields, they also make the analysis more challenging, particularly when studying new data and phenomena, where exploratory analysis is of the most importance.

Over the last decades, visualization experts have proposed many approaches to address the challenge, but even these methods have their limitations. As the data sizes grow, and human perception remains constant, it becomes increasingly difficult to meaningfully aggregate and present the data to the user. This general trend leads to the development of domain-specific visualization methods that can leverage the knowledge of the application context to appropriately filter and abstract the data. However, they require significant expertise and effort to develop, without being easily transferable to other domains. The fast pace of scientific research and its inherent high specialization exacerbate the problem, as existing and emerging subfields constantly set out to study new problems and might not have a dedicated visualization expert to research and develop the tools required for the analysis. Fortunately, domain-agnostic approaches can help to bridge this gap, by both being directly applicable in a large range of applications and by decoupling other visualization techniques from the underlying data.

The development of general methods is a difficult task, but recent advances in the field of Machine Learning (ML) can provide the tools needed to solve it. ML models are a particularly good fit as they can both benefit from the large amount of data present in the scientific context and allow the visualization system to adapt to the problem at hand, rather than rely on fixed algorithms. Still, the existing methods need to be adjusted to account for the spatiotemporal nature of the data, its size and the interactive performance constraints. Even more importantly, many of the successful ML methods rely on supervised data, which is lacking in the scientific context, where the problems are specialized, experts are few and their time is extremely valuable. Therefore, one should pursue models that are unsupervised or require only minimal supervised data.

This thesis presents research into how existing ML approaches can be adapted and extended to enable domain-agnostic visualization of spatiotemporal data. The first introduced technique enables detection of irregular spatiotemporal behavior by means of training prediction models and observing the deviation between their prediction and the actual data. The causes for this misprediction are studied, demonstrating how this information can be used to construct an exploratory visualization of the whole dataset. Furthermore, the prediction error is shown to produce a meaningful

similarity metric, when measured across different members of a simulation ensemble. Continuing the research in this direction, self-supervised methods are investigated in the context of spatiotemporal data, and a new similarity metric is developed. It is shown to outperform existing domain-agnostic approaches and works at interactive speeds, enabling the user to further refine the results. Building upon the insights into ML-based similarity metrics, a new conceptual approach is developed, called Metaphorical Visualization. It allows data to be mapped to a large variety of representations in a very generic fashion, with applications extending even beyond the scientific context. ML models are also applied to another aspect of visualization systems, predicting performance of a distributed rendering algorithm. The developed technique is data- and application-agnostic and enables cluster performance prediction given only single-node measurements. Reflecting upon this diverse set of successful applications, we present a summary of insights and experiences of applying ML models to visualization problems. Combined with the concrete approaches, it constitutes a broad contribution to introducing ML methods into scientific visualization.

Zusammenfassung

Seit den Anfängen von Computern ist ihr Einsatz in den Naturwissenschaften unverzichtbar geworden. Ob durch Simulation, computergestützte Beobachtung oder Datenverarbeitung, die Fortschritte in der Computertechnologie spiegeln sich in der ständig wachsenden Menge an der wissenschaftlichen Daten wider. Obwohl die Verbesserungen in Bezug auf Umfang und Auflösung wissenschaftlicher Ergebnisse für den Fortschritt in vielen Bereichen von entscheidender Bedeutung sind, erschweren sie auch die Analyse, insbesondere bei der Untersuchung neuer Daten und Phänomene, wo explorative Analyse von größter Bedeutung ist.

In den letzten Jahrzehnten haben Visualisierungsexperten viele Methoden vorgeschlagen, um diese Herausforderung zu meistern, aber auch diese Methoden haben ihre Grenzen. Da die Datenmengen wachsen, aber die menschliche Wahrnehmung konstant bleibt, wird es immer schwieriger, die Daten sinnvoll zu aggregieren und dem Benutzer zu präsentieren. Dieser allgemeine Trend führt zur Entwicklung von domänenspezifischen Visualisierungsmethoden, die das Wissen über den Anwendungskontext nutzen können, um die Daten angemessen zu filtern und zu abstrahieren. Die Entwicklung dieser Methoden erfordert jedoch erhebliche Fachkenntnisse und Aufwand und ist nicht ohne Änderungen auf andere Bereiche übertragbar. Der rasante Fortschritt der wissenschaftlichen Forschung und ihre immanente hohe Spezialisierung verschärfen das Problem, da bestehende und neu auftauchende Teilbereiche ständig neue Probleme untersuchen und möglicherweise keine speziellen Visualisierungsexperten haben, um für die Analyse erforderlichen Tools erforschen und entwickeln. Glücklicherweise können domänenagnostische Ansätze dazu beitragen, diese Lücke zu schließen, da sie in einer Vielzahl von Anwendungen anwendbar sind und andere Visualisierungstechniken von den zugrunde liegenden Daten entkoppeln.

Die Entwicklung allgemeiner Methoden ist eine schwierige Aufgabe, aber die jüngsten Fortschritte im Gebiet des maschinellen Lernens (ML) können die notwendigen Werkzeuge zur Lösung dieser Aufgabe liefern. ML-Modelle eignen sich besonders gut, da sie sowohl von der großen Datenmenge im wissenschaftlichen Kontext profitieren können als auch dem Visualisierungssystem erlauben, sich an das jeweilige Problem anzupassen, anstatt sich auf feste Algorithmen zu verlassen. Dennoch müssen die bestehenden Methoden angepasst werden, um der räumlich-zeitlichen Natur der Daten, ihrer Größe und den interaktiven Leistungsbeschränkungen Rechnung zu tragen. Noch wichtiger ist, dass sich viele der erfolgreichen ML-Methoden auf überwachte Daten stützen, die im wissenschaftlichen Kontext fehlen, wo die Probleme spezialisiert sind, und es nur wenige Experten gibt, deren Zeit extrem teuer ist. Daher sollte man Modelle anstreben, die nicht überwacht sind oder nur minimale überwachte Daten benötigen.

In dieser Arbeit wird untersucht, wie bestehende ML-Ansätze angepasst und erweit-

ert werden können, um eine domänenagnostische Visualisierung von raum-zeitlichen Daten zu ermöglichen. Die erste vorgestellte Technik ermöglicht die Erkennung von unregelmäßigem raum-zeitlichem Verhalten durch das Training von Vorhersagemodellen und die Beobachtung der Abweichung zwischen deren Vorhersage und den tatsächlichen Daten. Die Ursachen für diese Fehlprognosen werden untersucht, und es wird gezeigt, wie diese Informationen zur Erstellung einer explorativen Visualisierung des gesamten Datensatzes verwendet werden können. Außerdem wird gezeigt, dass der Vorhersagefehler eine aussagekräftige Ähnlichkeitsmetrik ergibt, wenn er über verschiedene Mitglieder eines Simulationsensembles gemessen wird. In Fortsetzung der Forschung in dieser Richtung werden selbstüberwachte Methoden im Kontext von räumlich-zeitlichen Daten untersucht und eine neue Ähnlichkeitsmetrik entwickelt. Es wird gezeigt, dass sie bestehende domänenagnostische Ansätze übertrifft und mit interaktiver Leistung arbeitet, die es dem Benutzer ermöglicht, die Ergebnisse weiter zu verfeinern. Aufbauend auf den Erkenntnissen über ML-basierte Ähnlichkeitsmetriken wird ein neuer konzeptioneller Ansatz entwickelt, die sogenannte Metaphorische Visualisierung. Sie ermöglicht es, Daten auf eine Vielzahl von Darstellungen in einer sehr generischen Weise abzubilden, wobei die Anwendungen sogar über den wissenschaftlichen Kontext hinausgehen. ML-Modelle werden auch auf einen anderen Aspekt von Visualisierungssystemen angewendet, nämlich die Vorhersage der Leistung eines verteilten Rendering-Algorithmus. Die entwickelte Technik ist daten- und anwendungsunabhängig und ermöglicht die Vorhersage der Leistung von Clustern auf der Grundlage von Einzelknotenmessungen. In Anbetracht dieser vielfältigen erfolgreichen Anwendungen präsentieren wir eine Zusammenfassung der Erkenntnisse und Erfahrungen bei der Anwendung von ML-Modellen auf Visualisierungsprobleme. Zusammen mit den konkreten Ansätzen stellt dies einen umfassenden Beitrag zur Einführung von ML-Methoden in die wissenschaftliche Visualisierung dar.

List of Abbreviations and Acronyms

AR	Augmented Reality
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
CPU	Central Processing Unit
EMD	Earth Mover's Distance
FC	Fully-Connected (Layer)
GAN	Generative Adversarial Networks
GD	Gradient Descent
GPU	Graphics Processing Unit
HCI	Human-Computer Interaction
HPC	High-Performance Computing
InfoVis	Information Visualization
IO	Input/Output
LAP	Linear Assignment Problem
LSTM	Long Short-Term Memory
ML	Machine Learning
MPI	Message Passing Interface
MSE	Mean Squared Error
NLP	Natural Language Processing
NN	Neural Network
POT	Python Optimal Transport (Library)
RAM	Random Access Memory
ReLU	Rectified Linear Unit
SciVis	Scientific Visualization
SGD	Stochastic Gradient Descent
SIFT	Scale-Invariant Feature Transform
SIMD	Single-Instruction Multiple-Data
SNE	Stochastic Neighborhood Embedding
SOM	Self-Organizing Map
SSIM	Structural Similarity Index
SSL	Self-Supervised Learning
STD	Standard Deviation
SVD	Singular Value Decomposition
UI	User Interface
UMAP	Uniform Manifold Approximation and Projection
VC-d.	Vapnik–Chervonenkis Dimension
VR	Virtual Reality
VRAM	Video RAM

1

Introduction

Spatiotemporal data is ubiquitous in natural sciences. It appears in many fields, as most physical phenomena have innate spatial and temporal extents. Conceptually, spatiotemporal data can be classified into sparse and dense. Sparse datasets are collections of objects that are freely positioned in space-time, e.g., particle data in molecular dynamics. In contrast, dense datasets are dense fields defined on a discrete mesh that constitutes the spatiotemporal domain of the data, such as velocity fields in computational fluid dynamics. This thesis is primarily concerned with field data, although many of the developed concepts could also be extended to particle datasets.

The spatial character of the data strongly compels us to also display it spatially. This can be a great advantage, as it makes the visualization intuitive, but can also be a curse since most datasets have three spatial dimensions that need to be displayed in just the two dimensions of a computer screen. Especially for larger and more complex datasets, this can result in occlusion and clutter that prevents a clear view of the data. The problem becomes even more pronounced when the temporal dimension is involved. Similarly to space, time is also commonly mapped to time in visualization, i.e., through animation. And while intuitive, getting an overview of the data can become even harder due to the difficulty of perceiving and recalling changes over time (*change blindness*). Thus, other visual mappings are often pursued as an alternative or an addition to a direct spatiotemporal visualization, for instance, mapping time to space or color. With this in mind, visualizing just the four spatiotemporal dimensions can already become a very challenging task, but the final "nail" in the hopes of a direct spatiotemporal mapping comes in the form of *ensemble data*. Ensembles are collections of simulation or experimental runs, gathered under similar conditions and analyzed together. This

type of spatiotemporal data adds yet another dimension, called “run” or “member”, that does not have a direct visual counterpart, but needs to be represented.

Considering the dimensionality of the spatiotemporal data, its visualization cannot rely solely on finding an appropriate visual mapping. Interactivity and filtering are required to reduce the data, allowing the user to select and view its parts separately. This can be as simple as controlling the camera view or scrolling through a temporal animation, but can also involve data-specific filters, 3D object manipulation, coordinated views and many other complex techniques. Another solution is to perform aggregation and feature extraction, trying to reduce the data while preserving the most important information. This could be done to remove some of the dimensions, construct an overview of the data or to present only the most relevant details to the user. And although many solutions were proposed, they inherently come with trade-offs in the completeness of the presented data and rely on specialized assumptions that may bias the analysis.

1.1 Motivation and Research Questions

The steady improvement in computational and sensor technologies accentuates the challenges of spatiotemporal visualization. On the one hand, larger scales of simulation and better resolution of experimental data call for increasingly sophisticated data reduction and abstraction. This reduction is often difficult to achieve without making domain-specific assumptions about the data, leading to powerful yet highly specialized solutions that can overlook the unexpected and do not easily transfer to other applications. And on the other hand, as computational resources become cheaper and more ubiquitous, more data from novel subfields and problems is collected. Smaller laboratories might not have sufficient visualization expertise to develop the specialized tools required for the analysis. Therefore, visualization should pursue generic domain-agnostic methods, as they can be useful in a wide range of scientific domains, reduce the number of assumptions in the analysis and provide a way of bridging existing visualization research with new data.

Although this is a difficult task, *Machine Learning* (ML) can provide the tools needed to create visualization methods that are more general. One way of looking at ML is that it allows us to learn algorithms from the data, rather than manually encoding our knowledge into a computer program. The past decades have shown that raw computational power eventually beats cleverly designed algorithms – the “bitter lesson” repeated in chess, speech recognition and computer vision (Sutton 2019). And while in many application areas this can be infeasible due to the lack of training data, visualization is inherently about data, and thus has a rich potential for integration with ML.

Furthermore, the state-of-the-art simulation methods are now also pivoting towards integration with ML models. This thesis project is a part of the Excellence Cluster

EXC-2075 "Data-Integrated Simulation Science (SimTech)" that pursues data-integrated simulation approaches as its primary research objective. Thus, ML-driven visualization techniques become even more desired, as they are now needed not only to analyze the simulation outputs, but also to understand the data that supports the modeling itself, helping to successfully merge physics- and data-based simulation methods (Focus Challenge 2 of SimTech). Furthermore, visualization is also in a unique position to alleviate one of the main concerns about ML models — their black-box nature and innate modeling uncertainty, since a visualization can help bring the scientist into the ML loop, and support the steering and the validation of the ML model.

Nevertheless, to successfully apply ML methods to the problems of scientific visualization, one needs to overcome a number of challenges, especially when dealing with spatiotemporal fields. These challenges translate directly into the research questions of this thesis:

> *RQ1. How to extend existing ML techniques to visualization of spatiotemporal fields?*

The machine learning techniques that are the most relevant to our visualization scenarios are usually designed for working with image and video data. Spatiotemporal data is conceptually similar, but often has additional dimensions and attributes that create both challenges and opportunities that need to be considered. Furthermore, while ML methods operate on and even require large data sizes, they are generally suited for a large number of data samples, rather than samples of a large size. Evaluating a typical model on a single scalar field can become very computationally expensive or even infeasible.

> *RQ2. How to train ML models for visualization with little or no supervised data?*

A lot of the most successful ML methods rely, at least in some part, on supervised data. Unfortunately, in the context of scientific data visualization, supervised data is generally not available. Domain sciences are highly specialized, and their number is large. At the same time, there are only few experts that could provide the supervised data, and the experts' time is highly valuable. Instead, techniques using unsupervised

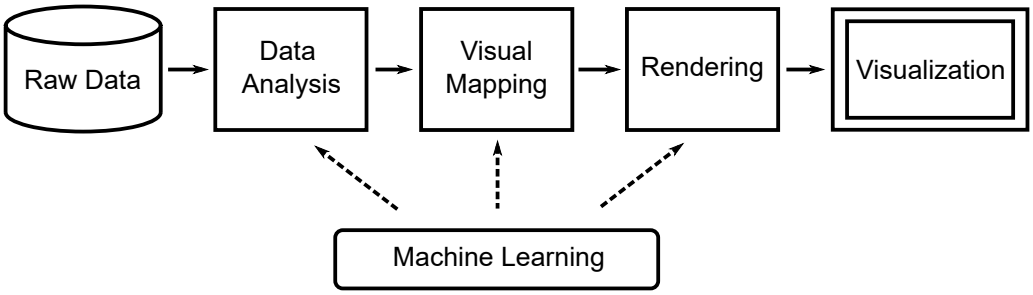


Figure 1.1 — The visualization pipeline. Machine learning methods can be applied at every step of the pipeline: to extract features during data processing, to generate new visual representations for the data or to speed up and enhance the rendering stage.

learning methods should be developed, particularly when domain-agnostic solutions are desired.

RQ3. How can the user interact with the ML model?

Although ML methods can be a powerful tool for data analysis, any visualization system is ultimately about enabling the user to solve their task. As such, it can become difficult to couple ML models with a human analyst, since models can often be opaque and not accept human input. In addition, ML techniques can incur significant performance costs, both during training and inference, thereby preventing an interactive workflow. For this reason, additional steps should be taken to ensure that the visualization system has sufficient performance and allows for user control.

1.2 Outline and Contributions

This thesis presents an investigation of how machine learning methods can be applied to spatiotemporal data to produce visualization techniques that are more automated and domain-agnostic. The contributions are best structured when viewed in the context of the visualization pipeline (Figure 1.1). The visualization pipeline (Johnson and Hansen 2004) is a conceptual model of a visualization application that partitions it into several distinct stages. First comes the data analysis stage, which prepares the raw data, performs appropriate data filtering and aggregation. Next, the prepared data undergoes visual mapping, where it is mapped to visual variables such as position and color, often stored in the form of geometric primitives and attributes. Finally, the rendering stage

generates the final image from the geometric primitives constructed at the previous stage.

As seen in Figure 1.1, ML techniques can find applications in all stages of the visualization pipeline. In the analysis stage, ML can help detect events or structures, measure similarity or extract expressive features that can be presented in the visualization. Contributions of this type are discussed in Chapters 3, 4. During visual mapping, visual representations can be automatically selected or generated by an ML model, which we discuss in Chapter 5. Rendering is the most performance-centric of the stages, and so ML methods are often applied here to speed up the computation by learning parts or even the whole rendering function. However, they can also be used to predict the rendering performance, helping to design an efficient visualization system (Chapter 6). See Section 7.1 for a detailed description of ML in the visualization pipeline.

Now, let us briefly summarize the content and the contributions of each chapter, and how they relate to the research questions stated above.

Chapter 3 — Detecting Irregular Behavior in Spatiotemporal Volumes

The chapter introduces a novel technique for detection of irregular behavior in spatiotemporal volumes (**Tkachev** et al. 2021a). In this method, we apply ML models at the analysis stage of the pipeline (Figure 1.1) by training a set of models to predict local behavior in the data and then extracting regions where the prediction fails. We demonstrate that this error marks regions of unique, uncertain or complex behavior, all of which are of interest to the analyst. This feature can be used to present an overview of irregularities in the data, or to guide the visualization by selecting key timesteps. Furthermore, using models of different capacity we can extract additional information about the irregular regions and help automatically select configuration parameters appropriate for a given dataset. The locality of the models makes the training computationally feasible (RQ 1), while the auto-regression task avoids the need for supervised data (RQ 2).

Chapter 4 — Learning Spatiotemporal Similarity Metrics

This chapter presents three self-supervised methods and focuses on learning similarity metrics for spatiotemporal data, thereby also applying ML at the analysis stage of the visualization pipeline. The first approach is based on the local prediction models (Chapter 3) and provides a similarity metric for ensemble datasets (**Tkachev** et al. 2021a) This is achieved by measuring the prediction errors of models when trained and evaluated on different members of the ensemble. Low errors imply that the predicted behavior was also observed during the training, indicating similar members. The second approach employs a self-supervised model to measure similarity of spatiotemporal patches (**Tkachev** et al. 2021c) and therefore addresses RQ 2. The resulting metric is more granular and allows data queries consisting of user-provided examples, enabling the user to influence the results (RQ 3). This method is also computationally

efficient and runs interactively as part of a prototype system for exploring ensemble datasets (RQ 1). In the third part, we briefly present a related approach (Gadirov et al. 2021) that utilizes autoencoders for visual exploration of ensembles (RQ 1, RQ 2).

Chapter 5 — Metaphorical Visualization

The chapter presents a new conceptual approach to visualization that allows data to be mapped to a diverse set of concepts such as words, images and visual styles (**Tkachev** et al. 2022). The core idea is based on finding a mapping of data to concepts that preserves similarity relationships between the data points, such as self-supervised similarity metrics learned by ML models (RQ 2). Because the ML models are used to compute a new representation for the data, this application of ML belongs to the visual mapping stage of the pipeline. The approach is very generic and can be applied to data from many domains, both scientific and social. We describe several types of metaphorical mappings and implement many different applications to demonstrate their flexibility and personalization potential (RQ 3).

Chapter 6 — Performance Prediction for Parallel Volume Rendering

In Chapter 6, we describe how ML can be used to predict performance of distributed volume rendering, thus situating the method at the rendering stage of the visualization pipeline. Training a model to predict cluster performance is a difficult task, since it would require supervised data (performance timings) to be captured on a large variety of cluster configurations, which is generally infeasible. Instead, we introduce a two-level approach, where we predict the cluster performance based on the local node performance, captured in the form of performance histograms (**Tkachev** et al. 2017). With this, the cluster model is insulated from hardware- and application-specific details of the local nodes, allowing us to train it with data from a single cluster (RQ 2). We demonstrate the accuracy of this approach and discuss how it can be used to help with procurement of rendering equipment.

Chapter 7 — Machine Learning in Scientific Visualization

In the final chapter, we present an overview of the existing ML applications in scientific visualization. We distinguish between quantitative and qualitative types, motivating further investigation of qualitative approaches. Then, we discuss the common challenges for this line of research and what we learned in the process of solving them. Here, we touch on all three of our research questions. Finally, we enumerate some general directions for future work in applying ML to SciVis.

Contributions Beyond This Thesis

During the work on this thesis, there were several other contributions to collaborative projects. We chose not to include them into the thesis to keep it more focused, but would still like to briefly mention them. See also the Author's Work list.

In collaboration with Polytechnique Montréal, we developed a technique for automatic detection of crack areas in camera images of fiber materials (Tabiai et al. 2019a). Here I designed and implemented the core extraction method, as well as the visualization of its results, and Ilyass Tabiai et al. collected the experimental data and provided their valuable expertise in material science. In Agarwal et al. 2020, we proposed an approach to dynamic set visualization based on the Formal Concept Analysis. I developed an early prototype of the idea, which was later reworked and extended considerably by Shivam Agarwal et al., earning the Best Paper Award at VMV 2020. Finally, in Heinemann et al. 2021, we introduced a system for analysis of droplet dynamics in spray simulations, demonstrating another application of our prediction-based irregularity detection (Chapter 3). For this work, I mainly advised on the machine learning aspects of the method, while Moritz Heinemann et al. performed most of the work.

We have also published the code for several of our publications. Tabiai et al. 2019b is the implementation and data from our crack area detection method (Tabiai et al. 2019a). **Tkachev** et al. 2021b contains the implementation of the self-supervised similarity approach (**Tkachev** et al. 2021c, Section 4.3), which received the Graphics Replicability Stamp. And in **Tkachev** 2022 we published PyPlant – a generic software framework for machine-learning pipelines that we developed to support the projects in this thesis.

2

Background

In this chapter, we briefly present the background knowledge that will be helpful in understanding the rest of the thesis. The first part (Section 2.1) is dedicated to visualization and two of its more advanced topics: ensemble visualization and distributed volume rendering. The second part (Section 2.2) is concerned with machine learning, starting with a quick introduction to the field, followed by a discussion of neural networks, and finally, advanced concepts like self-supervised and contrastive learning.

2.1 Visualization

Visualization is the central topic of this thesis, and therefore, we assume some minor prior knowledge about the field and its research challenges. Nevertheless, in this section we introduce the concepts that are most critical to understanding the following chapters. First, in the next subsection, we discuss ensemble visualization – a relatively new, but very activate research subdomain concerned with visualization of multiple simulation and experimental outputs. And in Section 2.1.2, we introduce the foundations of distributed volume rendering, which will be needed later in Chapter 6.

2.1.1 Ensemble Visualization

Since the available computational resources and measurement devices keep improving, domain scientists have the increased ability to generate larger and more accurate data. One particular outcome of this trend is the collection of series of simulations or experiments that pertain to the same studied phenomenon. The most common

example are simulation ensembles consisting of many runs under different simulation parameters. Such data can allow the domain scientist to study different physical regimes, quantify uncertainty and evaluate the simulation code. Ensemble visualization is a subfield of scientific visualization that focuses on analysis of ensemble data. It adapts the traditional SciVis techniques to ensembles, as well as develops new approaches that are necessary to handle its unique challenges.

The meaning of the term "ensemble data", and consequently, ensemble visualization, drifts across different application contexts, as one would expect from a new research domain. In some literature, ensemble data is understood as a subtype of uncertain data (Bonneau et al. 2014), with different simulation runs being interpreted as samples from some underlying distribution. The resulting distribution can capture the stochastic nature of the simulation method (e.g., Monte Carlo approaches) or the uncertainty over the simulation model and parameters (e.g., multi-model ensembles in weather forecasting). However, for our purposes, this perspective is limiting because some of our ensemble datasets (Section 4.3.6) contain significantly different simulation outcomes. The resulting distribution is complex and multi-modal, and thus precludes basic statistics from being descriptive, which would be the common way to proceed in ensemble visualization. In this thesis, we specifically focus on simulation and experimental ensembles. So we view ensemble data as a type of multi-faceted spatiotemporal data that is characterized by having the additional "member" dimension that enumerates different simulation or experiment outcomes (Kehrer and Hauser 2013). This additional dimension significantly complicates the visualization, as it increases the data size, presents yet another dimension that needs to be mapped in a meaningful way, and most importantly, is qualitatively different from being "just another spatial dimension". It needs to be treated differently during the analysis and induces new analytic tasks that need to be supported (more on tasks below).

In total, ensemble data has five distinct dimensions: *location, time, variable, member and ensemble* (Wang et al. 2019), which we illustrate in Figure 2.1. The *location* dimension consists of two or three spatial dimensions, but since they are homogeneous, we group them into one conceptual axis. The *variable* dimension enumerates different physical quantities recorded for each data point, e.g., pressure, velocity, temperature, etc. The *ensemble* dimension occurs in applications where different ensembles, i.e., set of runs, are compared with each other. For example, to compare the simulation model outputs for different output resolutions (Biswas et al. 2017). However, in this work, we do not perform ensemble-to-ensemble comparisons and leave out the fifth dimension.

Another important facet of spatiotemporal ensembles is their connection to the simulation/experimental parameter space. Typically, each member of the ensemble has additional metadata containing the inputs on which the generative process was conditioned to produced the member data. These could be the physical constants used in the simulation code or the experimental conditions for measurements. In general, the

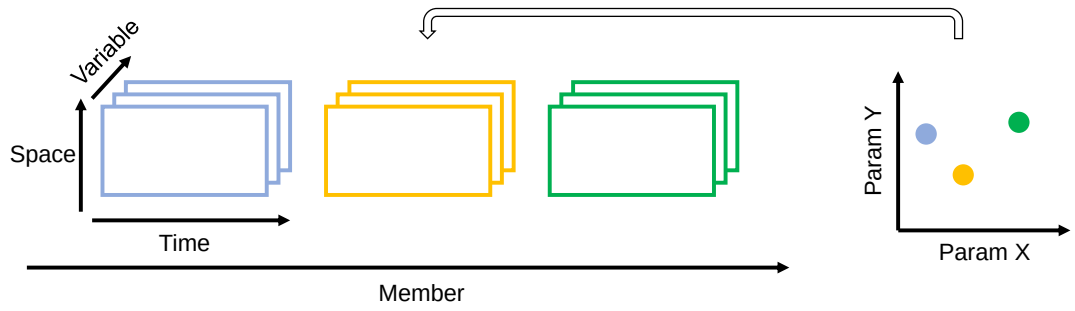
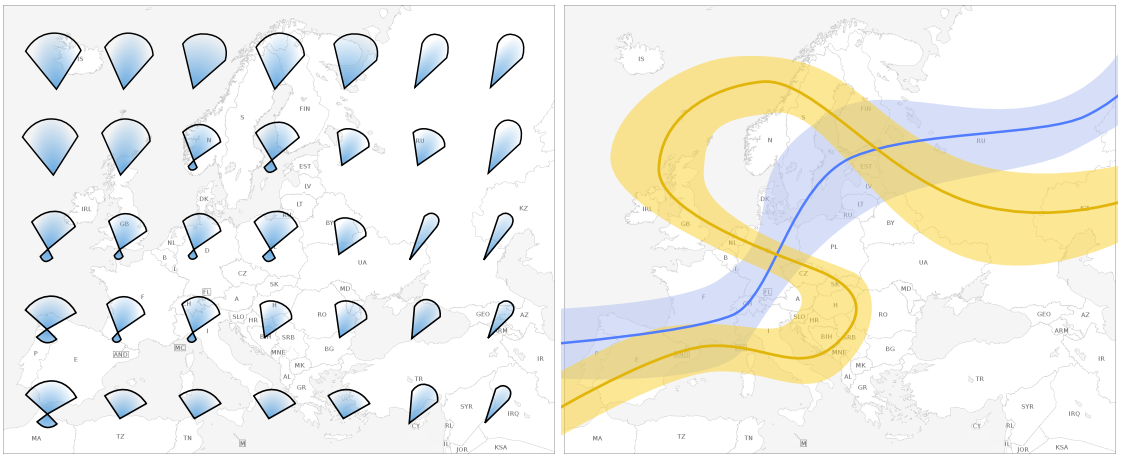


Figure 2.1 — Ensemble data consists of many members, each being a volume with two or three spatial and often a temporal dimension, e.g., 2D+T or 3D+T. Additionally, the volume may contain multiple variables (fields), e.g., pressure (scalar), velocity (vector), stress (tensor). Ensemble data is typically a product of data-generating process (simulation or experiment) that is conditioned on a set of parameters (simulation parameters, experimental setup). Thus, each member can also be viewed as a point in a multi-dimensional parameter space (on the right).

parameters are a heterogeneous set of values that could contain anything, but they often can be thought of as numerical values that form an n-dimensional parameter space (Figure 2.1, right). Then, each ensemble member corresponds to a point in that space. This perspective can be particularly useful when supporting parameter analysis tasks. Still, the parameter metadata is an optional feature of an ensemble dataset, and its importance is highly task dependent: sometimes, the members are stochastic realizations of the same underlying parameters, or the parameter values are not of significance, and only the actual spatiotemporal outcomes are of interest.

Having brought up the tasks, let us briefly discuss them. Following Wang et al. 2019, one can distinguish five key analytic tasks in ensemble visualization. *1. Overview* – get a high-level understanding of the structure and variations within the ensemble. *2. Comparison* – compare pairs or sets of ensemble members, find differences and similarities between them. *3. Clustering* – organize the ensemble members into groups based on their similarity. *4. Temporal analysis* – study how one or more ensemble members evolve over time. *5. Parameter analysis* – understand the connection between the members and the parameter space. In this thesis, we are concerned primarily with the development of similarity metrics for ensemble visualization (Chapter 4). Similarity metrics directly support the clustering task by proving a better basis for grouping the ensemble member and their individual regions. Furthermore, similarity and clustering are helpful both in extracting an overview of the dataset and in parameter space analysis (Section 4.3.6).



(a) Dense glyph-based **(b)** Sparse contour-based

Figure 2.2 — a: A dense glyph-based visualization of a vector field ensemble (Jarema et al. 2015). The ensemble members are sampled at grid-defined locations, aggregating local vector field values into a direction distribution glyph. b: A sparse contour based visualization (Ferstl et al. 2016). A single representative isoline is extracted to represent each member. The isolines are then grouped into two clusters and plotted as a contour showing the variance within each cluster.

It is difficult to succinctly describe the techniques in ensemble visualization, because they are quite diverse and often depend on the application. Still, the most common idea for how to incorporate the member dimension into the visualization is aggregation. One approach to aggregation (we will call it *dense*) involves computing statistics over the different simulation/experiment realizations at every spatiotemporal point or at least in a dense grid. An example of this approach is the glyph-based vector field visualization by Jarema et al. 2015 (Figure 2.2a), where several underlying vector fields were aggregated into directional distribution glyphs. We see a dense sampling of the underlying data, and the visualization tries to preserve as much spatial information as possible. This works well for spatially smooth data and sufficiently similar ensemble members, but can become problematic with highly heterogeneous and multi-modal ensembles.

In the *sparse* aggregation approach, each member is represented by a single distinct feature extracted from the data, e.g., the position of a vortex or an informative isosurface. For instance, see a schematic depiction of the approach by Ferstl et al. 2016. An isoline is extracted from each member, which can be plotted as a contour to represent the member (a spaghetti plot). The authors take it further and cluster the contours, producing the contour variability plots shown in Figure 2.2b. Such representation is sparse in the sense that it takes only select parts of the original member data, trading the potential

information loss for the ability to depict important aspects of individual members. The effectiveness of this approach wholly depends on the quality of the extracted features, which usually means that the feature extraction needs to be specialized towards the exact domain or even the single application at hand. We believe that machine learning methods can be particularly useful in that regard, and so we explore this possibility throughout the thesis.

These two examples are, of course, only a sample of the many proposed ensemble visualization techniques, and we refer the interested reader to the surveys by Kehrer and Hauser 2013, Sedlmair et al. 2014 and Wang et al. 2019.

2.1.2 Distributed Volume Rendering

Volume Rendering is a technique for visualizing dense 3D data defined on a mesh or a regular grid, such as the outputs of simulation software or dense experimental measurements (e.g., from cameras or scanners). Therefore, it is one of the core techniques in scientific visualization. The basic idea behind volume rendering is to shoot a ray from each pixel through the 3D dataset, and sample the data along the ray, accumulating the color as it would be "seen" from the camera.

Just as any rendering task, volume rendering is a highly parallelizable problem: the value of each pixel, as well as the transformation of every geometric primitive are mostly independent of each other. Historically, this parallel potential was exploited through multi-threading and data parallelism available to general computing architectures. Later, the increased demand for graphics processing lead to the development and wide-spread adoption of a *Graphics Processing Unit (GPU)*. Compared to a traditional CPU, a GPU sacrifices universality to specialize on efficient processing of graphics pipelines. This is achieved primarily by implementing the *Single-Instruction Multiple-Data (SIMD)* architecture, where the hardware is designed to efficiently execute a single instruction across many inputs. Such instructions are very common in graphical computations. For example, vertices of a mesh all undergo a projection into camera coordinates, and a shading computation is performed for every fragment.

Similarly, volume rendering can be significantly accelerated by a GPU, and so a lot of research was focused on GPU-based volume rendering, further optimizing its performance. As a result, today one can render scientific datasets of practical sizes in real time on a standard workstation. Nevertheless, scientific data is growing ever larger, as the same hardware advances that made the rendering faster also allowed for larger-scale data generation. When performance of a single machine becomes insufficient to handle, e.g., a super-computer simulation output, one turns to distributing the rendering process across multiple GPUs, machines or both.

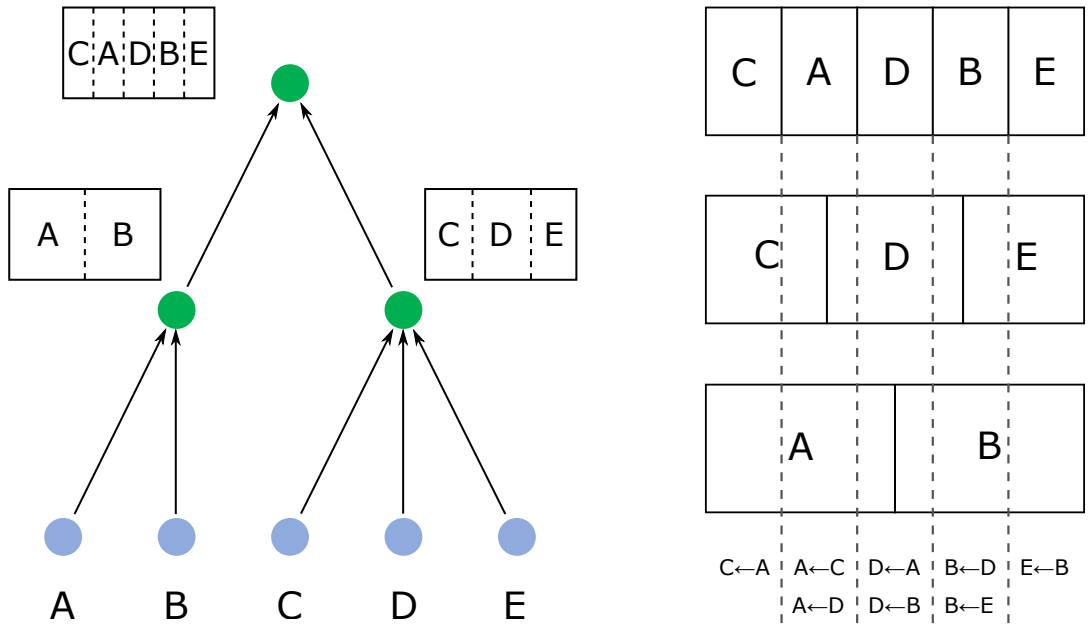
Distributed Volume Rendering, and distributed rendering in general, are methods of

parallelizing the rendering computing across multiple interconnected processors. A processor in this context can be a CPU core, a GPU, a cluster node or any computational unit that is capable of rendering computations. Each processor is assigned a part of the total computation, and the distributed rendering methods can be classified based on how the assignments are made. Rendering is essentially a process of computing the effect of each geometry primitive on each pixel, and can thus be viewed as sorting the primitives to their positions on the screen (Sutherland et al. 1974). For distributed rendering, in which different processors are responsible for different pixels and primitives, it means that the data needs to be re-distributed among them at some point of the process. Thus, Molnar et al. 1994 proposed to distinguish between *sort-first*, *sort-middle* and *sort-last* algorithms, depending on when the sort occurs wrt. the main two rendering stages – geometry processing and rasterization. *Sort-first* renderers first determine where each primitive falls on the screen, and distribute them accordingly among the processors, each responsible for a portion of the screen. In *sort-middle*, the primitives are distributed randomly, processed and then re-distributed before rasterization. And finally, *sort-last* methods process and rasterize the primitives randomly, but then re-distribute and merge the intermediate images.

Specifically in volume rendering, a similar classification can be applied, distinguishing among *image-space partitioning* (sort-first), *object-space partitioning* (sort-last) and *hybrid* approaches. *Image-space partitioning* methods assign each processor (node) a region of the final image. During rendering, the nodes need to compute which part of the volume affects their respective screen region and render it, producing a part of the final image. *Object-space partitioning* assigns each node a partition of the volume. When rendering, each node simply renders its own partition of the data, but afterwards, an expensive composition operation needs to be performed, to merge the results of each node into the final image.

In this thesis, in Chapter 6, we study the performance of an object-partitioning renderer, so let us discuss it in more detail. Any object-partitioning method consists of two main stages: local rendering and *compositing*. During local rendering, every node renders a data partition into its frame buffer. This can be repeated for multiple partitions, rendering in visibility order and compositing them into the same buffer. Because the partitioning of the data happens before rendering, i.e., before its location on the screen is determined, a node's partition can fall anywhere on the screen. Thus, these intermediate buffers need to be composited on top of each other to form the final image.

After the rendering, each node has an image covering the whole screen space, but containing only the contributions from its own volume partitions. Typically, the partitions are convex, e.g., boxes, so we can merge the local images by merging them in visibility order. A simple method of orchestrating this composition is to send all the images to one node, which then sequentially composes the images. This approach is easy to implement, but it is also very inefficient, because all the nodes are stalling during the



(a) (b)

Figure 2.3 — An example of 2-3 swap compositing. The compositing proceeds in steps, each step corresponding to a level of the compositing tree (a). First, at the leaf vertices, each node holds the whole image that has only local data rendered. With each step up the tree, nodes exchange their data and own a smaller but fuller regions of the image space. b: During the second step, each node receives data from others, whose regions overlap its new domain. This results in eight messages being sent.

compositing. A better method is to assign each node a partition of the screen space, and have each node send each partition of its intermediate image to the responsible node. In this approach, called *direct send*, the compositing operations are distributed across the nodes, utilizing them more efficiently. But another problem is introduced: each node is sending a message to every node, resulting in $n(n+1)$ messages, overloading the interconnect between the nodes. There are several proposed methods for improving the network efficiency, one of which is the *2-3 swap*.

The *2-3 swap* compositing scheme (Yu et al. 2008) organizes the communication to be performed in small groups. Even though the total amount of data sent is higher than with direct send, the lower message count leads to better overall performance. With 2-3 swap, the composition is done in steps, and in each step nodes exchange and compose image data in groups, communicating with at most four nodes per step. The communication and compositing are orchestrated using a pre-constructed tree (see Figure 2.3a). Each step of the method corresponds to a level of the tree, and each compositing group is

shown as a vertex of the tree.

The leaves represent the initial state, with each group consisting of only one node. At the start, every node is responsible for the whole image space, but has only its own data rendered. As we move up the tree with each step, nodes exchange data and compose the intermediate images, obtaining contributions from more data partitions, while reducing the image space region that they own. The process terminates at the root vertex, where each node ends up with a partition of the final image.

We illustrate the algorithm in Figure 2.3. In the first step, the nodes A and B form a new group and exchange their data. The node A receives the top half of the image from B, thus becoming responsible for this region, and vice versa. In parallel, node C, D and E trade their data, but in this group, each node communicates with two others. During the second (and final) step, the two groups from before are merged into the final group, where each node owns a fifth of the image, pulling data from nodes that held data relevant for their new region. For example, in Figure 2.3b the node D receives data from A and B, since they both have data that falls into D's new domain. Once the exchange is complete, each node holds a partition of the final result, which can be gathered on one node, stored to disk or sent to a display.

Using the 2-3 swap scheme, nodes send at most $4\lfloor \log_2 N \rfloor$ messages in total. And although up to $\frac{4}{3} < pixel\,number >$ data is exchanged overall (Yu et al. 2008), the scheme on average outperforms simpler alternatives, such as the direct send and the binary swap (Ma et al. 1994).

2.2 Machine Learning

This section provides an introduction to machine learning and specifically to the ideas that are particularly relevant for the following chapters. First, we present a high-level view of machine learning and its most ubiquitous concepts. Then, we focus on neural networks, since they are heavily used throughout this work. After covering the basics, we proceed to discuss more advanced ML approaches, including self-supervised and contrastive learning.

2.2.1 Machine Learning Basics

Machine learning is an area of computer science studying algorithms that improve their performance through experience rather than by being explicitly programmed. ML can also be seen as a type of applied statistics that focuses on the use of computers to perform predictions. Formally, a machine-learning problem can be defined as a combination of a task (T), experience (E) and a performance measure (P), thus the

system needs to improve its performance on the task T given experience E (Mitchell 1997).

Three most basic examples of a task are classification, regression and generation. In *classification*, the system needs to assign one of predefined categories to an input (usually represented as a vector $\boldsymbol{x}$). *Regression* is the task of predicting a real-valued vector $\boldsymbol{y}$ given an input vector $\boldsymbol{x}$. Note that this usage of the term deviates from its usual meaning in statistics, where it has more to do with the analysis of the factors that influence $\boldsymbol{y}$. Finally, *generation* means producing new samples of $\boldsymbol{x}$. Other more complex tasks could include denoising, machine language translation, playing chess, etc.

The experience E comprises all the information that the learning system uses to improve its performance. The experience is most typically represented as a *training dataset*, i.e., a set of example data points that are sampled from some underlying data-generating process. We usually conceptualize this process as a probability distribution, from which the training data points $\boldsymbol{x}_i$ are drawn: $\boldsymbol{x}_i \sim p(\boldsymbol{x})$. Depending on the form of the available data, one could roughly distinguish between unsupervised and supervised learning. In *unsupervised learning*, we have random samples $\boldsymbol{x}_i$ of the input and want to learn something about the distribution $p(\boldsymbol{x})$. While in *supervised learning*, the input samples $\boldsymbol{x}_i$ are accompanied by outputs $\boldsymbol{y}_i$, drawn from some distribution $p(\boldsymbol{x}, \boldsymbol{y})$. Our task then is to learn the distribution $p(\boldsymbol{y} \mid \boldsymbol{x})$, i.e., to predict the output given the input. Although the experience and the task are usually aligned in this way, this does not have to be the case. In principle, we could try to solve a supervised classification task using only unsupervised data (experience). We talk more about these distinctions in Section 2.2.4.

The performance measure P captures how well the system is solving the task and is usually task-dependent. Often, the task could be reduced to approximating some probability distribution, e.g., $p(\boldsymbol{y} \mid \boldsymbol{x})$. And so we measure the performance as a divergence between the distribution produced by our model and the data distribution. However, measuring the performance can be more difficult for other tasks, e.g., in machine text translation we cannot rely simply on the difference between the system's translation and the dataset, as there could be innumerably many valid translations.

How would we construct a learning system for T, E and P? There are many possible solutions, the simplest of which is to store the available data and provide answers by simply looking up the most similar example in our data. The most common approach is to define a parametric function f_θ and use it to model the data-generating or some related distribution, e.g., $p(\boldsymbol{y} \mid \boldsymbol{x})$ for a supervised task. The idea is that a function (a *model*), can efficiently capture the structure of the distribution and thus perform well on unseen data. To find parameters θ that result in a good approximation, we can use the training dataset $\mathbb{X} = \{x_i, y_i\}$. Typically, the performance on the training data is defined by an *error* or *loss function* L, and we search for parameters $\hat{\theta}$ that minimize

the error, thereby *training* the model f_θ:

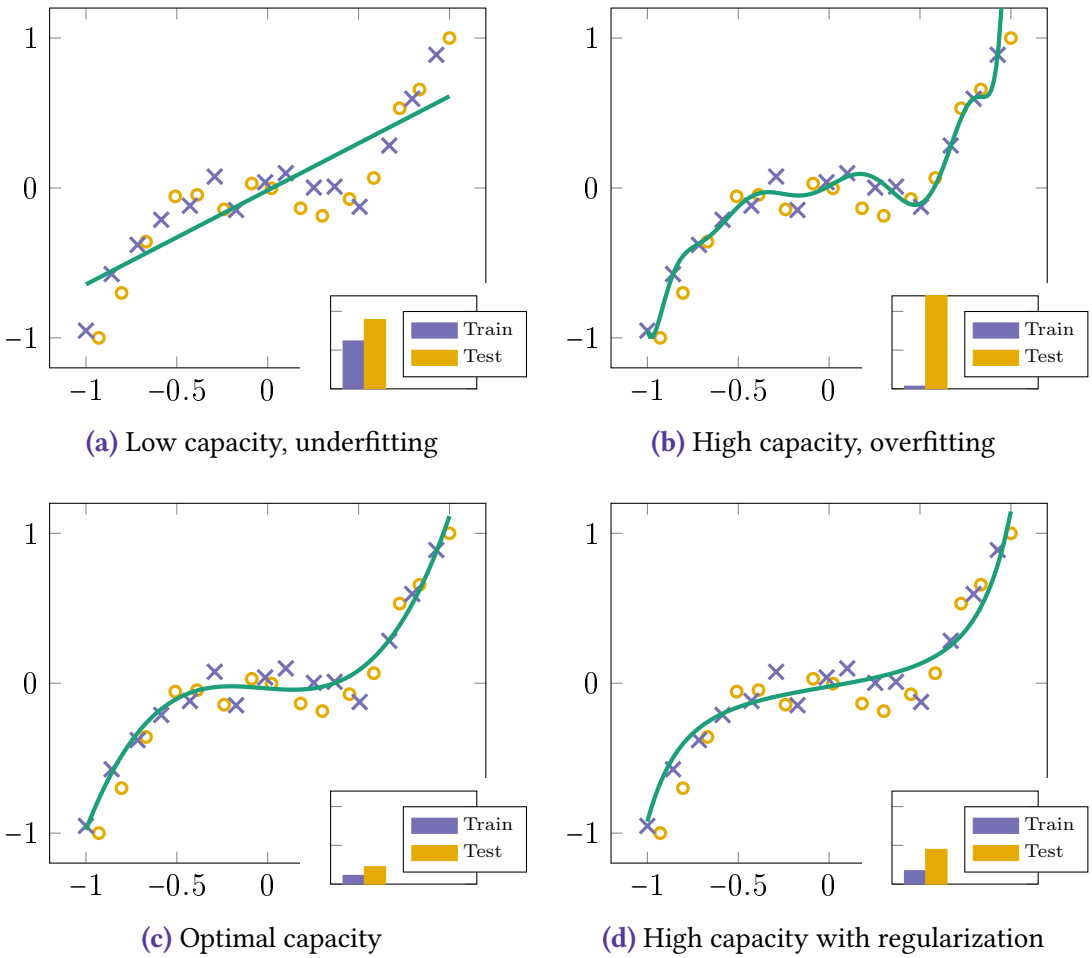
$$\hat{\theta} = \arg\min_{\theta} L(f_\theta, \mathbb{X}) \tag{2.1}$$

At the end of the day, most machine learning problems are reduced to a problem in mathematical optimization.

However, the single most important challenge that separates ML from optimization is that we want our model to perform well on new unseen data. The optimization problem that we are explicitly solving is only indirectly connected to our actual task. Good performance of our model on the training data does not guarantee good performance on new data samples. Therefore, there are two performance measures, or errors to consider. The *training error* measures how well our model performs on the training data, i.e., how well we solved our derived optimization problem. And the *generalization error* represents the expected model's performance on new data. Although we train the model by minimizing the training error, what we really want to optimize is the generalization error. One can typically estimate the generalization error by measuring its performance on a subset of the data that was not used for training, which is called the *test set*.

Assuming that our training set data is independently sampled from the same distribution as the test set, one could think that our model is expected to perform equally well on both. However, because we use the training set to choose the model, we actually expect the test error to be higher than the training error. The more the model specializes towards a particular training set, the larger this gap becomes. This is known as *overfitting*. On the other hand, if the model is not sufficiently utilizing the information from the training set, it is set to be *underfitting*, i.e., having a large training error. For example, in the extreme case we can choose a random model independently of the training set, which will result in the same but equally poor expected error for both sets.

One way of controlling the trade-off between over- and underfitting is by changing the model's *capacity*. Capacity can be seen as the model's ability to accurately represent large arbitrary datasets. More complex models with more parameters can typically fit more training data and thus have larger capacity. We can control the capacity by changing the function space of our parametric model. For example, a polynomial model has larger capacity to fit arbitrary data than a linear model. This is best demonstrated visually, as we do in Figure 2.4. A linear model in Figure 2.4a has low capacity and is unable to accurately fit the training set, underfitting it. The training error is high, but has a small gap to the test error: this model generalizes well to the unseen data, but is a poor model for this data distribution. A high-degree polynomial model in Figure 2.4b has larger capacity and can accurately fit the data, effectively interpolating it. However, it also captures structures that are specific to the training sample and introduces significant variance, thus overfitting the data. The training error is low, but the model fails to



(a) Low capacity, underfitting

(b) High capacity, overfitting

(c) Optimal capacity

(d) High capacity with regularization

Figure 2.4 — An illustration of under- and overfitting. Each subplot shows the training data (blue crosses), the test data (orange circles) and the training model (line plot). The bar chart shows the training and the test errors for each model. a: A linear model has low capacity and cannot fit the training data, resulting in underfitting, which is also evident from the high training error. b: A high-degree polynomial model has much higher capacity and overfits the training data. The training error is low, but the generalization error is much higher. c: A model with optimal capacity provides the best trade-off between fitting the training data and generalizing to unseen data. d: A regularization term can help adjust the capacity of the high-degree model and lead to better generalization.

generalize, which is evident from the high test error. The model in Figure 2.4c has an ideal capacity for this task and uncovers the underlying structure, while ignoring the noise. It strikes a good balance between fitting the training data and generalization,

thus resulting in the best test error of the three.

Another way of controlling the balance between over- and underfitting is *regularization*. Informally, regularization can be understood as providing a preference for some of the possible functions, i.e., parameter values. Unlike the hard restriction of the function space, which excludes a subset of possible solutions, regularization adds a soft constraint that increases the error when the preference is violated. Thus, undesired solutions should only be chosen, if the data strongly suggests it, i.e., if the error would be considerably higher otherwise. For instance, let us consider the polynomial function from above, which might look like $y = f_\theta(x) = b + w_1 x + w_2 x^2 + \dots w_n x^n$, where the weights w_i and the bias b are its parameters $\theta = \{\boldsymbol{w}, b\}$. We can add a term to the loss function that encourages weights $\boldsymbol{w}$ with a smaller L2-norm: $L_{reg} = L(\boldsymbol{w}, \mathbb{X}) + \lambda \boldsymbol{w}^T \boldsymbol{w}$. This is known as *L2-regularization* or *weight decay*. The coefficient λ is used to control the magnitude of the penalty, that is, how strongly the preference is enforced. In our example, regularization can help reduce or even remove the contribution from some of the polynomial terms when the improvement in the training error would be insignificant. In Figure 2.4d, we train this regularized model on the data from our previous example, and we see that it produces a nice balance between training and generalization, showing performance similar to the model with optimal capacity (Figure 2.4c). This demonstrates that regularization can provide a way of adjusting the effective capacity of the model based on the training data.

Of course, regularization does not completely free us from choosing appropriate capacity or function family of the model. In fact, we had to introduce an additional coefficient λ that now also needs to be selected. It is a *hyperparameter*, i.e., a value that is chosen as part of the model design, rather than learned from the training data. Choosing a very large λ can constrain the model too much, leading to underfitting, and vice versa. To select an appropriate amount of regularization or any other hyperparameter or even the model itself, we could compare their error on the test set and choose the one that "generalizes" the best. But this procedure would make our model dependent on the test set, thus introducing the same overfitting issue that we have with choosing the model parameters from the training data. For this reason, we introduce an additional set of held-out data, (confusingly) referred to as the *validation set*. The validation set can be used to choose hyperparameters during the model development, and then the untouched test set can provide an estimate of the generalization error.

Choosing λ, a regularization scheme, a model, an optimization procedure or any other aspect of the learning system introduces an *induction bias* – an implicit assumption about the data-generating distribution that we believe will help improve the generalization performance of our model. Without at least some assumptions, it is impossible to say anything certain about the data distribution from the training sample alone. For example, consider that given a finite sample of data points we can come up with an infinite number of functions that interpolate them. How do we choose one of them

as our model? Unfortunately, there is no silver-bullet solution to this problem and to achieving good generalization. Theoretical analysis shows that all models perform equally poorly when averaged over all possible data-generating distributions (Wolpert and Macready 1997). Luckily, we are not interested in all possible distributions, but only in ones that occur in practice. Therefore, the focus of machine learning research lies not in the development of a single perfect algorithm, but in understanding the structure of practical tasks and introducing appropriate inductive biases that lead to better generalization performance.

2.2.2 Neural Networks

Neural networks (NN) are a broad type of models that are the driver behind many of the recent successes in machine learning. Here we briefly summarize their structure, training and the reasons behind their effectiveness.

The core idea behind neural networks is to stack a number of simple repeating functions to form a more complex function. The most basic form of a NN is a *fully-connected* network, also referred to as the multi-layer perceptron. It consists of a series of *layers*, each being a composition of an affine function and a non-linear *activation function* (Figure 2.5). The output $\boldsymbol{a_j}$ of a j-th layer l_j can be written down as

$$\boldsymbol{a_j} = l_j(\boldsymbol{a_{j-1}}) = h(\boldsymbol{W_j}\,\boldsymbol{a_{j-1}} + \boldsymbol{b_j})\,, \tag{2.2}$$

where $\boldsymbol{W_j}$ and $\boldsymbol{b_j}$ are the learnable linear weights and biases, $\boldsymbol{a_{j-1}}$ is the output (activations) of the previous layer and $h(\cdot)$ is the activation function. Classically, layers are introduced as consisting of individual *neurons*, each computing a scalar function $h(\boldsymbol{w^T x} + b)$, but today it is more practical to think in terms of layers, since this is how NNs are conceived, constructed and computed. Thus, a basic neural network is a composition of its k fully-connected layers:

$$\begin{aligned} Net(x) &= l_k(l_{k-1}(\ldots l_1(x))) && (2.3)\\ &= h\left(\boldsymbol{W_k}\, h\left[\boldsymbol{W_{k-1}}\, h\left(\ldots h\left[\boldsymbol{W_1}\,\boldsymbol{x} + \boldsymbol{b_1}\right] + \ldots\right) + \boldsymbol{b_{k-1}}\right] + \boldsymbol{b_k}\right) && (2.4)\end{aligned}$$

The non-linearity introduced by the activation function $h(\cdot)$ is a critical component of a neural network. Without it, Equation 2.4 could be reduced to a single matrix multiplication and summation, i.e., to a linear (affine) model. Although any continuous non-linear function could theoretically be used as activation, the choice can significantly affect the training efficiency of the model. The early NNs used the *tanh* function, which was later replaced by the popular *rectified linear unit* function $ReLU(\boldsymbol{x}) = max\{0, \boldsymbol{x}\}$. Many other options are available, but before we can understand what properties a good activation function should have, we need to first review how neural networks are trained.

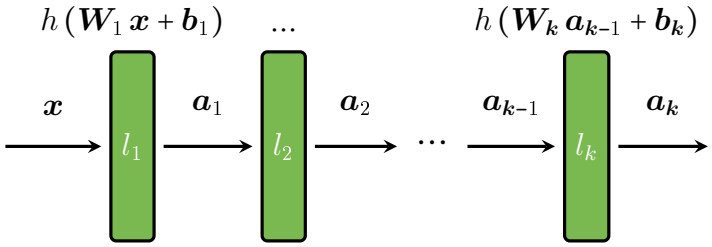


Figure 2.5 — A basic neural network consists of a series of layers $l_1, l_2, \ldots l_k$. Each layer performs an affine transformation $\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}$ and applies the activation function h to the previous layer's output $\boldsymbol{a}$ (activations). The output of the final layer $\boldsymbol{a_k}$ is the output of the network.

Training. Neural networks are often employed as a supervised model that learns a mapping from inputs $\boldsymbol{x}$ to outputs $\boldsymbol{y}$ from a training dataset $\{\boldsymbol{x_i}, \boldsymbol{y_i}\}$. The model is trained to optimize a loss function L that measures the deviation between the model's outputs and the target outputs $\{y_i\}$. The loss function is typically an average of the losses from each individual training points:

$$L_{train} = \frac{1}{N}\sum_{x_i,y_i} L\left(Net(\boldsymbol{x_i}), \boldsymbol{y_i}\right) . \tag{2.5}$$

Since the neural network is a complicated non-linear function, there is no general closed-form solution to minimizing even the most simple loss functions. Therefore, approximate iterative methods are used, with *gradient descent (GD)* being the most prevalent.

Gradient descent starts by choosing random initial values for the NN parameters. Then, we perform a series of parameter updates, where at each step we compute the gradient of the loss wrt. parameters and shift the parameters along the gradient direction, i.e., towards the steepest descent of the loss:

$$\boldsymbol{W_k^t} \leftarrow \boldsymbol{W_k^{t-1}} + \alpha \frac{\partial L_{train}}{\partial \boldsymbol{W_k^{t-1}}} , \tag{2.6}$$

where $\boldsymbol{W_k^t}$ are the parameter values at step t (we omit $\boldsymbol{b_k}$ for brevity) and α is the step size, also known as the *learning rate*. The learning rate is a critical hyperparameter that has a significant impact on the outcome of the training. A small learning rate can lead to slow convergence, and a large value can result in "stepping over" a good local minimum of the loss or even non-converging behavior.

The mention of a local minimum brings us to an important point. The gradient descent is a first-order method and is not guaranteed to converge to a global minumum of the non-convex function $L_{train}(\boldsymbol{W}, \boldsymbol{b})$ (Equation 2.5). In fact, considering the typically

large number of NN parameters and the highly erratic shape of the resulting loss function, reaching the optimum is extremely unlikely. Nevertheless, empirical results suggest that gradient descent outperforms more principled higher-order algorithms due to its computational speed. Computing the first derivative is fast and allows for training larger networks on larger data, which in turn leads to better generalization performance.

Still, practical gradient descent implementations rely on an additional set of modifications, some of which approximate higher-order derivatives without sacrificing performance. The first critical modification is the use of *Stochastic Gradient Descent (SGD)*. SGD is a modification of gradient descent where each update is performed using the gradient from a single data sample, as opposed to an average over the whole dataset. This is desired when the data is simply too large to compute the derivative each step and is thus very useful for training neural networks on big datasets. SGD can serve as a fast approximation of gradient descent, but introduces additional noise to the training process, which can result in slower or even nonexistent convergence. In practice, SGD is usually implemented as *Minibatch Gradient Descent*. At each step, we take a sample of b training points (called a *minibatch*) to compute the gradient and perform a parameter update. This provides a balance between the speed and stability, which can also be tuned using the hyperparameter b, known as the *batch size*.

Another common modification of gradient descent is *momentum*. The momentum method maintains a moving average of past gradients and uses the average to update the parameters at each step. This helps to both smooth out the noise introduced by SGD and to perform larger steps in the flat low-gradient regions of the loss function. Momentum should be seen as an approximation of the second derivative that is much cheaper than explicitly computing the curvature. Practically, momentum makes the training more stable and significantly faster, therefore finding place in most NN training implementations.

The final missing component for training a neural network is the actual computation of the loss gradient. A naive approach is to use the *finite differences* method. We can perturb each parameter, compute the resulting loss difference and thus approximate the derivative wrt. that one parameter. This approach, albeit generic, is very slow: for each parameter, we need to evaluate the whole network several times. A better method is to utilize the fact that the network is a composition of individual layer functions and apply the chain rule. The derivative of each layer's parameters wrt. the loss is dependent on the derivatives of the following layers, evaluated at the current input value to that following layer. Thus, we can first evaluate the whole network (*forward pass*) and store the activations (each layer's outputs). Then, we compute the derivatives of the loss wrt. to parameters, going layer-by-layer in reverse order, starting with the last layer (*backward pass*). Each time, we can reuse the derivative of the later layer to compute the derivative of an earlier layer. This algorithm is called *backpropagation* and it can also be

viewed as an instance of dynamic programming: each problem of finding a derivative is trivial given the solution to the derivative of the following layer. Backpropagation allows the gradient to be efficiently computed, requiring only some additional memory to store the layer outputs and the intermediate derivatives. Furthermore, implementation of backpropagation is made simpler today by numerous *automatic differentiation* software frameworks that seamlessly track the function compositions (the *computational graph*), store intermediate outputs and backpropagate derivatives.

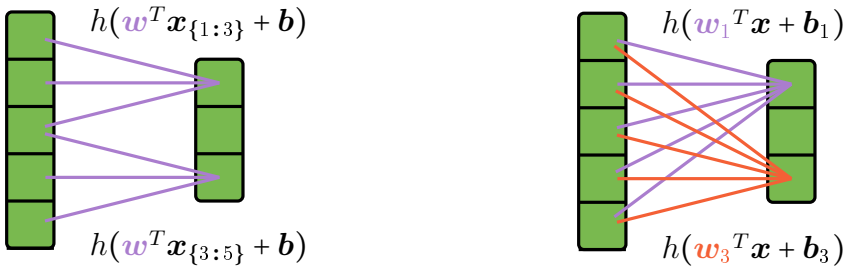
Now that we discussed how backpropagation is used to update the gradients, we can quickly come back to the activation function and the logic behind their selection. The "classical" functions *tanh* and *sigmoid* were inspired by the early theoretical results, which showed that a neural network can approximate any continuous function (on a compact subset), given sufficient size and a bounded monotonically-increasing activation function (Hornik et al. 1989). The main issue with bounded monotonic functions is that they have regions where the derivative is small as it converges to zero. Compounded with the numerical issues caused by repeatedly multiplying small derivatives in backpropagation, the bounded activation functions lead to the problem of *vanishing gradients*. Some of the neurons can end up in regions with virtually no gradient, slowing down the training. Later, it was shown that the universal approximation can also be achieved by an *un*bounded non-polynomial function. In practice, the *ReLU* function ($max(0, x)$) became a popular unbounded solution to the vanishing gradients, due to its computation efficiency and strong empirical results. However, the *ReLU* neurons are prone to "dying", since there is a region of the parameter space where they have zero gradient across all data. This is typically addressed by making it unbounded also on the negative side, thus introducing a small gradient, e.g., *LeakyReLU* or *PReLU*. Recently, functions like *GeLU* and *Swish* are getting increased attention, as they are continuously differentiable, have a non-zero derivative, but are still bounded on the negative domain. This prevents the issue where a severely negative neuron, which should be turned off, overrides positive contributions at the next layer. Overall, activation functions are an ongoing area of research with many alternatives still being discussed. In practice, one should monitor the gradients of the model for potential numerical issues, introducing changes if necessary. But more importantly, like any hyper-parameter of the NN, the activation function should be selected empirically on the validation set.

Training setup. Having discussed the two most important parts of neural network training – stochastic gradient descent and backpropagation, let us examine a typical training setup. Continuing with the supervised case, we have a training dataset of the form $\{\boldsymbol{x_i}, \boldsymbol{y_i}\}$ and our goal is to learn the mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$. First, we need to split the data into a training, validation and test sets. It is crucial to sample the test data early and to avoid both explicit training and informing hyperparameter choices using the test data. Otherwise, we condition the final model on this sample and make it a

poor evaluation of the final generalization performance. Next, the training data needs to be normalized to have zero mean and variance of one. Normalization helps keep the layer outputs centered and close to zero, making it easier for neurons to be turned on or off during early training, and ultimately improving the training convergence. In some cases, unnormalized data can saturate a significant number of neurons and completely hinder the training process. Next, the network parameters needs to be randomly initialized, and for similar reasons, the initialization needs to be designed such that the layer outputs follow the standard distribution, given normalized inputs. In this way, all the layers of the network will observe a diverse distribution of inputs, leading to better training.

After normalization and initialization, we can proceed to optimization. Practical mini-batch gradient descent is usually implemented not by randomly sampling minibatches, but by splitting the whole dataset into batches, each getting "shown" to the network once, i.e., sampling without replacement. This is not strictly necessary, but can produce more stable results by making sure that the model observes each data point regularly. For each minibatch, we compute the gradient and update the model parameters. We keep iterating through minibatches until making a full pass through the dataset. A full pass is called an *epoch*, and once an epoch is complete, a new one is started, going over the same data again. However, the training data should be shuffled in-between epochs to prevent the model from being biased towards a particular order of data. The training is terminated after a pre-determined number of epochs has elapsed or when the training loss has stopped decreasing. A better approach is to monitor the validation loss (e.g., computing it once per epoch) and terminating training once it stopped improving. This is called *early stopping* and acts as a form of regularization, counteracting the tendency of neural networks to overfit towards the end of the training. The complete training process can be repeated multiple times to compare different hyperparameter values and model architectures (on the validation dataset), or to measure the training uncertainty. And as the last step, we can evaluate our best trained model on the test data, getting a final unbiased estimate of its generalization error.

Deep Networks. Overall, various forms of deep (many-layer) neural networks have demonstrated strong performance on a large number of practical tasks. Their ability to generalize well even when applied to high-dimensional data is usually attributed to their compositional structure. Each layer is able to reuse the computations of the previous layers and applies its own space partitioning on top of the previous partitions, which results in an exponential number of input space regions that can be efficiently identified by the network (Montúfar et al. 2014). However, there is still no strong theoretical framework to understand and predict the NNs generalization behavior, and much of the progress in the field is driven by empirical results. The philosophy of modern deep neural networks is to prioritize high computational performance through composition



(a) Convolutional **(b)** Fully-connected

Figure 2.6 — Schematic comparison of a convolutional and a fully-connected layers. a: The output components of a convolutional layer are computed from local regions of the input, unlike the fully-connected layer on the right (b). Furthermore, the convolution uses the same trainable filter w, while each fully-connected output learns its own transformation (w_1, w_3). Locality and positional independence are strong assumptions, but hold for images and other grid-defined data, significantly reducing the computational costs.

of simple repeating components. In fact, many researchers attribute the many recent successes of neural networks primarily to the increase in the available computational resources (Amodei and Hernandez n.d.; Thompson et al. 2020). Brute force compute power tends to beat carefully designed algorithms over time (Sutton 2019).

2.2.3 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a type of a neural network that specializes on processing data defined on a grid: images, videos or, in our case, simulation outputs. The namesake component of a CNN, the *convolutional layer*, performs a discrete convolution of its input with a small learnable filter, which gives it two key properties. First, each component of the output depends only on a small region of the input, as opposed to a fully-connected layer (Section 2.2.2), where each output component is computed from the whole input (via a matrix multiplication). In Figure 2.6a, we show a schematic depiction of a one-dimensional convolution. Notice that the first component of the output is computed from only the first three inputs, unlike in Figure 2.6b. Essentially, the convolution introduces a strong assumption that nearby locations in the data are related to each other but unrelated to far away locations. This locality is common to grid data, e.g., to photos, where nearby pixels often belong to the same object. The second property is that the same convolutional filter is applied everywhere, thus each output component computes the same function but over different input locations. In Figure 2.6a, all the outputs are using the same trainable filter $\boldsymbol{w}$. In contrast, for a fully-connected layer, each output component (neuron) learns an individual mapping

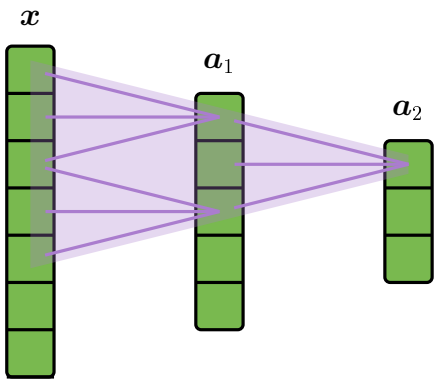


Figure 2.7 — Each convolutional layer aggregates information across the spatial dimension of its input. The aggregation thus compounds as we move deeper into the network, with later layers being affected by a larger region of the original input $\boldsymbol{x}$. This input region is called the *perceptive field*.

from the input (Figure 2.6b). Once again, this positional independence is a suitable assumption for images, since we want to apply the same basic processing to all locations (e.g., detect edges, circles, etc.) regardless of their absolution position. A translated and flipped photo is still a valid photo, unlike a sentence, in which the position of a word can be critical.

A convolutional layer usually applies not one, but many trainable filters. Each filter is convolved with the input independently from the others and the results are stacked together to form the layer's output. Furthermore, a CNN is built from multiple layers, and each applies convolutions to an output from a previous layer. Because every layer aggregates information from a region of the previous layer, the aggregation compounds, and the output of the later layers gets affected by an increasingly large area of the original input (Figure 2.7). Thus, the early layers are learning to extract highly local, low-level, position-independent features that get incrementally aggregated into large-scale high-level information. In practice, the network is often structured with this process in mind, reducing the spatial dimensions with each consequent layer by adding *stride* to the convolutional layers. A stride of two, for example, means that only every second output position is actually computed, reducing the spatial extent of the output by half. The later layers are also given a larger number of filters, so the network is effectively converting spatial dimensions into feature dimensions. Depending on the task, the very last layers can be made fully-connected, completely eliminating the spatial dimensions.

Convolutional networks are designed around the idea of hierarchically applying simple trainable filters, which enables CNNs to process large spatial data, while keeping the model size relatively small. Interestingly, CNNs are a subset of fully-connected

networks, which can theoretically learn the exact same *sparsely-connected* structure of a CNN. Nevertheless, fully-connected networks generally perform worse on image tasks, in part due to their computational costs, and in part due to having to learn what is already built-in into CNNs. Thus, convolutional networks are a good example of an appropriate *inductive bias* – they make a set of useful task-specific assumptions that make it easier to learn the underlying structure of images and other grid-defined data.

2.2.4 Machine Learning Research Terminology

This thesis frequently uses various machine learning terminology (thus this background chapter), but no concept is brought up more often than *Self-Supervised Learning* (SSL). The motivation for SSL comes from the common observation that most of the data is unsupervised, since labeling the data is difficult, expensive and for some tasks even impossible. Given the large amounts of "unused" unsupervised data, can we learn anything from it that can help improve our models? This question has existed for decades and is connected with the development of many related families of methods: unsupervised learning, representation learning, transfer learning, semi-supervised learning, self-supervised learning, multi-task learning, zero/few-shot learning and likely some more. This abundance of terminology and its inconsistent usage can be sometimes confusing. Thus, we would like to first clarify what the terms signify and how they relate to self-supervised learning.

Unsupervised learning is the broadest of the concepts and probably the most familiar to the reader, as it appears in virtually every introduction to machine learning (including ours in Section 2.2.1). The term is defined by its contrast to supervised learning where the model is learning to map inputs to outputs, and the correct input-output pairs are available as training data. This is commonly formalized as learning the distribution $p(\boldsymbol{y} \mid \mathbf{x})$, with $\mathbf{x}$ being the input and $\boldsymbol{y}$ the output. In unsupervised learning, the outputs are not available and the goal is to learn something about the distribution $p(\boldsymbol{x})$. This can take the form of density estimation, i.e., explicitly modeling the probability density function $p(\boldsymbol{x})$, or training generative models to produce new samples from $p(\boldsymbol{x})$. There is however a subtle ambiguity in how the term "unsupervised learning" is used. In some cases, it means unsupervised learning tasks where we want to learn about $p(\boldsymbol{x})$, e.g., generative modeling – a pure unsupervised task. And in others, it refers to unsupervised methods, which can sometimes be used for supervised tasks that lack supervised data, for example, in zero-shot learning we still want to learn the $p(\boldsymbol{y} \mid \boldsymbol{x})$ and classify the inputs, but we lack the labels for the classes of interest.

Representation learning is less of a taxonomical term as it refers to a more specific problem, namely, finding a good data representation. What it means for a representation to be "good" can differ depending on the context. Sometimes, the goal is to find a representation that has certain statistical properties, e.g., independence. Or we might

want to extract features that improve the performance on a particular task. The general observation is that some representations are more suitable for certain tasks, for instance, an average person might find it easier to multiply numbers in decimal form rather than in hexadecimal. Representation learning is typically unsupervised in the sense that we do not have a dataset of "correct" representations, but instead rely on some generic prior or regularization. For example, a sparse autoencoder is learning to map data to a representation that is constrained to be small and sparse. Representation learning can sometimes be called supervised when the representation is learned as a part of a supervised task, e.g., the success of deep learning is attributed in part to deep neural networks learning an efficient representation. (Montúfar et al. 2014). The early layers transform the inputs into a representation where a supervised task can be solved by a linear classifier. Some "pure" representation learning methods seek such mappings explicitly, for instance, aiming to cluster data points of the same class in the learned feature space (Mairal et al. 2008). In these scenarios, the representation itself is still technically unsupervised, but it is adapted to some supervised data.

Transfer learning is concerned with reusing what has been learned on one task to help improve the performance on the other (Pan and Yang 2010). Generally speaking, the tasks can be represented as any two related probability distributions p_s, p_t, but the most common situation is that both are supervised and require different outputs for the same inputs, i.e., the source distribution $p(\boldsymbol{y_s} \mid \boldsymbol{x})$ and the target $p(\boldsymbol{y_t} \mid \boldsymbol{x})$ (other scenarios are also possible, e.g., in domain adaptation outputs are similar, but inputs differ). The idea of transfer learning is that the knowledge about the source distribution can also be helpful for modeling the target (Goodfellow et al. 2016). This could be exploited in many different ways, e.g., by transforming data from one domain to the other, or constraining the target model to be similar to the source. See the survey by Zhuang et al. 2021 for an overview. One popular approach is to perform representation learning on the first task and use the representation to fine-tune a model on the other task. If the tasks are sufficiently similar, the appropriate representations should be similar as well, preserving the same information in a compatible form.

In a *semi-supervised learning* setting, we have a partially labeled dataset where only some of the samples have labels. The goal is to solve the supervised task of estimating $p(\boldsymbol{y} \mid \boldsymbol{x})$ using both labeled samples from $p(\boldsymbol{x}, \boldsymbol{y})$ and unlabeled samples from $p(\boldsymbol{x})$. The underlying assumption is that the structure of $p(\boldsymbol{x})$ is useful for modeling $p(\boldsymbol{y} \mid \boldsymbol{x})$, for example, due to samples of the same class being nearby in the input space (Chapelle et al. 2006). This connection then can be exploited, for instance, by learning a representation of $\boldsymbol{x}$ that separates the clusters and then training a linear classifier on top (Chapelle et al. 2003). Or by training a model with both a discriminative (on $p(\boldsymbol{y} \mid \boldsymbol{x})$) and generative loss (on $p(\boldsymbol{x})$), (Lasserre et al. 2006). Semi-supervised learning can be considered as a specific class of transfer learning, where we transfer knowledge from an unsupervised to a supervised task.

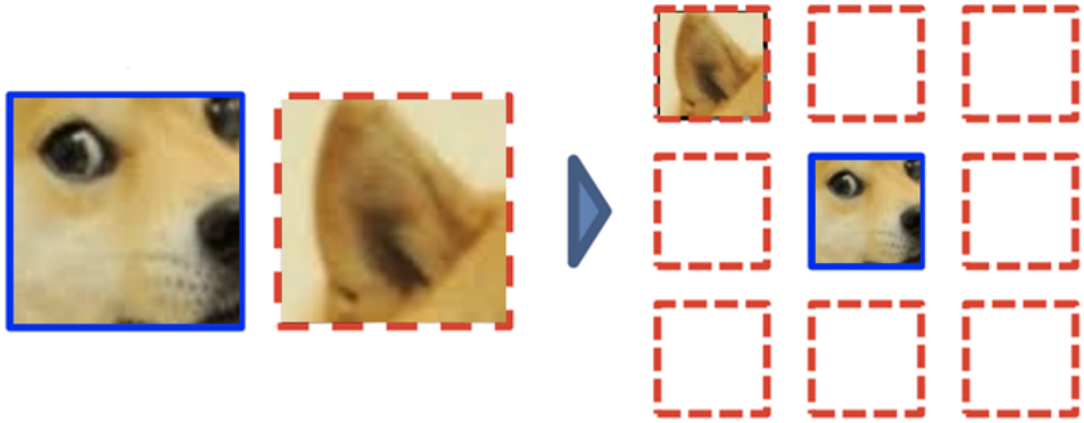


Figure 2.8 — The self-supervised pretext task of predicting relative positions of image patches. Given an image, two patches from a predefined pattern are extracted (on the left). The model's task is to predict the second patch's relative position, formalized as an 8-way classification problem (right). By solving the task, the model learns a feature space that is useful for detecting and relating parts of different objects.

Finally, *self-supervised learning* is a more recent and specific approach of learning representations from unsupervised data. One defines an auxiliary supervised task (a.k.a. the pretext task) on the unsupervised data, by deriving the supervision signal automatically from the data structure or metadata. For example, the task could be to predict the next word in a sentence or to correctly arrange parts of an image. The training data can be generated by masking parts of a sentence or shuffling image patches. Next, a supervised model is trained to solve this task, learning a useful representation in the process. The representation is then used to help with another (typically supervised) task, called the downstream task. Thus, self-supervised learning is related to semi-supervised and transfer learning, and shares the same motivating assumption that the two data distributions – of the pretext and downstream tasks – share some underlying structure. And more specifically, that the representation learned under the pretext task is also useful for solving the downstream task. Therefore, SSL is also referred to as self-supervised representation learning. Summing it all up, self-supervised learning is about applying supervised methods to solve an (unsupervised) representation learning task and then transferring the representation to a downstream supervised task. So in jest, SSL can be called supervised unsupervised representation transfer learning. But practically, it is simply an umbrella term for methods that emphasize the use of an auxilliary task on unlabeled data to learn a good representation.

2.2.5 Self-Supervised (Representation) Learning

The idea of self-supervised learning has been around for some time in the field of natural language processing. Word2vec (Mikolov et al. 2013b) is the most prominent example, where the pretext task is to predict the surrounding words for each word in a sentence. A model can be trained on a large unsupervised corpus learning a word embedding (representation) as a result. The learned embedding would then be used as a part of other models to help with downstream tasks. It took some time before the concept was studied closely in other application areas, likely because learning a representation for words is in some sense necessary, the alternative being dictionary-sized one-hot encoded vectors that all have the same distance to each other. Images, for example, already have somewhat useful distances, so the need for representation learning is less pronounced (Goodfellow et al. 2016).

Nevertheless, self-supervised learning gained popularity specifically through its applications in computer vision problems, where many pretext tasks were proposed over the years. For instance, Doersch et al. 2015 extracted patches from an image and trained a model to predict their relative position (Figure 2.8), and Noroozi and Favaro 2016 have tasked their model with solving a jigsaw puzzle of image patches. To correctly predict the spatial arrangement of a patch, the model needs to learn visual features that discriminate and relate different parts of the objects depicted in the images. The same features are likely to be a good representation for solving other visual problems, e.g., classifying the objects. For an overview of what other visual tasks exist see surveys by Doersch and Zisserman 2017; Jing and Tian 2020.

In recent literature, self-supervised learning is sometimes classified into generative and contrastive (Liu et al. 2021). The former includes auto-regressive models that predict future values in sequential inputs. As an example, language models like GPT-2 (Radford et al. 2019) predict future words and sentences for text sampled from large text corpora. And for images, one can mask a part of it, and train the model to fill-in the missing fragment (Pathak et al. 2016). This idea is similar to the de-noising autoencoder (Vincent et al. 2008) – another generative example, as it learns to recover the original input from its noise-corrupted version, essentially finding a noise-invariant representation. All of these methods are considered generative because the pretext task is related to generating the original data or its parts. In contrast, contrastive methods learn the representation by comparing the inputs, e.g., predicting if two images contain the same object or not. Since this is the approach we favor in the following chapters, we review it in more detail below.

2.2.6 Contrastive Representation learning

The aim of contrastive representation learning is to obtain a good representation by learning to compare input samples. The idea is that the inputs that are similar should

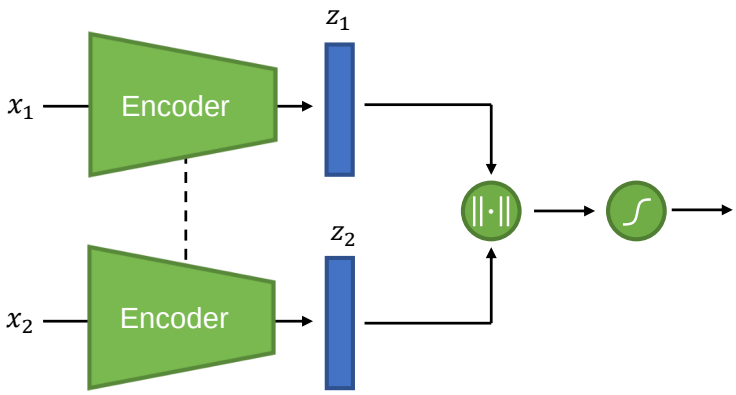


Figure 2.9 — The architecture of a basic siamese model. Two inputs x_1, x_2 (e.g., images of a handwritten signature) are each passed through an identical encoder, producing the representations z_1, z_2. A distance between the representations is computed and used to output the probability of the two inputs being similar (e.g., the signature belonging to the same person).

be mapped to similar representations, and dissimilar inputs should be pushed further away in the feature space. This is where the name contrastive comes from: the model is learning to contrast similar and dissimilar pairs.

One of the earliest examples of contrastive learning is the siamese neural network introduced in Bromley et al. 1993 and Baldi and Chauvin 1993 to solve the problem of signature and fingerprint verification. A siamese network consists of two identical encoders that each transform an image into an encoding vector (Figure 2.9). Then, a cosine distance between the two vectors is used as a prediction of whether the two images contain the signatures belonging to the same person. The model is trained by sampling valid and invalid pairs from a supervised dataset of signatures and a loss that is minimized when valid pairs have a distance of one, and invalid pairs distance of minus one.

A more explicit example of contrastive learning is the work of Schroff et al. 2015, who proposed the triplet loss. The authors constructed a facial verification model that works by learning a representation for input images in which images of identical faces are placed closely in the feature space. The model is trained by sampling triplets of images consisting of an anchor, a positive example and a negative one. Then, the loss pushes the positive example closer to the anchor and the negative is pushed away. After the training, face similarity can be derived directly from the distances between the encoded images.

These two methods are both traditional examples of supervised learning – they utilize a dataset of valid face/signature pairs to generalize to new examples. Koch 2015

demonstrated that siamese networks can also be used for one-shot learning. One-shot learning is an extreme case of transfer learning, where one has only a single example of the target classes to correctly classify images among them. Just as before, they train a siamese model by sampling pairs of data points that belong to the same or different classes. Specifically, the data comes from the Omniglot dataset (Lake et al. 2015), containing images of hand-written characters from 50 different languages. When testing, the model is presented with a single example for 20 characters from previously unseen alphabets and tasked with classifying a test image among them. This is achieved by returning the character whose example image shares the most similarity with the test image in the learned feature space. The model is still trained on supervised data by distinguishing different classes, but tested on the new task of distinguishing among unseen classes (transfer learning). In Section 4.3, we introduce a similar few-shot learning method, but extend it further, training the siamese model without the supervised data, using a self-supervised pretext task.

The approach of using self-supervised contrastive tasks became popular in the last few years, as it benefits from the large amount of unsupervised data and was shown to outperform purely supervised models. Instead of using a supervised dataset and training to contrast different classes, we use a self-supervised approach and generate classes automatically through data augmentation. For instance, images can be distorted, rotated, cropped, etc. to generate related samples, effectively producing a new quasi-class for each original image. We can then learn an image representation by training a model to distinguish instances of the same quasi-class.

A recent application of this idea is SimCLR (Chen et al. 2020). In SimCLR, a batch of images is sampled from an unsupervised dataset and each undergoes two different random transformations. The transformation is defined as a combination of color distortion and cropping. Transformed images are then encoded into the feature space using a shared encoder (like in a siamese network). The encodings are then passed through a small projection block (a pair of fully-connected layers with a non-linearity) and the distance between them is used to predict which of the other transformed images was the same image originally. So, given a batch of N images, the model is solving a $2N$-way classification problem. The learned representation was evaluated by fine-tuning the whole model or adding a linear classifier on top and training on supervised task, showing that it outperforms purely supervised methods. In Chapter 5, we use SimCLR to learn an image similarity metric and also extend it to visual styles.

3

Detecting Irregular Behavior in Spatiotemporal Volumes

This chapter is based on the following publication:

Gleb Tkachev, Steffen Frey, and Thomas Ertl (2021a). "Local Prediction Models for Spatiotemporal Volume Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 27.7, pp. 3091–3108

When it comes to the analysis of spatiotemporal volumes, the most popular approach is still to manually browse through the individual timesteps or to use animation. This allows one to look at the full spatial information and apply well-known rendering and interaction techniques. However, relying on the animation alone has been shown to be ineffective, as only a limited number of frames can be memorized by an observer (e.g., Joshi and Rheingans 2005). Furthermore, smaller details that are often crucial can be easily missed in the bulk of the dataset. A logical alternative would be to generate a static summary of the data or to select its most important parts, but this is a challenging problem due to issues ranging from occlusion and visual clutter to performance restrictions. Even more problematically, filtering and aggregation of data might necessitate specialized assumptions that may exclude the very details we wanted to preserve. For instance, while temporally dense techniques visualize all timesteps, these typically restrict themselves to specific (user-defined) features to circumvent the occlusion problem (Balabanian et al. 2008; Joshi and Rheingans 2005; Liu et al. 2017). And selecting a subset of timesteps for visualization (e.g., Lu and Shen 2008; Tong et al.

2012; Frey and Ertl 2017a) requires either data-specific selection criteria or involves the costly explicit quantification of timestep differences, and interesting process transitions may still be missed.

In this chapter, we present how ML models can be used to help with this problem by detecting irregular regions in spatiotemporal data. Instead of explicitly defining indicators of irregularity, we aim to learn what is typical behavior for a given dataset by training neural networks to predict the data. This provides an important advantage of avoiding domain-specific assumptions, making the method much more generic. Once the models are trained, we investigate the spatiotemporal locations where their prediction deviates from the actual data. This can happen for a number of reasons, but all of them indicate irregularity that would be of interest to the analyst, e.g., outliers or uncertainty. Thus, this information can be used to aid in spatiotemporal visualization, both explicitly, to highlight the irregular regions, but also in other applications, such as automatic timestep selection.

3.1 Related Work

Time-varying data visualization. A large body of work in time-dependent volume visualization is based on feature extraction. Time Activity Curves that contain each voxel's time series have been used in several techniques (e.g., Fang et al. 2007; Lee and Shen 2009). Lee and Shen 2009 extract trend relationships among variables for multifield time varying data. Wang et al. 2008 extract a feature histogram per volume block, characterize the local temporal behavior, and classify it via k-means clustering. Based on similarity matrices, Frey et al. 2012 detect and explore similarity in the temporal variation of field data. We also perform temporal feature extraction, however, we do not manually define our features, but learn them from the data.

Dutta and Shen 2016 use Gaussian Mixture Models for tracking user-defined distribution-based features in volume data. Tzeng and Ma 2005 apply neural networks to generate adaptive transfer functions based on key frames. In contrast, our prediction models are unsupervised. Muelder and Ma 2009 also use prediction, utilizing an analytical predictor-corrector approach to track feature regions. However, we learn prediction mappings from the data and perform prediction directly on data values, without prior feature or predictor definitions.

Another line of work uses the notion of a space-time hypercube to apply operations like temporal transfer function in Balabanian et al. 2008 or slicing and projection techniques in Woodring and Shen 2003 (see Bach et al. 2016 for an overview). Tong et al. 2012 use different metrics to compute the distance between datasets and employ dynamic programming to select the most interesting timesteps. Based on a similar concept, Frey and Ertl 2017b generate a distribution-based distance measure to select timesteps. In

a follow-up work (2017), they use neural networks to estimate this distance metric for time series data. Our technique also allows for adaptive timestep selection, but it relies on prediction-based temporal irregularity rather than distribution-based distance metrics.

Information theory in visualization. Information theory has recently been gaining attention in visualization research. Bordoloi and Shen 2005 use an entropy-based feature for view selection in volume rendering. Viola et al. 2006 present an approach to automatically focus on objects in volumetric data by minimizing mutual information. Chen and Jaenicke 2010 as well as Wang et al. 2011 provide an overview and discuss the applicability of information theory in visualization. Related to our approach, Jänicke et al. 2007 use the domain-agnostic information-theoretic notion of statistical complexity that estimates predictability of spatiotemporal regions. As opposed to using a probabilistic definition of predictability, we train an actual predictor on the data, measuring the error of the deviation as an estimate of irregularity. This allows us to not only benefit from higher-level features and generalization behavior of ML models, but also to use multiple different models to produce a diverse visualization.

ML in scientific visualization. Machine learning has long been considered to have great potential in visualization (Ma 2007). Originally, it found most applications in the area of Visual Analytics (survey in Endert et al. 2017). For example, Fuchs et al. 2009 demonstrated how to foster interaction between a human analyst and a genetic learning algorithm. And in epidemiological analysis, Klemm et al. 2015 employed decision trees to study relationships between image shape descriptors and non-image features. Unsupervised learning via self-organizing maps (Kohonen 1990) has also been popular. For instance, Andrienko et al. 2010 investigated how SOMs can be integrated into the visual analysis process of spatiotemporal data and Sacha et al. 2018 presented a VA approach to analyze time series data using SOMs.

In recent years, learning-based approaches have been applied in scientific visualization. Some have focused on efficiently representing and interpolating volume data. For example, Zhou et al. 2017 presented a CNN-based approach to upscaling volume data, while Han and Wang 2020 employed a recurrent generative model to interpolate in the temporal, and later, in both spatial and temporal dimensions (Han et al. 2021). Lu et al. 2021 took a different approach, and used a neural network to represent the data itself, storing the model weights as a lossy compressed version of the volume. Jakob et al. 2021 generated a large training dataset and learned an interpolant for vector field data. Finally, Shi et al. 2022 trained a surrogate model for parameter space exploration of ocean simulations.

Other works aimed to learn the volume rendering function. Berger et al. 2019 developed a neural network to generate and explore volume-rendered images. Similarly, Hong et al. 2019 utilized an adversarial framework to restyle and generate new renderings from

existing images. The method by He et al. 2019 is related to both groups, as they learn to generate volume rendering images under interpolation of both visual and simulation parameters. The above works focus primarily on the rendering pipeline, while we train models that support exploration and navigation of the data. Another use of adversarial methods was presented in He et al. 2020b, performing comparison for collections of ensembles that represent different simulation models. Han et al. 2018 have shown that autoencoders can be used to learn a latent space for the analysis of streamlines and streamsurfaces. Recently, Guo et al. 2020b used LSTM autoencoders with attention to embed and find similarities in sequential medical data. The autoencoder objective is related to self-supervised learning and thus to our method from Chapter 4. But in contrast, we develop a new self-supervised task suitable for similarity searches and focus on staying domain-agnostic, handling varied ensembles of spatiotemporal volume data. Zheng et al. 2021 proposed an explicitly self-supervised approach, pretraining a model for medical image segmentation and classification. However, they used a smaller labeled dataset to fine-tune the pretrained model, while our approach in Chapter 4 is trained fully unsupervised.

A few related ML-based approaches were investigated in the field of video analysis (preliminary results in Hassanien et al. 2017; Gygli 2017), where neural networks are used to classify video frames into normal frames and shot boundaries. In this chapter, we also utilize machine learning models to detect irregular events but for spatiotemporal visualization. More importantly, we do not make any assumptions about the events in the data, using prediction error as an indicator of irregular behavior, taking an unsupervised approach (as opposed to training the model to explicitly classify behavior as regular/irregular on supervised data).

3.2 Prediction-based Irregularity Detection

This approach is aimed at the detection of irregular events in spatiotemporal volumes without making domain- or dataset-specific assumptions. We achieve this by using generic ML models that are trained under a very general task. Specifically, we train a set of models to predict future data values based on the past. Once our models are trained, we evaluate them on the data and compare their prediction to the actual data. The prediction difference that we obtain is itself a spatiotemporal volume, which we use as an indicator of irregular behavior. Futhermore, we use multiple models of different capacity as detectors of different "sensitivity". Simpler models can only predict the most basic behavior and fail often, while more complicated models produce more compact and sparse regions of inaccurate prediction. An overview is presented in Figure 3.1.

High prediction error in a particular region tells us that the model failed to capture the local behavior. There are several reasons why this could happen: the behavior is too

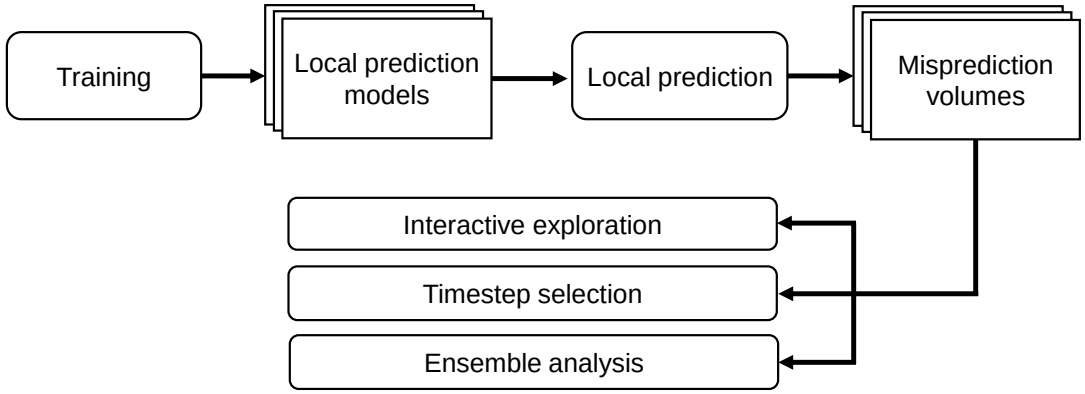


Figure 3.1 — An overview of our approach. We begin by training several local prediction models on a dataset. Then we perform prediction for the same data and obtain spatiotemporal misprediction volumes (error). We use the results for several applications, including exploration via spatial and temporal views, adaptive timestep selection and ensemble dissimilarity analysis.

complex to be expressed by the model; the behavior is rarely observed in the dataset (an outlier); or the available information is insufficient to predict the outcome. This is investigated in more detail in Section 3.3. We argue that these scenarios describe events of importance to the analyst and can be used to guide the visual exploration process, highlighting interesting events and providing a meaningful overview of the dataset, acting as a starting point for a more detailed and specialized analysis.

Once the prediction errors are collected, there are several ways to exploit them in visualization. Their primary application is to present irregular spatiotemporal regions directly to the analyst, which can help to explore the most interesting events in the data. We discuss these techniques in Section 3.5. However, this is not the only application (Figure 3.1). The prediction errors can also be used to perform automatic timestep selection (Section 3.6), and in the next chapter, we will describe an ensemble similarity metric based on the prediction models (Section 4.2).

3.2.1 Prediction Model

In principle, any model can be used as a predictor. However, the accuracy of the prediction plays an important role: if the model cannot capture even the most trivial behavior, we gain no additional information from the analysis. For simpler datasets, analytical models can be used, repeating past values or estimating local derivatives to do extrapolation. But as the data grows more complex, we require increasingly complex models, which inevitably implies additional assumptions and specialization. In contrast,

we want our models to be uniformly applicable to datasets from different domains, and thus take a data-driven approach, using machine learning methods to learn local behavior from the data.

Specifically, we opted to use neural networks with simple architectures, utilizing only convolutional and densely-connected layers. This provides us with a number of advantages. First of all, neural networks can represent a large number of functions and can be efficiently trained on large data. Second, simpler neural networks have fewer hyperparameters and allow for easy and gradual adjustment of the model capacity by changing the number of layers and neurons. More advanced network architectures, e.g., using recurrent connections, often require adjusting several related components together, each of which may have a complex impact on the prediction. Simple networks allow us to reason about the models in terms of their overall capacity, rather than qualitatively different architectures. Most importantly, by using basic densely-connected and convolutional layers we make as few dataset- or domain-specific assumptions as possible. Convolutional layers are basic locally-connected layers that exploit spatial coherence, which is a reasonable assumption for most types of scientific data. Nevertheless, we use networks both with and without convolutions, demonstrating their appropriateness for the task.

3.2.2 Local Prediction Problem

One of the key characteristics of our method is that we opted for a local approach to prediction, i.e. the model predicts each future value based on values that occurred in the spatial neighborhood of the point in preceding timesteps. In principle, a different path could be taken, for example predicting the full field belonging to a timestep. However, there are a number of advantages to the local approach. Most importantly, locality removes positional information from the input data, making the learned prediction mapping translation-invariant. First, this makes the results more intuitive: equivalent behavior occurring in different locations in the data produces an equivalent effect on the prediction error, and thus has the same impact with respect to the visualization. Second, it simplifies the prediction problem, which means that much simpler models could be used for analysis: the model does not need to learn the same behavior in different locations separately. The local assumption also makes it less likely that the model will simply "memorize" the dataset, overfitting it heavily. A simple example would be an object that spontaneously appears: a local model has no way of anticipating that event. It has to predict that empty space leads to empty space, since that is what happens 99% of the time. If the model had full positional information, it could associate certain "landmarks" from distant regions of the data with the object appearing, effectively overfitting the dataset, requiring a lot of care with model regularization and validation. Local prediction also provides several performance gains. Using smaller patches and

simpler models means that we require less training data and update fewer parameters, speeding up the training process. It also helps avoids exceeding the GPU memory, which is a common problem when passing large fields through a neural network. Finally, finer data partitioning increases the parallelization potential both during training and prediction. For further discussion see Section 3.10.

In formal terms, local prediction means that our model, given a voxel value $D(p,t)$ at the spatial position $p = (x,y,z)$ in timestep t of a dataset D, is trained to perform the following mapping:

$$Patch(p,t,l_s,l_t) \rightarrow D(p,t+1) \tag{3.1}$$

where $Patch(p,t,l_s,l_t)$ is a spatiotemporal box with spatial extent l_s and temporal extent l_t centered in space around point p. Note, that unlike the spatial extent l_s, the temporal extent l_t covers only one direction, thus the input includes only the past values, and none from the future.

For additional control over the difficulty of prediction, we generalize the problem to predicting not one, but d timesteps ahead. Thus, we add a delay between the input patch and the target value:

$$Patch(p,t,l_s,l_t) \rightarrow D(p,t+d), \quad d \geq 1 \tag{3.2}$$

This extension is useful for datasets with high temporal resolution. Increasing the value of d makes learning of simple mappings (e.g., repeating the most recent value) less feasible, forcing the model to learn more complex temporal relationships. The effects of the prediction problem difficulty are illustrated in Section 3.8.1).

To obtain the data for training the model, we extract all possible spatiotemporal patches of spatial radius l_s, temporal extent l_t and delay d. This means, that for a dataset of size (X,Y,Z,T) we have n data points:

$$n = (X - l_s) \times (Y - l_s) \times (Z - l_s) \times (T - (l_t + d)) \tag{3.3}$$

The size of the extracted data grows polynomially with the dataset resolution, and for large volumes it may easily reach terabytes, making the training computationally impractical. To alleviate the problem, we perform random undersampling of the data, including a given patch into the training data with probability p_u.

After the training, we evaluate the model on the whole dataset, i.e. on every possible patch, obtaining a full spatiotemporal prediction volume. The prediction volume is slightly smaller than the original dataset, since we cannot perform prediction near the borders. Finally, we compute the absolute difference between the prediction volume and the original data. The resulting misprediction volume is what we use as input for our explorative visualization (Section 3.5) and other applications.

3.2.3 Multiple Prediction Models

We can extract more information about the behavior irregularity by using not just one, but a set of models with varying capacity. Simpler models can only capture basic temporal relationships during training, producing accurate predictions only for the most common and simple behavior. More complex models are able to represent many scenarios and produce fewer large errors, failing only on the most irregular behavior. This allows to roughly categorize different data regions in terms of their irregularity.

When doing prediction with multiple models we follow the approach described previously in Section 3.2.2. Only the extracted training data is reused by different models, while weight initialization, training and prediction are performed separately. Once we have made a prediction using each of the models, we compute the absolute differences to the original data, obtaining a misprediction volume for each model. We describe how these volumes can be used for visualization in Section 3.5.

3.3 Causes of Prediction Error

Before using the models and their prediction errors to support visualization, it is important to first understand their properties, which we discuss in detail next. There are several potential causes for errors of a local prediction model. We distinguish between three scenarios that can lead to high prediction errors, exemplify them via dedicated synthetic datasets and demonstrate the corresponding results of our prediction-based approach:

- **Uncertainty (Section 3.3.1)** Behavior cannot be predicted completely from the input (due to stochastic processes or insufficient data).
- **Uniqueness (Section 3.3.2)** Behavior only rarely occurs (i.e., underrepresented in the training data).
- **Complexity (Section 3.3.3)** The model's capacity is insufficient to accurately fit numerous different behaviors.

3.3.1 Uncertainty

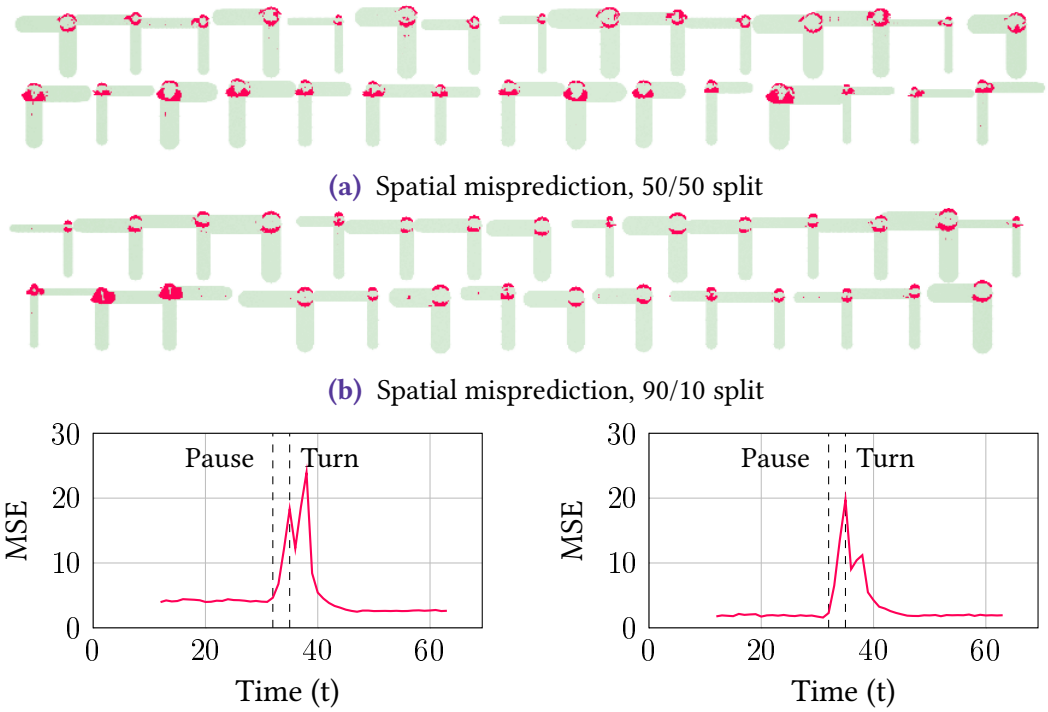
The uncertainty scenario arises when the target volume value is not completely determined by the model's input. This can happen due to hidden variables that are not included in the data, the patch size being too small to contain all the relevant information (in space or in time) or due to innate stochasticity of the data-generating process. Conceptually, even an optimal predictor would have non-zero error in this scenario, i.e., Bayes error rate (Kulkarni and Harman 2011).

To analyze this scenario, we constructed a dataset consisting of thirty small circular objects moving in space. The objects have randomized velocities and radii, but all move uniformly upward. After 32 timesteps, all the objects simultaneously stop moving, staying in place for three timesteps. Afterward, all of them continue the uniform motion but in different directions: half of the objects move to the left, and the other half moves to the right.

This dataset contains uncertainty, because it is impossible to predict whether an object is going to turn to the left or to the right using only local information. The pause in the objects' motion is introduced to signal the upcoming change of direction, in other words, the model "knows" when the turn is going to happen, but does not "know" whether it is a left or a right turn. By restricting the uncertainty to only the direction we are able to more clearly observe its effects.

We trained a small two-layer neural network using a prediction delay of three frames. Specifically, we use a "D64-D32" model throughout this section, with the model notation introduced in more detail later in Section 3.4. In Figure 3.2, we show the resulting spatial regions of high error and a line plot of the total error over time. The first important feature is the two peaks in the temporal view (Figure 3.2c). The former peak corresponds to the pause in the objects' motion, since it cannot be expected from the local information alone (a global model could "memorize" that it happens at a certain position). The latter peak occurs when the objects continue to move left or right, and the model cannot predict the direction. In the spatial view (Figure 3.2a) we show the traces of the objects (in pale green) and the errors (in red). We see that for many objects the model predicts some combination of the left and right motion, resulting in error blobs on both sides of the turn. This shows that the model predicts the turn, since predicting a lack of motion would result in even larger errors, but it cannot predict the exact direction due to its uncertainty. Once the model observes the first of the frames after the turn, the uncertainty is no longer there, and the model's prediction becomes accurate again (thus there are no large errors after timestep 39).

For another experiment, we generated a similar dataset, where only three out of thirty objects turn right, and the rest turn left. We then trained the model with the same configuration as previously. Figure 3.2d shows the temporal misprediction, where we can see that the second peak corresponding to the turn became significantly smaller. This can be explained by looking at the spatial misprediction in Figure 3.2b: the three objects that turned right have large errors, because the model has learned to predict the dominant left direction. The rest of the objects only have errors from the pause event (half-circles above and below, cf. Figure 3.2a), thus resulting in an overall smaller error peak. Although the uncertainty is still present, the right turn scenario occurs less often in the data, and thus the average loss of predicting the left turn is significantly lower than the other alternatives (predicting a right turn or some mixture of both). Thus, the model always predicts the left turn which is the optimal prediction in this uncertain



(a) Spatial misprediction, 50/50 split

(b) Spatial misprediction, 90/10 split

(c) Temporal misp., 50/50 split

(d) Temporal misp., 90/10 split

Figure 3.2 — Demonstration of the **uncertainty** scenario. The data has small circular objects with different radii moving upward with different, but uniform velocities. After 32 timesteps and a pause of three timesteps, they randomly move left or right. We show the traces of the 30 objects in pale green, with the width of the trace corresponding to the radius of the object. Since neither the pause nor the turn are deducible from the local data, the outcome is uncertain, leading to prediction errors (in red). **a**: When two turn directions are evenly split, the model predicts a mixture of both turns, incurring large errors for most objects. **b**: When most of the objects turn left, the model predicts the left turn, incurring larger error on the right-turning objects. **c, d**: Average error in each timestep, both the pause and the turn events are visible. The error at the turn decreases for the 90/10 case, since the outcome is less uncertain.

scenario (both in terms of the MSE loss and the Bayesian decision rule).

Note that we used the larger prediction delay of three frames to better illustrate the uncertainty scenario, but the results do not change qualitatively when using a prediction delay of one. The model still cannot anticipate the direction change and produces prediction errors. However, in this case they occur only during one frame and the error peak immediately follows the event.

(a) Nineteen objects use a period of seven frames, the right-most object has a period of four.

(b) The left-most object uses a period of seven frames, the rest have period of four frames.

Figure 3.3 — Demonstration of the **uniqueness** scenario. The two datasets consist of 20 objects with various shapes and sizes moving upward, changing their voxel values according to the same periodic function. We visualize the traces of objects (in pale green) and regions with high errors (in red). **a:** When one of the objects uses a different period than the rest, it is an outlier and causes high prediction errors. **b:** When all but one object use the former "outlier period", what used to be an inlier becomes an outlier instead.

3.3.2 Uniqueness

If a distinct pattern of local behavior is sufficiently unique, the models tend to produce high errors predicting it. Since the behavior is underrepresented in the training data, the model is likely to prioritize more typical patterns, especially for models of small capacity. There is also a connection to the generalization performance of the model and hence, regularization: accurately fitting an outlier may imply overfitting the training set, which becomes harder when using simpler models and/or stronger regularization.

To demonstrate the uniqueness scenario in practice, we constructed a simple dataset where twenty small objects of various shapes move uniformly upward with a constant velocity. The objects change their voxel values following a periodic function ("blinking"), with all but one object using a period of seven frames. This single object is meant to be an outlier and has a period of four frames.

The results obtained using the "D64-D32" model and a three-frame prediction delay are presented in Figure 3.3a. There we visualize the traces of the moving objects (in pale green) and regions where the model produces high errors (in red). As we can see, the first nineteen objects (left-to-right) did not incur large prediction errors, and the last outlier object is easily detected. In an inverse experiment, we used the seven-frame period for a single object, and the period of four frames for the following nineteen objects. The results for a model trained on this data are shown in Figure 3.3b. Here we see a symmetrically opposite outcome: the objects with the former "outlier period" of seven frames are now accurately predicted, while the four-frame object has incurred

large errors.

This behavior might appear to be similar to the uncertainty scenario, however there is an important distinction. With uncertainty, two different outcomes follow identical (or very similar) local behavior. Thus, it is impossible for the model to "disentangle" the two outcomes in the input space, regardless of its capacity. In contrast, in the uniqueness scenario, the rare outcome is still completely determined by the prior local behavior.

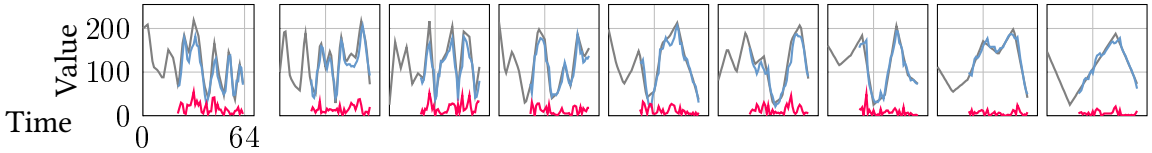
3.3.3 Complexity

The complexity scenario refers to the situation where the model's capacity is insufficient to accurately fit the data. Conceptually, as the number of distinct local behavior patterns increases, the learned mapping is expected to produce a correct prediction over an increasing number of distinct regions of the input space. When the capacity of the model is too low, it cannot separate the different regions, leading to prediction errors. Although determining the capacity of a neural network is still a difficult problem (Goodfellow et al. 2016), some understanding can be gained from statistical learning theory and the VC-dimension (Vapnik and Chervonenkis 1969), which measures a classifier's capacity as its ability to separate arbitrary points in the input space.

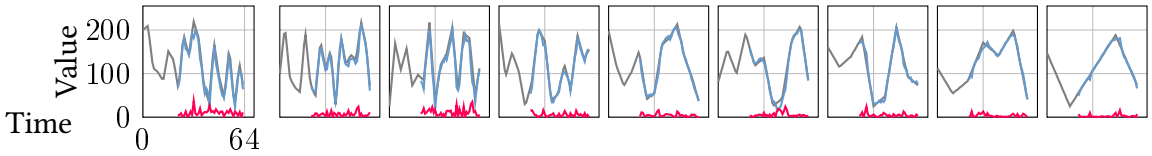
To illustrate the effects of complexity, we constructed a dataset consisting of 27 small objects of varying shapes and radii, which change their value following one of several predefined patterns. There are three possible shapes and three radii, resulting in nine unique object types. The first frame of the dataset is presented in Figure 3.4a, demonstrating the shapes of the objects. For each of the nine shape-radius combinations we defined one unique temporal pattern. The patterns are polylines with a different number of segments, resulting in time series of varied complexity. Note that since each series corresponds to a unique spatial shape, there is no uncertainty in the data, and the future values are completely determined by the past. We trained a "D16-D8" model with a prediction delay of three frames on this data, using a long 20-frame patch size to minimize uncertainty of prediction. For convenience, we plot the prediction over time only at the center point of each of the nine objects, resulting in one temporal plot for each unique shape (Figure 3.4b). Even though the model was trained on the data, its capacity is insufficient to capture the many possible local behavior patterns. Thus, the model incurred significant errors (in red), with higher errors corresponding to the more complicated fast-varying temporal sequences.

In a complementing experiment, we trained a larger "D64-D32" model under the same configuration, with the results plotted in Figure 3.4c. Although the data has not changed, the model was able to better predict the behavior, since its capacity is larger (overall MSE of 1.34 vs. 3.22 for the smaller model). Many more of the various patterns in the data are now captured well, thus marking a smaller subset of spatiotemporal events as irregular.

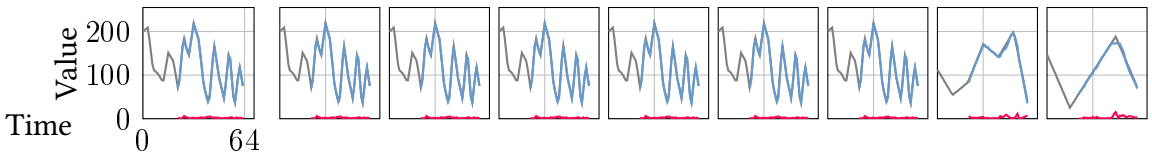
(a) The first frame of the dataset, showing 27 objects with 9 different spatial shapes.



(b) Results from the "D16-D8" model, showing the object value (gray), the model prediction (blue) and their difference (red) over time.



(c) Results from the "D64-D32" model, showing the object value (gray), the model prediction (blue) and their difference (red) over time.



(d) Results from the "D16-D8" model on the modified dataset, where the most complicated time series was used for most of the objects. Showing the object value (gray), the model prediction (blue) and their difference (red) over time.

Figure 3.4 — Demonstration of the **complexity** scenario. **a:** The dataset consists of stationary objects with nine unique radius-shape combinations. Each object type "blinks" following a unique time series. **b:** Value at the center of nine objects (one for each unique behavior), shown together with the prediction of a simpler model (in blue) and the resulting error (in red). The more complicated fast-changing time series (on the left) result in larger and more frequent errors. The larger model in **c** produces smaller errors with fewer spikes, thus highlighting fewer spatiotemporal patterns. **d:** Interaction between the complexity and the uniqueness scenarios. When the most complicated behavior was made frequent in the dataset, the model captured it more accurately than the simpler one.

The complexity scenario is distinct from uniqueness, because all the behavior patterns occur with the same frequency. However, in practice they often interact, with both complexity and frequency of local patterns dictating what is predicted accurately by the model. We demonstrate this complexity-uniqueness interaction in Figure 3.4d, where we modified the dataset, such that the most complicated time series is used

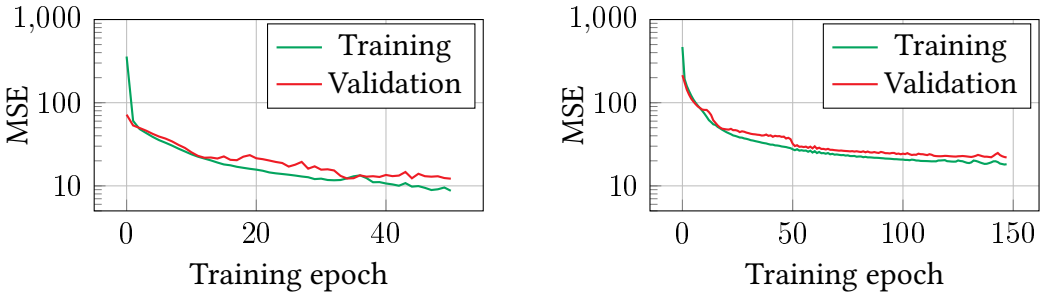


Figure 3.5 — Training history for the models "C64-C64-D256-D256" (left) and "D256" (right) on the droplet dataset. We perform early stopping to avoid overfitting, terminating the training when the loss on the validation dataset has stopped improving.

more often than the simpler ones. Specifically, we replaced seven out of nine temporal patterns with the most complicated series from the previous experiment (on the left), leaving the two simpler patterns on the right unchanged. First of all, the overall MSE of the "D16-D8" model has decreased from 3.22 to 0.23 (cf. Figure 3.4b), since there are fewer unique local patterns to capture. More importantly, the model has predicted the complicated behavior more accurately than the simpler one, because it is now common for the dataset. This illustrates a key advantage of ML models compared to manually defined features: they can still account for complex behavior if it is typical for a given dataset, without prior assumptions about the exact characteristics of this behavior.

3.4 Model Architecture, Training and Prediction

All of our prediction models are feed-forward neural networks with two types of layers: convolutional and fully-connected. This choice was made to keep them both simple and generic (see Section 3.2.1). All convolutional layers act on spatial dimensions using filters of size $3 \times 3 \times 3$. For 2D datasets, filters of size 3×3 are used. When referring to models, we use short names like "C64-D32" that encode their architecture. The convolutional layers are encoded as "CX", where X specifies the number of feature maps. We refer to fully-connected (dense) layers as "DX", where X is the number of neurons in the layer. Both layers types are also followed by a ReLU activation. When using a combination of convolutional and fully-connected layers, we insert a non-parametric flattening layer in-between, which reshapes the input 4D array into a flat array appropriate for fully-connected layers. Additionally, all models have a final fully-connected layer with a single neuron and no activation, which acts as a linear output unit for the regression task.

For instance, model "C64-C64-D256-D256" has the following layer sequence: "Convolution(64, 3^3), Activation(ReLU), Convolution(64, 3^3), Activation(ReLU), Flat(), FC(256),

Activation(ReLU), FC(256), Activation(ReLU), FC(1)". A special case is the model we refer to as "D1", which in fact consists only of a flattening layer and a single neuron with no activation function. Thus the model learns a linear function of the supplied inputs.

As described in Section 3.2.2, the model is trained on spatiotemporal patches sampled from the input volume dataset. We normalize the training data to mean zero and standard deviation of one, but otherwise do not perform any pre-processing. This is an important aspect of keeping the overall method domain-agnostic.

We train all our models using the "Adam" (Kingma and Ba 2014) variation of the stochastic gradient descent, with the learning rate of 0.001 and the batch size of 1024. Following standard practices, the training is performed in epochs, where in each epoch the model observes the whole training dataset once. We perform holdout validation, splitting off 20% of our data into a validation dataset to monitor the generalization performance. Furthermore, we use the validation data to perform early stopping as a form of regularization (Goodfellow et al. 2016), although due to the relatively small capacity of our models and large amounts of data we do not experience severe overfitting, as indicated by our similar training and validation losses (Figure 3.5). We monitor the validation loss and stop the training process when no improvement has been observed for 25 epochs and take the model checkpoint from the best epoch as our trained model.

To perform prediction on a dataset we need to evaluate the trained model on every spatiotemporal patch. Even for medium-sized volumes the amount of input data may easily reach terabytes. Because of this, we try to reduce the amount of processed data through undersampling (Section 3.2.2) and have also heavily optimized our implementation. We use Python for all the high-level operations, while IO operations, patch extraction, metric computation and other performance-critical components are implemented in C++. The first challenge is that the total size of all the patches, even with undersampling, is too large to fit into RAM, let alone VRAM. Therefore, our implementation operates out-of-core, loading batches of patches on the fly during both training and prediction. Furthermore, reading 4D patches from the volume data stored in C-order on disk incurs a high load of unaligned reads that significantly slow done the processing. We alleviate the issue by mediating the reads with a RAM read/write cache that is dynamically flushed based on the incoming read/write operations. Next, to further speed up the extraction of patches (which also includes checking them for empty space), it is performed in multiple threads synchronized through atomic operations. Once the patches are extracted, they are uploaded to the GPU, processed and in case of prediction, the results are written to disk, again, through a RAM cache that batches the IO operations. An additional optimization technique that we use is caching of the prediction result for spatiotemporal patches that contain nothing else but empty space (in the input and in the target). We compute the model's prediction for an empty patch once, and then re-use this result whenever an empty patch is detected during the extraction. As a

result, we obtain identical prediction results, but can reduce the GPU execution time for some of the datasets.

3.5 Visualization of Prediction Error

Using our prediction-based approach (Section 3.2.1) we aim to capture the properties of the dataset in a domain-agnostic fashion and provide an overview of regions with irregular local behavior. For this, misprediction volumes obtained from multiple local models can be used to construct spatial and temporal views of irregularities in the data. Due to the stochastic nature of the model training process, the raw misprediction volumes are noisy, with the exact error magnitude varying among neighboring voxels. To suppress this noise and present a visually clear overview of each model's large-error regions, we apply spatial smoothing to each volume. This makes sure that large spatially-coherent prediction errors have a stronger effect on the results than sparse random deviations. For consistency, we use a kernel radius of five for all results presented below. After the smoothing, we aggregate each misprediction volume separately using the maximum function. This way we avoid summing up smaller prediction errors, allowing us to distinguish between spatial regions where brief unpredictable behavior took place and regions where low errors consistently occurred throughout the dataset. Thus, we end up with multiple spatial volumes, each voxel of which represents the largest smoothed prediction error that a particular model has at the given spatial location.

Next, we assign a transfer function with a single distinctive color to each of the aggregated volumes, and perform multi-volume raycasting. We aggregate the samples along the ray front-to-back, compositing a sample from each of the volumes at each spatial location. This allows the user to distinguish where prediction errors occurred for each of the models. When compositing samples, we order the volumes according to their model's capacity, highest first. Although the transfer functions can be adjusted interactively, we often use single-peak transfer functions for iso-surface-like rendering of all but the first volume, which corresponds to the model with highest capacity. This way the user can always see the regions where even the most sophisticated model has failed, i.e. regions with the most unpredictable behavior. For context, we also render the original spatiotemporal volume simply averaged over time. Here we typically use a relatively transparent ramp transfer function to keep the focus of the visualization on the prediction error volumes. We used this approach already in Section 3.3 (e.g. Figure 3.2a) and demonstrate the results in more detail in Section 3.5.1.

In addition to a spatial misprediction view of the dataset, we provide a temporal view. Its goal is to highlight the timesteps when the most unpredictable changes occurred and help to detect different temporal phases of the data. To this end, we aggregate the misprediction volumes separately for each model, for each timestep, eliminating

the spatial dimensions. The aggregation is done using the average function, which we found to be more appropriate for distinguishing temporal phases. As a result, we obtain a time series for each model, which we plot as a line graph. A previous example can be seen in Figure 3.2d, while a detailed discussion of the results follows in Section 3.5.1.

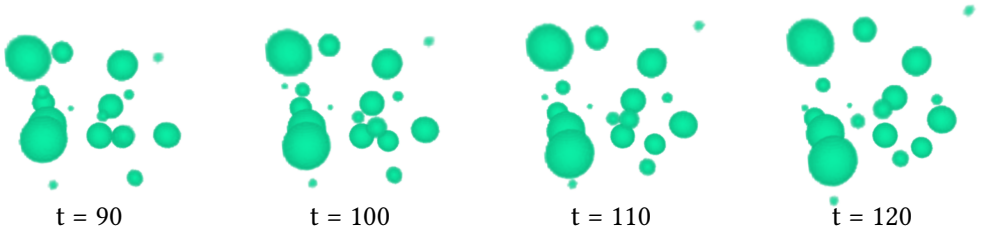
The spatial and temporal views can be linked together via interaction for dataset exploration. Specifically, the user can interact with the temporal misprediction view by selecting a time range. When the range is selected, the spatial view is recomputed using only the timesteps that are part of the specified time range. This can be used to focus on periods of time with large prediction errors, providing a spatial view of the regions that couldn't be predicted well by the model. If only one timestep is selected, we show a single timestep of the original data augmented by regions of high prediction error without any aggregation. In this mode the user can access the lowest level of detail and search for explanation of the patterns occurring in the aggregated views.

3.5.1 Results

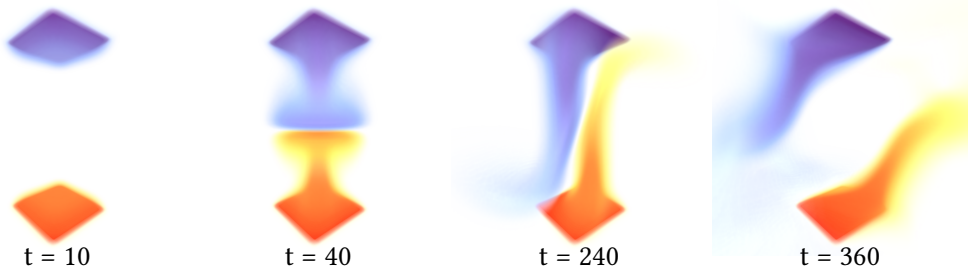
Now that we discussed the method of visualizing the prediction errors, we present the corresponding results. The results are structured according to the datasets, which we show in Figure 3.6. We use the following three models throughout this section: "C64-C64-D256-D256", "D256" and "D1". For brevity and uniformity, we refer to them as "Model A", "Model B" and "Model C" respectively. We also introduce a short patch size notation: "$l_t \times l_s^k$", where l_t and l_s are temporal and spatial extents (Section 3.2.2) and k is the number of spatial dimensions, e.g., 7x5^3 is a patch of size 7x5x5x5.

Synthetic dataset (Figure 3.7). We begin with a synthetic dataset for which we can provide the ground truth describing all the events that are happening in the data. This allows us to validate that the model produces error around irregular behavior and not randomly. Furthermore, using a controlled dataset we can more cleanly demonstrate some of the common properties of the prediction errors, which will be helpful before moving on to real-world datasets. The synthetic dataset contains smooth spheres traveling along straight trajectories. Spheres fully-elastically react to collisions with each other and with the boundaries of the dataset (Figure 3.6). Some of the spheres shrink or grow as they move, and may abruptly change their trajectory. Overall, the data contains simple behavior (linear motion, size change) as well as more interesting and hard-to-predict events (collisions and trajectory changes).

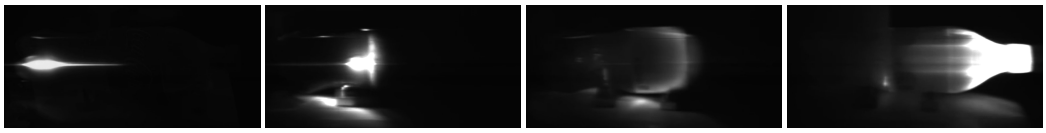
We applied models *A*, *B* and *C* to the data and present the resulting spatial misprediction view in Figure 3.7a. What we can see is that the model with the highest capacity (orange) produces large errors only in smaller regions. These regions correspond to sphere collisions (appearing as two adjacent objects) and significant trajectory changes. The model was able to adequately capture the behavior associated with motion, and thus isolated the collisions and sharp trajectory changes. A simpler model (*B*, in magenta)



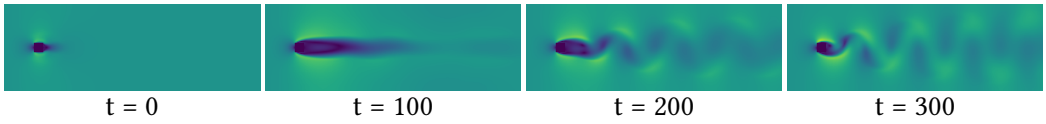
spheres: synthetic data containing moving spheres with collisions.



hotroom: air temperature simulation.



bottle: images of a laser pulse propagating through a bottle (Velten et al. 2013).

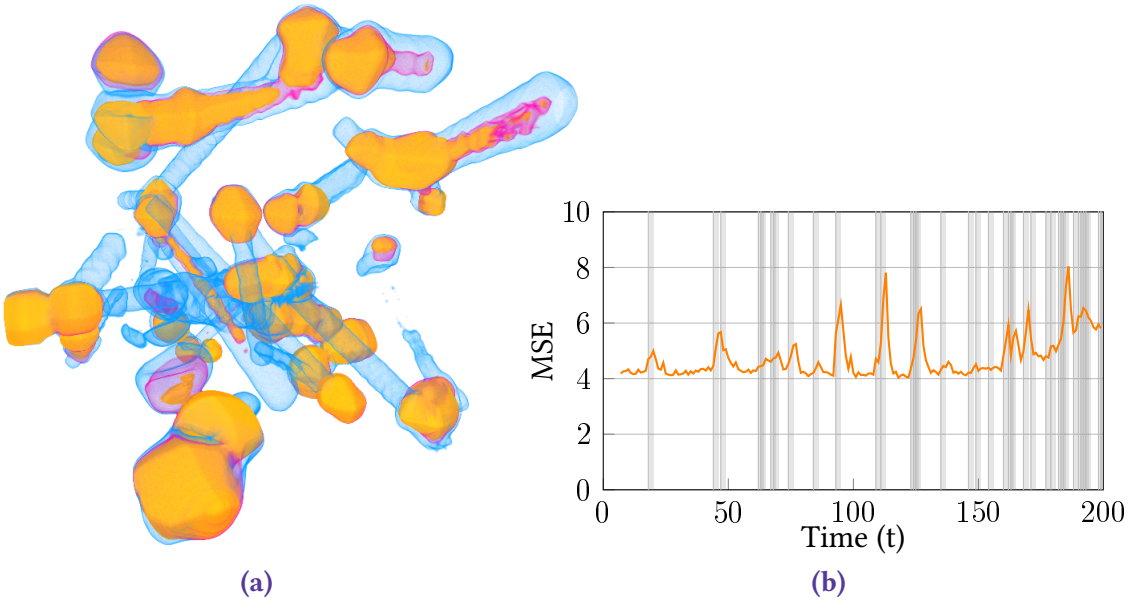


vortex street: CFD simulation of flow around a cylinder.

Figure 3.6 — The datasets used to evaluate the visualization of prediction errors.

failed in some additional regions, corresponding to trajectories of smaller fast-moving objects. Finally, the simplest model (C, in blue) displays significant prediction errors for many faster-moving objects, resulting in long tube-shaped regions representing the linear trajectories of the spheres.

An interesting property to observe is that the high prediction error regions are often subsets of each other: the simple model fails where complex model fails, but also in some additional locations. This is the expected behavior, considering the fact that all the model architectures are (1) qualitatively similar (in terms of their components), differing mostly quantitatively, and (2) trained on the same data with the same loss function. Data points that produce large gradients during training result in highest "priority" for all models. The simpler models do not have a high capacity for capturing



(a) (b)

Figure 3.7 — a: Spatial misprediction on the synthetic dataset with moving spheres. The most complex model (orange) detects sphere collisions, while simpler models (magenta, blue) also highlight some of the faster and irregular motion. **b:** Comparison of detected temporal events to the ground truth. **Orange line:** Mean prediction error of the most complex model (A) in each timestep. **Vertical lines:** Timesteps when events happen. Light-gray stripes show the expected error delay (two frames). Whenever an event occurs, we observe a corresponding spike in the prediction error.

different scenarios, so after fitting the most "important" data points, they cannot fit additional scenarios without worsening their overall performance.

In Figure 3.7b we present the temporal misprediction view for our synthetic dataset obtained using the model A. To compare our results to the ground truth, we plot the model's error (orange) and mark the timesteps where an event has occurred with a gray vertical line. The figure shows that when no events happen, the model maintains a steady prediction error. However, when an event takes place, we can see a corresponding spike in the graph. Another interesting observation is that the peak error occurs several frames later than the event itself. This is due to the prediction delay (Section 3.2.2), which in this case was three frames ($d = 3$). To better illustrate this effect we have also plotted the expected error delay with a wider stripe. In the frame when a collision happens, the objects change their movement direction, but still have not traveled too far apart from their original trajectory. In the next $d - 1$ frames, the model still has not observed the collision (because of the prediction delay), so it continues to predict movement along the original trajectory, while the object continues to move away, increasing the error. Once the model has seen the collision, it "corrects" the prediction,

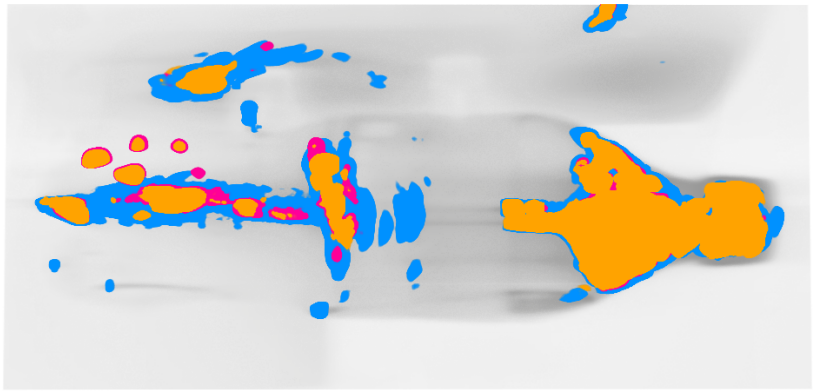


Figure 3.8 — Spatial misprediction view of the "bottle" dataset, with original data for context in gray. The complex model (in orange) highlights only the transition regions, while the others also show the initial pulse trajectory (on the left). The diffuse phase is completely filtered out (in the middle).

and the error drops to its normal level. The drop is also not immediate, since the model initially sees only one frame of the new trajectory, and typically needs several timesteps to accurately extrapolate the object's future position.

Bottle (Figure 3.8). The data contains femto-photography images of a laser pulse propagating through a plastic bottle (Velten et al. 2013), shown in Figure 3.6. We used models A (in orange), B (in magenta) and C (in blue) to perform the prediction. As a result we obtain a visualization highlighting regions of transitional behavior (Figure 3.8). Roughly speaking, the pulse moves from left to right, exhibiting different phases: appearance of the initial pulse, transition into the diffuse phase behind the bottle label, appearance of inter-reflections near the neck of the bottle. Importantly, the complex model (in orange) highlights only the transitional regions. Both the propagation of the diffused pulse (in the center), and the propagation of the inital pulse (on the left) are not exhibiting irregular behavior and are captured by the model. The simpler models have a harder time predicting it, resulting in a difference between the misprediction regions. Similarly, the areas in the upper part of the dataset correspond to the appearance of the reflection of the pulse (detected by all models) and its propagation (shown by the simplest model in blue).

Hotroom (Figure 3.9) We use the "hotroom" dataset (Figure 3.6) to show our results on spatially and temporally smooth data. It depicts a simulation of air temperature in a room with a hot and a cold plate on the bottom and the top, respectively. This produces a dense smoothly-changing temperature field (we cut off neutral temperatures in the

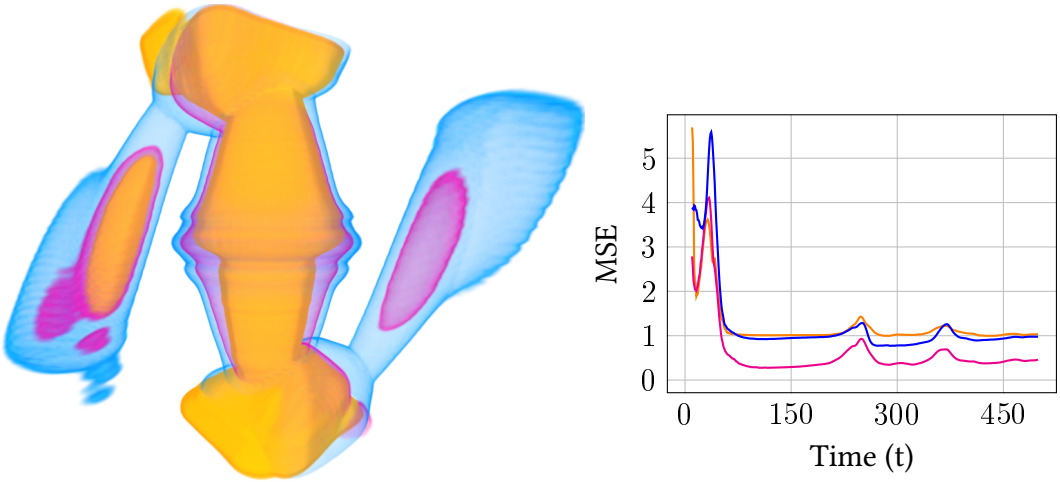
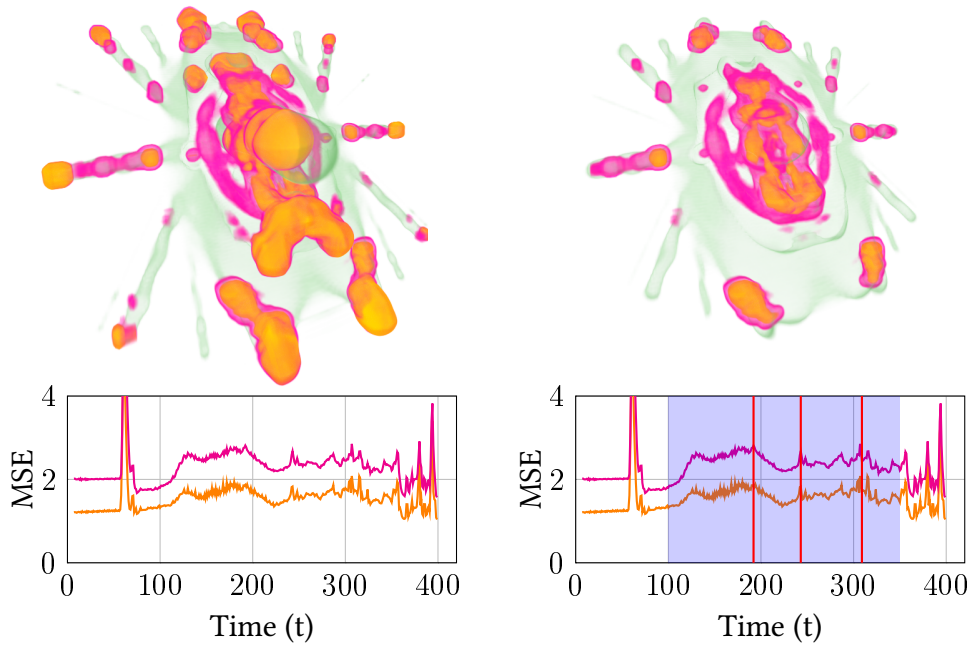


Figure 3.9 — Spatial and temporal misprediction views of the "hotroom" dataset. Three significant events can be identified in otherwise smooth data: the initial front collision ($t = 21$), collision with the plates ($t = 231$) and the formation of new streams ($t = 350$).

transfer function to highlight the cold and the hot air streams in Figure 3.6). We use models A, B and C to construct spatial and temporal views presented in Figure 3.9. The spatial misprediction regions correspond to three major events that are also apparent in the temporal view: (1) the initial collision of cold and hot streams (pillar-like structure in the middle), (2) the collision of the streams with the plates of the opposite temperature (asymmetric regions top and bottom), and (3) the formation of two new air streams, after the initial fronts have disappeared (side regions).

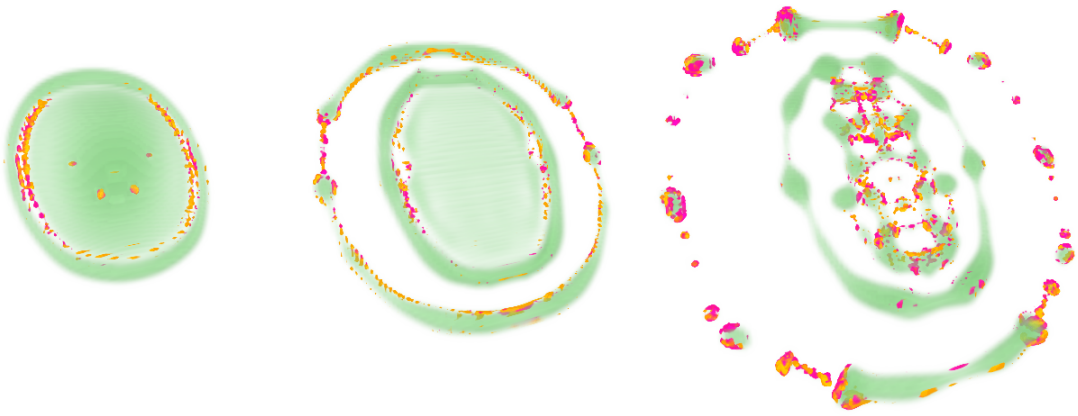
Droplet (Figure 3.10). To demonstrate the interaction between the temporal and the spatial misprediction views, we have applied our method to the droplet dataset. The dataset comes from a two-phase flow simulation of two colliding droplets. After the initial collision, the droplets form an expanding disk of fluid that consequently splits into many secondary droplets. Drop collisions are highly relevant in many technical applications, e.g., fuel injection and fire suppression. The data contains both temporal and spatial events, providing an interesting test case for our approach. For our analysis, we consulted with the domain scientists from the field of aerospace thermodynamics who conducted the simulation.

In Figure 3.10a we present the temporal misprediction view of the data. Although a lot of smaller events happen in the data, we can distinguish the initial two-droplet phase with steady error (frames 0-70), disk expansion phase with increasing error (frames 70-130), and a long tail of smaller spikes with an overall decrease in error, as many droplets separate (causing spikes) and continue to travel predictably (reducing the overall error). Of special interest to us were the two biggest spikes at frames 62 and 394. The model's prediction contains massive errors that couldn't be explained by the



(a) Initial spatial (top) and temporal (bottom) misprediction view.

(b) Spatial (top) and temporal views (bottom) for the subrange.



(c) t = 190

(d) t = 243

(e) t = 308

Figure 3.10 — Interactive analysis of the "droplet" dataset. **a**: The models (orange and magenta in spatial view) highlight the initial droplets, the expanding ring, a few larger droplet separations and several oscillating droplets. Two largest spikes in the temporal view correspond to simulation timestep changes. **b**: After selecting a subrange, we no longer see the inital droplet collision and can better focus on the locations of droplet separation. **c, d, e**: Single-timestep views showing exact locations of the prediction error (see vertical lines in **b**).

data. After consulting with the domain scientist we discovered that the simulation has irregular timestep sizes, and the two spikes in model's error align with two abrupt changes of the simulation timestep, effectively highlighting an irregular temporal event.

In Figure 3.10a we also show the spatial misprediction view of the data, obtained using the model A (in orange) and the model B (in magenta). We also aggregate and render the original data in faint green for context. To filter out the initial droplet phase and the discovered simulation timestep changes, we select a temporal subset of our misprediction data (Figure 3.10b). Inspecting the corresponding spatial view, we see several locations of irregular behavior: the central regions, where many secondary droplets have separated, the large ring corresponding to the ring separation, and four smaller regions in the corners, where several large droplets have formed and split off. We also see abrupt traces of several large droplets flying off to the left and to the right, caused by their irregular oscillating motion. To investigate further, we focus on a few smaller temporal events, inspecting three individual timesteps (red lines in Figure 3.10b). The resulting spatial views (Figure 3.10c, 3.10d, 3.10e) show where the high errors occurred. In Figure 3.10c and 3.10d we see the source of the ring observed in the aggregated view: this is the location where rings of fluid have separated from the initial expanding disk. Figure 3.10e highlights the secondary droplets that split off in the center, as well as the four large droplets starting to form and separate, producing the corner regions in the aggregated view. Interestingly, we found an unexpected feature in the data: there are four small regions of error in the center of Figure 3.10c, which we could not explain. After an investigation together with the domain scientist it turned out, that the fluid disk contains small bubbles of air caught during the collision. Its existence was previously unknown and undetected using domain scientists' current tools: volume and vector visualization in ParaView and histograms of droplet properties extracted via scripts (mass, surface, etc.).

The discovery of the irregular timesteps (believed to be an issue with the pressure solver), as well as of the bubbles (known neither to us nor to the domain scientist) informally demonstrates the potential of our domain-agnostic approach to detect unexpected data features. Overall, navigation of the misprediction volumes allows us to get a quick impression of the most complex temporal and spatial regions, and then to use this information to "drill down" into the data, studying individual spatiotemporal events.

3.6 Automatic Timestep Selection

Spatiotemporal data often consists of many timesteps, all of which cannot be statically presented to the user, motivating temporal subsampling (e.g., Lu and Shen 2008; Tong et al. 2012; Frey and Ertl 2017a). However, a meaningful selection of timesteps typically requires data-specific selection criteria, since relying solely on the quantification of

differences can often lead to interesting process transitions being missed. Here, we aim to use the prediction error as a generic method to select the timesteps of interest. Our goal is to automatically present the significant irregular temporal events to the user, which provides an additional exploration tool, complementary to the misprediction views described in Section 3.5.

Our technique uses the misprediction volumes obtained from local models (Section 3.2) to detect timesteps that contain irregular temporal events. As usual, we begin by training a local prediction model and measuring the prediction error on the original data. As described in Section 3.5, we average the misprediction volume spatially, constructing a time series for the model. The resulting series describe the mean prediction error that occurred at a certain timestep (e.g., Figure 3.2d).

For timestep selection, we now consider the maxima of the time series, since they mark irregular temporal events and transition points in the data. To remove smaller deviations and help focus on the most significant events, we perform bilateral filtering of the time series, using a Gaussian kernel for both the time and value dimensions. The overall effect of bilateral filtering is smoothing over time that avoids smoothing over elements with large differences in value, thus preserving the significant error peaks that we are interested in. Then, we extract the local maxima points of the series and remove consecutive points that are closer than a predefined threshold. This helps to avoid choosing peaks that are only a few timesteps away, which can sometimes happen for events that occur over several timesteps and cause some lingering fluctuation of the prediction error. Generally, we set the threshold to only a few frames, to avoid missing separate consecutive events. After the maxima are detected, we pick the corresponding timesteps as the ones containing significant temporal events.

We exemplify the results of our timestep selection approach on the droplet dataset. We used the "C64-C64-D256-D256" model to construct the misprediction volume. The respective temporal misprediction graph is plotted in Figure 3.11. The time series was smoothed using bilateral filtering, using a Gaussian kernel with STD of 5.0 for time and 1.0 for value. The selected timesteps (corresponding to the error maxima) are marked with vertical lines, and their renderings are presented in Figure 3.12). The first few selected timesteps represent several important events: droplet collision, first ring separation, second ring separation, and the break-up of the ring into multiple droplets. The later timesteps correspond to several smaller separations and oscillating droplets. We can see that our method chose the timesteps near the key large events and more densely sampled the temporal phase that has many smaller interactions. And importantly, this happened without the need to define domain-specific selection criteria or features. This technique could be used in conjunction with the misprediction views to provide an overview of irregularities in the data, especially when manual navigation is not desired or not possible.

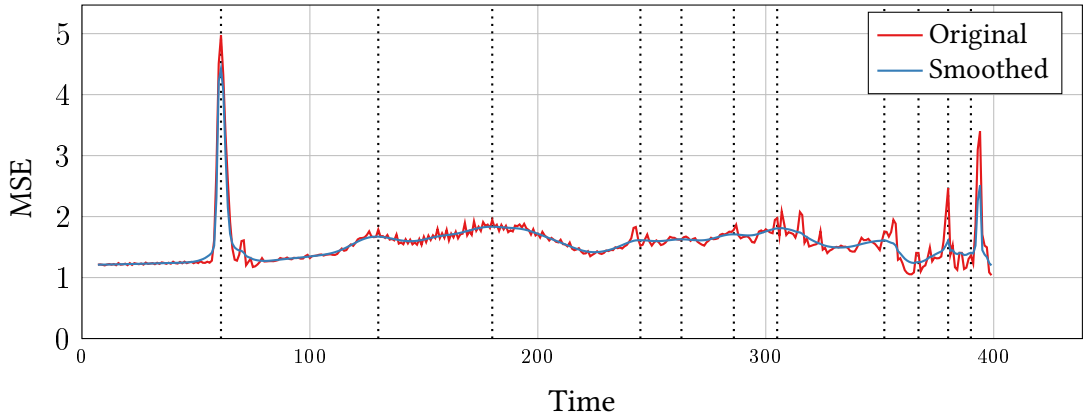


Figure 3.11 — Smoothed prediction error on the "droplet" dataset. By finding maxima of the smoothed prediction error we are able to find key irregular events in the data.

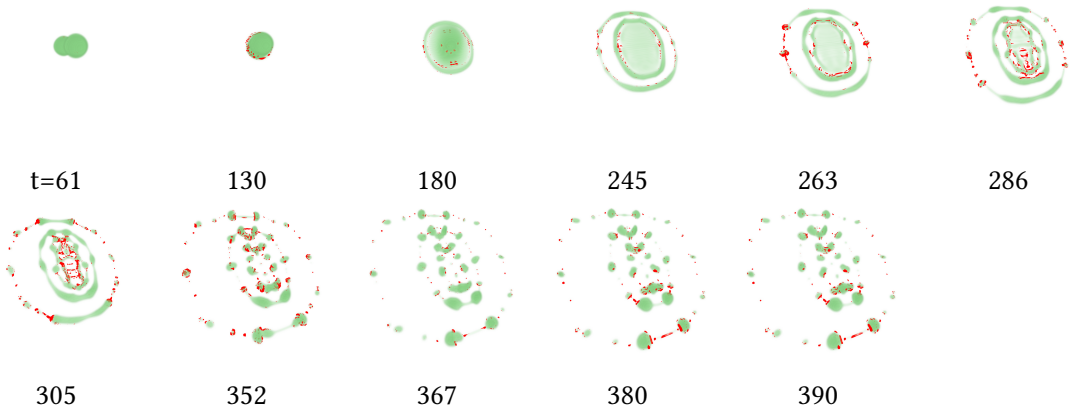


Figure 3.12 — Timesteps of the "droplet" dataset that were automatically selected by finding maxima of smoothed model error (shown in Figure 3.11). High error locations are rendered in red. Several sparse important events are captured in the beginning: the initial collision, two disk separations, outer disk break-up. The rest of the timesteps correspond to a more chaotic mass separation phase of the data.

3.7 Multi-field Volumes

So far we have concerned ourselves with the analysis of scalar volumes. However, data coming from scientific simulations is typically multifield, containing velocity vectors, pressure, temperature and other relevant properties that may be essential for the prediction task. Therefore, we extend our models to receive additional fields as input and then investigate how this affects the prediction, and thereby the detected spatiotemporal regions.

Dataset	Prediction delay	Model	MSE
cylinder multi-field	1	C64-C64-D256-D256	5.21
cylinder scalar	1	C64-C64-D256-D256	15.53
cylinder multi-field	1	D64	10.93
cylinder scalar	1	D64	14.03
cylinder multi-field	1	D1	31.54
cylinder scalar	1	D1	31.83
cylinder multi-field	6	C64-C64-D256-D256	24.64
cylinder scalar	6	C64-C64-D256-D256	45.42
cylinder multi-field	6	D64	43.97
cylinder scalar	6	D64	53.59
cylinder multi-field	6	D1	167.03
cylinder scalar	6	D1	176.27

Table 3.1 — Comparison of the total model error using scalar and multivariate prediction. Multivariate input data leads to fewer errors, especially for larger models and far-ahead prediction, since they have more capacity to leverage the additional information.

We performed a set of experiments on one of the turbulent CFD ensemble members. The model was supplied with velocity vectors, velocity magnitude and pressure fields, while predicting the velocity magnitude. We used three models of different capacity ("C64-C64-D256-D256", "D64-D32" and "D1"), two prediction delays (one and six frames) and compared to the prediction made using only the scalar velocity magnitude field as input (Table 3.1). As we can see, providing additional fields to the model results in lower overall prediction error. The differences are more significant when we increase the delay: predicting further into the future is easier with the additional information. We also notice that the larger models benefit more from the additional fields, since they can represent a more complex relationship between the input fields.

To investigate the exact spatiotemporal differences between the multifield and scalar models we performed an additional experiment on the droplet dataset. We compared a multifield model that received volume-of-fluid, pressure and velocity vector fields, to a scalar model that received only the volume-of-fluid values. The multifield model achieved an overall MSE of 0.14 and the scalar model an MSE of 0.23 (values are relatively low due to large amounts of empty space in the dataset). Thus, similarly to the previous experiments, providing more data allowed for an overall more accurate prediction. In Figure 3.13a we present the temporal misprediction of both models, where we see that they highlight the same events, with the only significant difference occurring around the initial droplet collision (frames 50-100). The multivariate model has a lower overall error, though the scalar model "recovers" slightly faster due to focusing on the motion of fluid and being unaffected by changes within other fields.

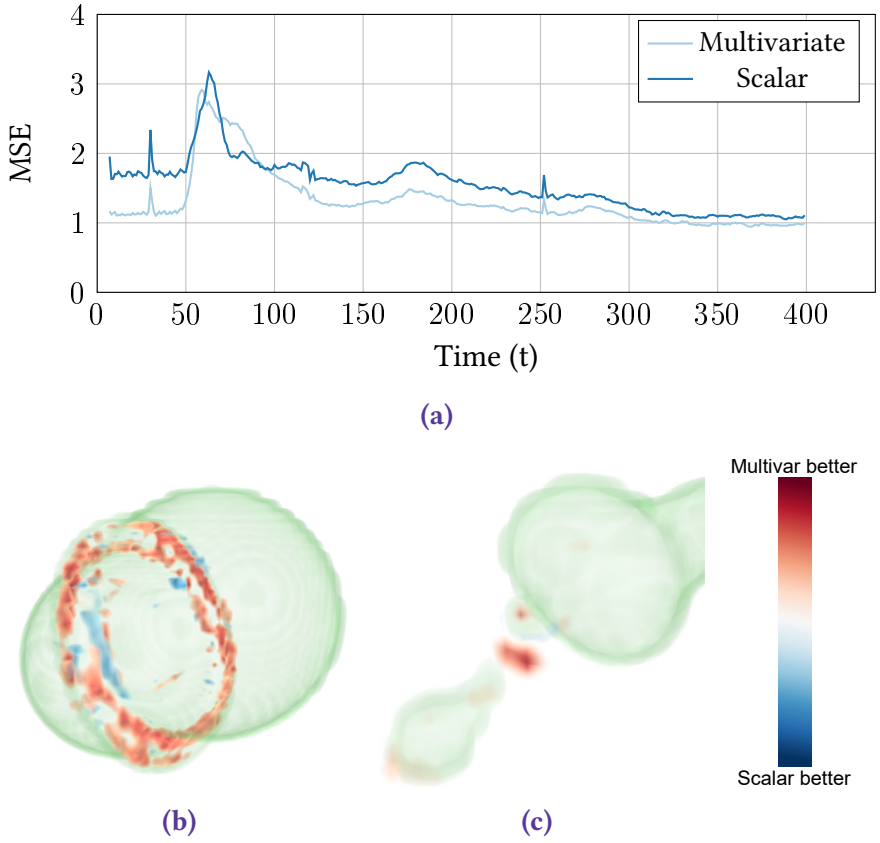


Figure 3.13 — Comparison of scalar and multivariate prediction on "droplet". **a:** Temporal misprediction views: scalar prediction is less accurate overall, but detects similar temporal events. **b, c:** Spatial difference between models' error regions, with **red** showing where the scalar model had larger errors, and **blue** – the opposite. The multivariate model is able to better predict the disk expansion (**b**) and small droplet separation (**c**) events.

In Figure 3.13b we show the difference between the models' absolute errors at frame 62 around the initial collision. We see that the scalar model produces larger errors around the expanding disk between the two droplets, while the multivariate model is able to use the high pressure values that accompany this expansion to distinguish and more accurately predict this behavior. Then, we inspected the data for other regions of significant deviation and found several smaller regions near ligament separations, where the multivariate model produces a more accurate prediction (Figure 3.13c).

Overall, we conclude that although multifield and scalar models are generally similar, providing and withholding fields from the model can in some cases affect the results qualitatively. Both approaches can be used depending on the application scenario, while

combining them can act as an additional analysis tool.

3.8 Assisted Parameter Selection

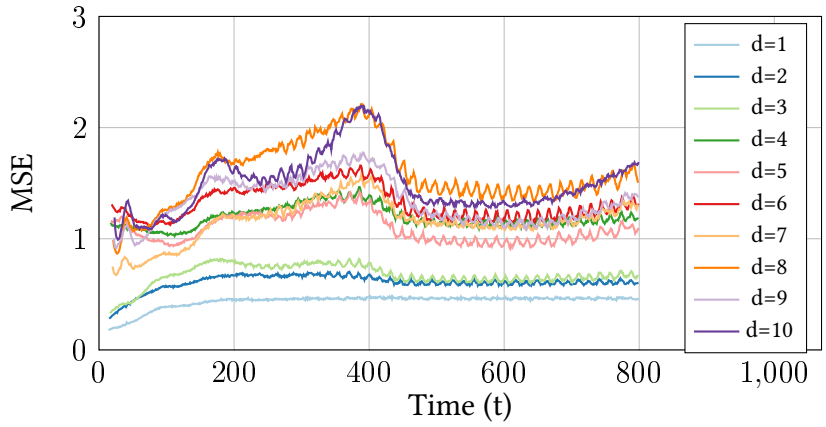
The main motivation behind our irregularity detection approach is to develop a method that is domain-agnostic. We achieve this by refraining from any domain-specific assumptions, but another way in which a method can become specialized is by relying on a large number of complex parameters. Although the parameters can be tuned to apply such a method to any domain, their adjustment might require significant expertise, either in machine learning or in the target domain. This would defeat one of the main purposes of having a domain-agnostic method in the first place, and should thus be avoided.

In this section, we discuss and investigate our most important parameters that directly influence the difficulty of the prediction problem (Section 3.2.2) and thereby, the detected irregular behavior. We demonstrate that the parameters have a predictable impact on the results, and furthermore, show how they can be selected in a semi-automated fashion.

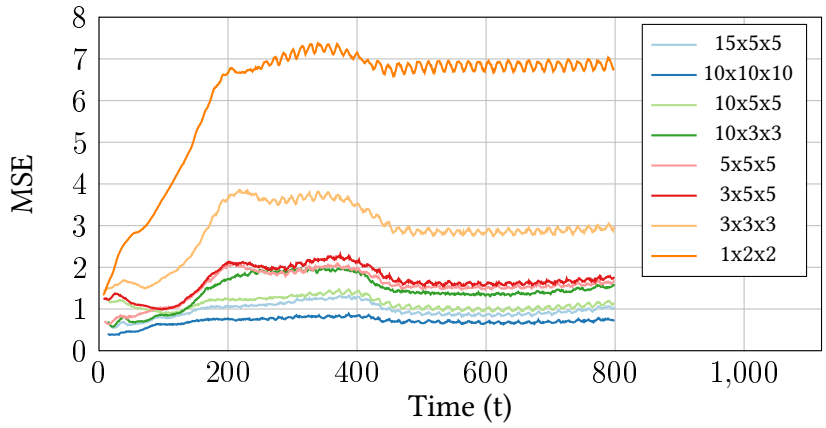
3.8.1 Parameter Study

Prediction delay (Section 3.2.2) is the distance in time between the last timestep provided to the model and the predicted timestep. To demonstrate its effects, we perform prediction with ten "C64-C64-D256-D256" models, each using a different delay on the "vortex street" dataset (Figure 3.6), which has a very fine temporal resolution. The temporal misprediction view for each delay setting is presented in Figure 3.14a. As we can see in the cases of one- and two-frame delays, low prediction delay has a very clear symptom: model prediction error exhibits almost no variation. Since the prediction task is too easy, the model is able to predict the whole dataset equally well. When we increase the delay, we can observe an overall increase in the error, as well as more pronounced differences between temporal phases. Some parts of the data remain simple to predict, but the more irregular regions start to produce errors that we can use in our analysis. Crucially, despite the quantitative variation between the different series, all models trained with the prediction delay of more than two frames highlight similar temporal events: growth of the trail and of the error until timestep 180, transition to more turbulent behavior until timestep 400, and eventually the onset of fully-periodic behavior indicated by the decreasing model error.

Patch size is an important parameter for our local prediction approach, since it defines what is local, i.e. what information is available to the model. For consistency, we use the "vortex street" dataset and perform prediction with eight "C64-C64-D256-D256" models



(a) Prediction delay



(b) Patch size

Figure 3.14 — A parameter study on the "vortex street" dataset showing temporal misprediction plots. **a:** Models with a delay below three frames do not detect any events because it is easy to predict a few frames ahead. **b:** Most of the models successfully highlight the complexity of the setup phase (frames 100-400). However, the models with very small (orange) or very large (blue) patch sizes do not clearly distinguish it.

using different patch sizes. The results are presented in Figure 3.14b. An important property for this fairly regular dataset is whether we can distinguish the different temporal phases from the model's misprediction plots. The "contrast" between the setup and the periodic phases varies with different patch sizes, and there are two extreme cases. First, a very restricted patch size of 1x2x2 produces a constantly growing error graph, showing that even high-capacity models cannot produce an accurate prediction given insufficient information. Second, the very large 10x10x10 patch produces low error compared to other models, but the misprediction graph of this model alone shows much smaller differences between the temporal phases. A large model with a very wide

view can predict well even the complex setup phase. But overall, the rest of the models produce reasonable and comparable results.

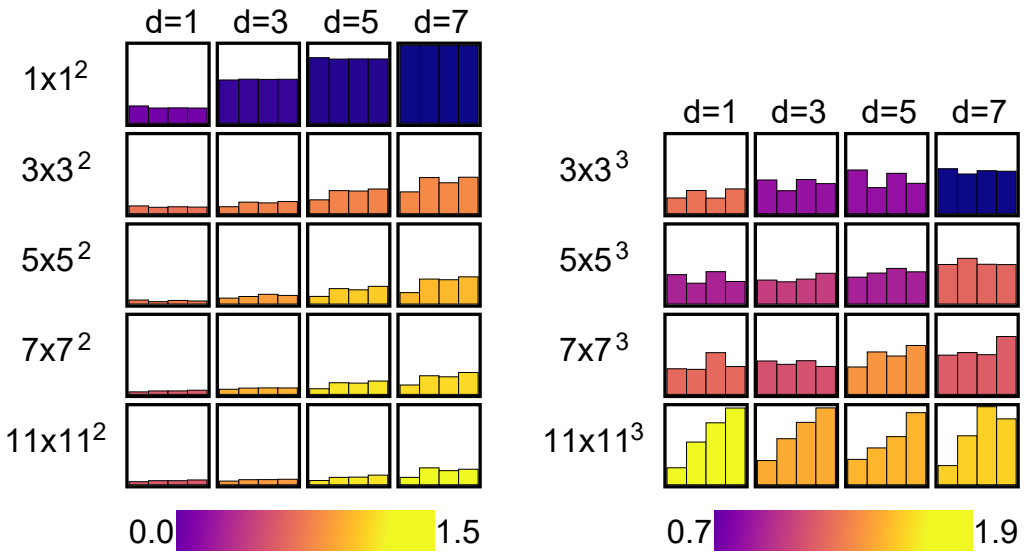
3.8.2 Parameter Selection

The patch size and the prediction delay parameters control the difficulty of the prediction problem: large delay (further ahead prediction) and small patch size (less input) make the prediction more difficult, while small delay and large patch size make it easier. Since our technique differentiates spatiotemporal regions based on prediction errors, it functions best between the extremes of impossible and trivial prediction, when some regions are hard to predict and some are not. A key observation is that as we approach these extremes, models of different capacity tend to produce similar errors. When the prediction is too difficult, all the models display similarly large errors, because additional model capacity does not help when information is missing. And when the prediction is too easy, all the models display similarly low errors, since even the simplest models can predict slow smooth processes. Thus, we can use this idea to help select appropriate parameter values by comparing the models' prediction errors and maximizing their diversity, i.e., favoring configurations that lead to larger differences between simple and complex models.

Specifically, we run our method using multiple models and compute misprediction volumes as described in Section 3.2.2. Then, for each spatiotemporal position we compute the diversity as the difference between the smallest and largest errors for this point among the different models. Then, we average this value over the whole dataset and divide it by the average prediction error of all the models, effectively computing the relative average prediction error range as our selection criterion. The normalization allows for better comparison across different parameter configurations and datasets. Overall, the larger error ranges indicate better parameter settings.

Using this information we construct a parameter space visualization. We render a grid, where each row corresponds to a patch size value and a column to a prediction delay value, and show a bar chart in each cell. The bar chart displays the average error of each model (ordered complex to simple, left-to-right), allowing the user to see a summary of models' performance. Additionally, we color each bar chart (grid cell) according to our error range criterion, showing how appropriate each configuration setting is in terms of local error diversity.

In Figure 3.15a we show results on the vortex street dataset, using models "C64-C64-D256-D256", "D256", "D64-D32" and "D32-D16" (left to right). Here we also include the trivial patch size setting of $1x1^2$. First, as to be expected when considering prediction difficulty, we see that larger patch size and lower prediction delays lead to lower overall errors. Importantly, when the patch size is too small ($1x1^2$) or the delay is too low ($d = 1$) the models show very little diversity, either failing or succeeding together. But when



(a) "vortex street" **(b)** "hotroom"

Figure 3.15 — Parameter space visualization for two datasets. We show a grid of bar charts for different values of patch size (rows) and prediction delay (columns). Each bar chart encodes the mean errors of the four models (sorted complex to simple, left to right). Additionally we compute an error diversity criterion which we use to color each cell. Higher delays with larger patch sizes lead to higher error diversity, which indicates a prediction problem of appropriate difficulty.

the patch size is large enough, larger delays lead to more diversity, since more complex models can still predict well enough, while the simpler models fail. Here specifically we find that patch sizes above $5x5^2$ and delays above 5 lead to best results, which is also reflected in the diversity score (encoded as color).

Next, we performed experiments on the hotroom dataset, using the same four models as before. Since 3D datasets are much larger and generate a lot of input data, we introduced stronger patch undersampling ($p_u = 10^{-4}$), which leads to somewhat noisier results, but still allows trends to be studied. Thus, we can perform a parameter study faster, and if needed later, run the core method with more data using the chosen parameter values. We demonstrate the results in Figure 3.15b. Overall, we see a similar pattern: small patch sizes lead to uniformly large errors, while larger delays improve error diversity, especially for larger patch sizes. In this case, all the settings with $11x11^3$ patch size lead to larger error ranges, this is explained by fact that very simple non-convolutional models tend to perform worse on very large 3D patches. For the hotroom, we find patch sizes of at least $7x7^3$ and the delay of 5 as the most appropriate. In general, we recommend picking the largest patch size that is computationally feasible and the delay value that maximizes model error diversity.

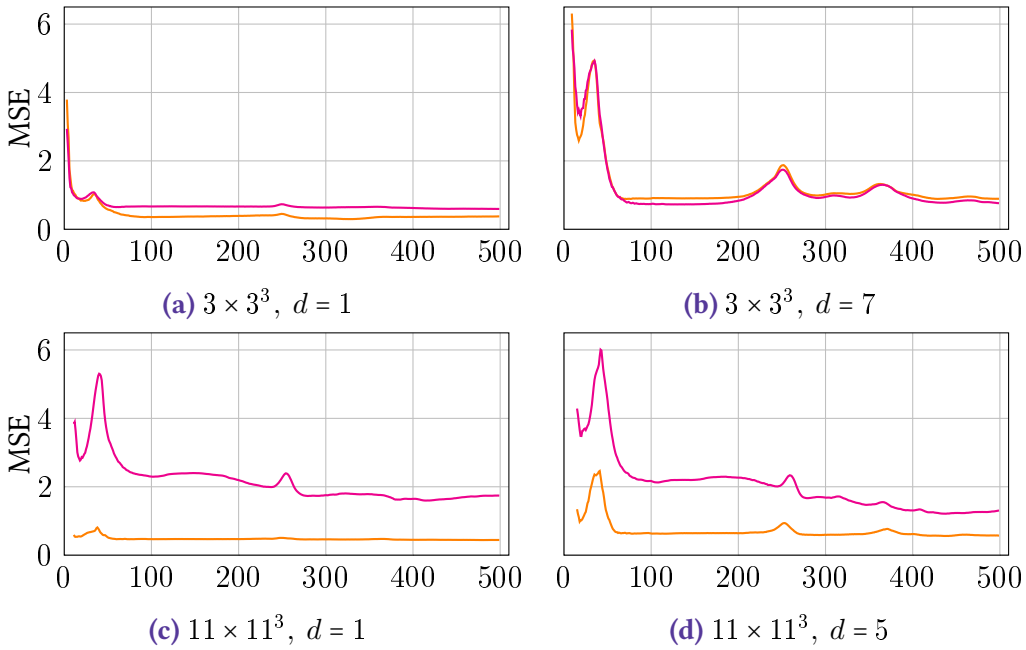
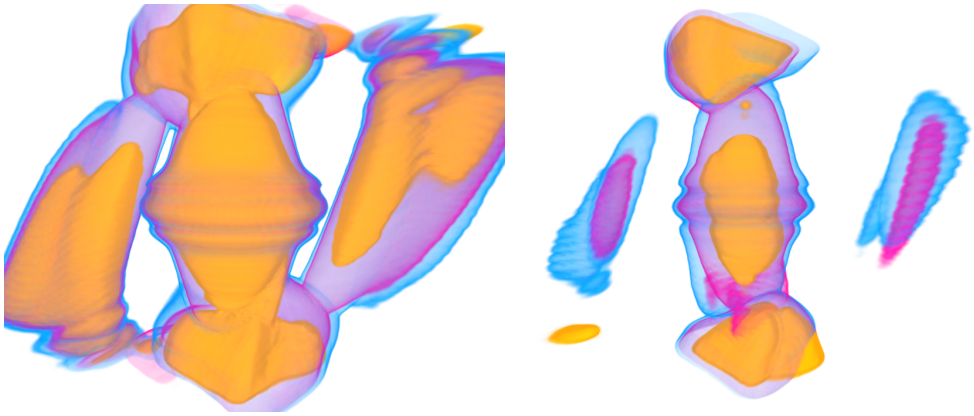


Figure 3.16 — Temporal misprediction for four different parameter configurations on the "hotroom" dataset (cf. Figure 3.15). With low prediction delay (**a, c**) prediction is too easy and leads to models showing few errors, missing an event around frame 370. With small patch size (**a, b**) there is little difference between simple (magenta line) and complex (orange line) models. Larger patch size and delay (**d**) allow for detection of all three events and distinction of temporal differences between the models.

We also investigated temporal misprediction graphs for four different settings using the most complex and the simplest models (Figure 3.16). As we can see, runs with small delay (Figure 3.16a, 3.16c) tend to have very smooth low-error curves and miss the stream emergence event around frame 370. The run in Figure 3.16b (due to its larger delay) highlights all three events, but both models provide identical information. The configuration with larger patch size and large delay (Figure 3.16d) further improves this result, with the complex model clearly distinguishing the three significant events, and the simpler model also showing a decreasing error trend as the air becomes more diffused. These findings align well with our parameter space visualization from Figure 3.15b.

In Figure 3.17 we also show the spatial misprediction views for two configurations: one with low and one with high error range criterion values. We observe that when the prediction problem is too hard (Figure 3.17a), similarly to the temporal view (Figure 3.16b), all models fail more often and in similar regions. But when we increase the patch size, and thereby error diversity, we can better separate different irregular regions, e.g., regions on the sides corresponding to later air stream formation are not



(a) $3 \times 3^3,\ d = 7$ **(b)** $7 \times 7^3,\ d = 5$

Figure 3.17 — Spatial misprediction views for configurations with low (**a**) and high (**b**) error diversity on "hotroom". When the prediction problem is hard and diversity is low, different models produce many large errors and highlight similar regions, however when the problem has suitable difficulty we can distinguish regions of different irregularity.

showing errors from the most complex model (in orange).

Crucially, as discussed above, most parameter configurations yield similar results, with particularly poor settings being easy to avoid using either our selection method or minor prior knowledge of the data. Furthermore, we found that the parameter space visualization provides further insight into the data and can be used to complement our core technique, e.g., we see that for the hotroom the prediction delay has a much smaller impact on the model error compared to vortex street, which points to a much smoother and predictable nature of the data.

3.9 Performance

Since our method involves both handling of spatiotemporal fields and training of neural network models, performance is an important aspect to consider. See Section 3.4 for more details about our efficient implementation. Here we provide performance measurements, shown in Table 3.2. All tests were performed on a machine with an Intel Xeon E5-2630 v4 CPU and an Nvidia Tesla V100 GPU. The dataset size is provided in the format Time × Width × Height × Depth, and p_u refers to the undersampling probability (Section 3.2.2). We also provide the number of training epochs that passed before the convergence condition was triggered (Section 3.4). We used separate configurations for 2D and 3D datasets, with stronger undersampling and a smaller patch size for the 3D datasets, which otherwise generate superfluous amounts of data for our relatively simple models. Overall, we can see that the prediction often takes a

Dataset	Dataset size	Patch size	Train set	p_u	Model*	Train (min.)	Predict (min.)	Epoch
droplet	400x256x256x256	$5x5^3$	712.1K	0.01	A	35.0	171.5	76
droplet	400x256x256x256	$5x5^3$	712.1K	0.01	B	79.6	170.7	173
droplet	400x256x256x256	$5x5^3$	714.0K	0.01	C	135.3	177.6	315
spheres	200x128x128x128	$5x5^3$	185.2K	0.01	A	7.3	16.6	60
spheres	200x128x128x128	$5x5^3$	185.2K	0.01	B	56.7	16.3	500
spheres	200x128x128x128	$5x5^3$	185.4K	0.01	C	16.5	16.3	146
bottle	465x450x215x1	$15x5^2$	3.4M	0.1	A	87.9	8.8	71
bottle	465x450x215x1	$15x5^2$	3.4M	0.1	B	234.2	8.6	200
bottle	465x450x215x1	$15x5^2$	3.4M	0.1	C	26.2	10.3	23
vortex st.	800x101x301x1	$15x5^2$	1.8M	0.1	A	115.9	5.8	173
vortex st.	800x101x301x1	$15x5^2$	1.8M	0.1	B	157.3	5.8	250
vortex st.	800x101x301x1	$15x5^2$	1.8M	0.1	C	31.4	5.6	52

Table 3.2 — Performance of our local prediction implementation with different datasets and models. We undersample the training data with probability p_u and cache prediction results for empty space. Large and dense 3D volumes require the most computation.
* *Model A*: C64-C64-D256-D256, *Model B*: D256, *Model C*: D1.

considerable amount of the execution time. This is due to the fact that we cannot undersample the prediction data and need to obtain a prediction for each data voxel. For datasets with large amount of empty space (e.g., droplet) we can reduce the amount of processing (see Section 3.4), while others do not allow this optimization.

For our parameter selection study (Section 3.8.2) on the vortex street dataset, we performed 80 runs, spending a total of 64 hours to train the models and 108 minutes to perform the predictions. For the hotroom dataset (500x181x91x181 in size), we used stronger undersampling ($p_u = 10^{-4}$) to perform 64 runs with a total of 54 hours to train and 56 hours to predict. Although the time required to sample the parameter space is significant, the task is highly parallelizable and has linear speedup, since each experiment can be performed on a separate machine, which we have avoided to provide comparable performance measurements.

3.10 Limitations and Future Work

Local prediction. Our choice of making the prediction based on local information yields several advantages, including training performance, translation-invariance and model simplicity (Section 3.2.2). It also introduces an additional parameter – the patch size, which can have an impact on prediction performance both in terms of execution time and accuracy (Section 3.8.1). Although locality is often a reasonable assumption

for scientific data, relationships extending beyond the patch size cannot be captured by our models. When such effects need to be accounted for, the models could be extended to incorporate global context, e.g., by using downsampled data.

Performance. A limiting factor of our approach is performance. The performance costs come from training multiple neural networks on large data, as well as evaluating them to obtain predictions (Table 3.2). Although smaller patch sizes combined with undersampling are often sufficient to produce accurate results, analyzing even moderately-sized datasets may take hours and days. Fortunately, the approach has large parallelization potential. Both neural network training and prediction can be distributed in terms of data across multiple machines, and parameter studies are even easier to parallelize. Finally, pretrained models can be utilized for applications within the same domain or simulation ensembles.

Temporal interpolation. Our approach operates by detecting regions that are hard to predict based on past events. We have shown that by varying the prediction delay we can vary the difficulty of prediction, increasing it for datasets with fine temporal resolution (Section 3.8.1). However, this also implies that in the opposite case of very coarse temporal resolution the prediction task can become too hard, causing the models to produce high errors everywhere, making it impossible to distinguish behavior of different complexity. An interesting extension to address this case could be to simplify the prediction problem by considering not only the past, but the future too, essentially performing interpolation, which would have lower resolution requirements than extrapolation.

Sequence modeling. For simplicity and generality, we used relatively basic neural network models (Section 3.2.1, 3.4). However, building upon our findings with simpler models, more complex and specialized architectures could be used to improve the results. For example, recurrent neural networks are being successfully used to model temporal data and might provide a more meaningful prediction for spatiotemporal volumes as well. Moreover, they can be applied to temporal sequences of variable length, which can produce additional information about the local temporal complexity by monitoring the length of accurately predicted sequences.

Physical constraints. While we have used models with no physical or domain-specific assumptions, there is a number of generic physical properties that could be explicitly modeled: energy and mass conservation, flow incompressibility, etc. These could be introduced to the model's loss function to aid in doing physically-accurate predictions, or to find spatiotemporal regions where the model violates these conditions.

Enhancing user control. Currently, the user can influence the results by defining models and some parameters before running our method, and explore the results interactively afterwards (Section 3.5). However, our approach could be extended to provide further control to the analyst. On the one hand, more traditional visualization

techniques like clustering can be applied to discover further instances of similar behavior or filter detected irregularities. On the other hand, ML-based approaches like one-shot learning could be explored, reusing the feature space learned by the models to detect more specific behavior based on the analyst's feedback. In the next chapter (Section 4.3), we present a different ML-based approach based on this idea. It can be used for similar applications, but shows greater performance and interactivity, allowing the user to directly query the ML model.

4

Learning Spatiotemporal Similarity Metrics

This chapter is based on the following publications:

Gleb Tkachev, Steffen Frey, and Thomas Ertl (2021a). "Local Prediction Models for Spatiotemporal Volume Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 27.7, pp. 3091–3108
Gleb Tkachev, Steffen Frey, and Thomas Ertl (2021c). "S4: Self-supervised Learning of Spatiotemporal Similarity." In: *IEEE Transactions on Visualization and Computer Graphics* DOI: 10.1109/TVCG.2021.3101418
Hamid Gadirov, **Gleb Tkachev**, Thomas Ertl, and Steffen Frey (2021). "Evaluation and Selection of Autoencoders for Expressive Dimensionality Reduction of Spatial Ensembles." In: *International Symposium on Visual Computing*. Springer, pp. 222–234

In many cases, scientists collect data not from just a single simulation run but from hundreds or even thousands of different configurations. These so-called ensembles are fundamental in studying the effects of varying input parameter values, boundary conditions, materials, etc. (Section 2.1.1). Visualization is essential in making sense of ensembles, and explorative analysis plays a particularly important role, since there is often only limited prior knowledge of the newly generated data.

Many of the common tasks in exploratory analysis, e.g., partitioning of the output space or outlier detection, require a measure of distance or similarity. In some cases, there

may be domain-specific measures. However, these are not generally available, and the development of such algorithms requires not only detailed domain knowledge but also a lot of time. Therefore, generic techniques are needed to support the exploration, especially when new types of data are involved. Standard metrics operating directly on the raw data like the mean squared error (MSE) or the earth mover's distance (EMD) are generic, but do not yield expressive results for most application contexts (Frey and Ertl 2017a, 2017b). For example, a phase shift between two members would produce a large quantitative difference, while qualitatively they are very similar. Instead, we propose to use machine-learning models, which are capable of extracting higher-level features from the data, allowing for a more meaningful comparison.

Learning-based approaches have been shown to deliver great results in the quantification of image similarity (Wang et al. 2014). However, such models are often pre-trained on photographic images and are not suitable for scientific data. And training new models on more appropriate data requires labels at some point in the process, which are typically not available for simulation and experimental data. The two approaches that are presented below are aimed specifically at addressing this challenge. The former (Section 4.2) is an extension of our prediction-based method from Chapter 3 and focuses on quantifying similarity between ensemble members. And the latter (Section 4.3), is a more general self-supervised method that can be applied to any large spatiotemporal datasets. Furthermore, in the third part (Section 4.4) we briefly discuss a related autoencoder-based method that focuses on visual exploration of ensemble data.

4.1 Related Work

Ensemble visualization and similarity measures. The analysis of data ensembles is a challenging visualization task (Obermaier and Joy 2014). Potter et al. 2009 and Sanyal et al. 2010 proposed some of the first approaches for climate ensembles, while Waser et al. 2010 demonstrated a system for interactive steering of simulation ensembles. Sedlmair et al. 2014 and Wang et al. 2019 provided detailed surveys of the techniques in the area.

In the context of ensemble visualization, a similarity measure often plays an important role, and so many methods have been explored. Bruckner and Moeller 2010 used squared differences to explore the visual effects simulation space, Hummel et al. 2013 determined similarity between regions via joint variance, Wei et al. 2017 efficiently computed similarity between local histograms and Kumpf et al. 2019 tracked statistically-coherent regions using optical flow. Jarema et al. 2015 utilized Gaussian Mixture Models to compute a similarity matrix for vector fields. And Wang et al. 2016 proposed a vector field similarity based on a 3D SIFT implementation. Hao et al. 2016 constructed octrees to calculate shape similarities for particle data, while He et al. 2020a employed surface

density estimates for distances between surfaces. Fofonov and Linsen 2019 developed an isosurface-based similarity for multi-fields. We also propose a similarity metric that can be used for spatiotemporal ensemble data; however, we use a learning-based approach that is domain-agnostic but capable of adapting to the dataset. Furthermore, we focus on the search for similar behavior, allowing the user to interactively influence the similarity score.

Video object detection. The problem of spatiotemporal similarity is related to object detection in video, which is extensively studied in the field of computer vision (Zou et al. 2019; Jiao et al. 2019). Nevertheless, scientific data presents a unique set of challenges. Some detection approaches target specific object categories such as people (e.g. Li et al. 2014; Ahmed et al. 2015; Huang et al. 2018), while in our domain-agnostic setting we cannot make similar assumptions. Other, especially deep-learning-based techniques (e.g. Han et al. 2016; Bertasius et al. 2018; Deng et al. 2019), can detect a diverse but fixed set of objects, while also requiring at least some supervised data. Finally, the computer vision techniques detect spatial objects in video by exploiting frame coherence, while we are interested in fundamentally temporal processes, thus treating temporal and spatial dimensions equally.

Self-supervised learning. In recent years, self-supervised learning has been gaining popularity, especially in computer vision. For example, Dosovitskiy et al. 2014 used random image transformations to generate surrogate image classes and learn a robust feature space. Misra et al. 2016 learned their representation by predicting if a sequence of video frames was given in the correct order, while Doersch et al. 2015 predicted the relative position of two image patches. Continuing the trend, many other self-supervised tasks were proposed in the following years (Doersch and Zisserman 2017). Our approach also uses the idea of self-supervised learning, but instead of fine-tuning the pretrained model on supervised data, we use the learned representation directly to find similar behavior.

ML in scientific visualization. See Section 3.1 for an overview of works on this topic and a discussion of how they relate to our methods.

4.2 Prediction-based Ensemble Similarity

Here, we propose a domain-agnostic (dis-)similarity measure for ensemble members that could be used to augment existing specialized techniques. The measure is an extension of our local prediction approach from Chapter 3. The main idea is to perform *cross-prediction* across the ensemble members: training a local prediction model on one member and then applying it on another, measuring the overall prediction error. This effectively estimates the difference between the behavior captured by the model from one member and the actual behavior of another member. Conceptually, the measured

error can also be thought of as the generalization error of the model: by observing how well the model performs on unobserved data we are estimating how different the data is from the training dataset.

4.2.1 Method

We train a separate local prediction model (Section 3.2.1) on each member of the ensemble. Then, using each model, we perform prediction for each member (including the one that the model was trained on) and compute the mean squared prediction error across the whole spatiotemporal domain. Thus, we end up with a square matrix of cross-prediction errors E, where each cell $E(i,j)$ specifies the error resulting from training on the i-th member and predicting for the j-th member. We further normalize the cross-prediction error by the error measured when training on the same data. This addresses the fact that some members can be significantly harder to accurately predict than others due to a higher amount of irregular behavior, regardless of which data the model was trained on (Section 3.3). This could lead to cross-prediction errors being large for a similar pair of complex simulations and low for a similar pair of simple simulations, although qualitatively the dissimilarity within both pairs should be the same. Therefore, we use the *relative cross-prediction error* E_r:

$$E_r(i,j) = E(i,j)/E(j,j). \tag{4.1}$$

Naturally, the cross-prediction error is not symmetric. This is not only a result of model training being a stochastic process, but also an important insight into the meaning of the error. A large cross-prediction error means that local behavior common to the predicted member did not occur in the training member (or occurred too rarely). This implies, that training on a simulation with diverse local behavior may produce a low cross-prediction when predicting for another member. The predicted member on average may be very different, but its local behavior is a subset of what the model has captured during training and can reproduce during prediction. Thus, to construct a dissimilarity measure from the cross-prediction error we consider the error in both directions:

$$\overleftrightarrow{E}_r(i,j) = \overleftrightarrow{E}_r(j,i) = \sqrt{E_r^2(i,j) + E_r^2(j,i)}. \tag{4.2}$$

This means that the dissimilarity between a pair of datasets is only low if both cross-prediction errors are low.

Having a dissimilarity measure for the ensemble, several visualization techniques could be applied. We chose to visualize the ensemble using a dissimilarity matrix encoded as a heatmap. This allows us to better evaluate our approach, since we directly visualize the values of our measure for different member pairs. To improve the visualization we also sort the ensemble members using hierarchical clustering.

Specifically, we perform agglomerative hierarchical clustering with centroids, i.e. when merging clusters, distance between clusters is defined as distance between the cluster centroids. The resulting tree is then sorted, such that the distances between adjacent leafs are minimized (Bar-Joseph et al. 2001). In other words, the ensemble members in the visualization are arranged such that the dissimilarity between neighbors is small.

4.2.2 Results

To evaluate our prediction-based ensemble dissimilarity measure, we have applied it to a CFD ensemble (similar to the "vortex street" dataset). The ensemble has three parameters: Reynolds number, obstacle offset and obstacle radius. To obtain a diverse set of simulations we have randomly sampled the whole parameter space, obtaining 25 simulations.

For computing our dissimilarity measure (Section 4.2) we trained an instance of the "C128-D256" model for each member (notation from Section 3.4), and performed a full set of cross-predictions (625 predictions in total). The results are presented in Figure 4.1. In the header we visualize a single timestep of each ensemble member and specify its three simulation parameter values. The same timestep has been used for all members, aiming to give an impression of the whole ensemble.

The first observation is that the dissimilarity metric led to ensemble members being roughly sorted according to how turbulent they are. The most turbulent members were put on the left, with members slowly becoming more laminar as we move to the right. Looking at the dendrogram in the header, three high-level clusters can be distinguished: strongly-turbulent (on the left), turbulent (middle) and mostly-laminar (right).

When inspecting the heatmap, several groups of simulations can be spotted. The most prominent group of members is (d - j), which has high dissimilarity to many other members outside of the group. This is due to their highly turbulent behavior that does not appear in other simulations. Importantly, the dissimilarity inside this group is low, showing that our measure captures the differences in local behavior, rather than just differences in value (direct difference between two turbulent members would still be high). Another significant group of simulations is (m - x), which represents members with mostly laminar behavior. Again, they have low dissimilarity in-between, and higher values to other members, especially towards the highly turbulent ones. Within the groups we can look at a few members that were added to the cluster later (because of their slightly higher dissimilarity to the rest), e.g. (v) and (w) that are significantly asymmetric.

Simulations (k, l) also form a low-dissimilarity pair and represent a less distinct case of slower turbulent results. We observe a medium dissimilarity to the most turbulent members, as well as to the mostly laminar members. Presence of some turbulence

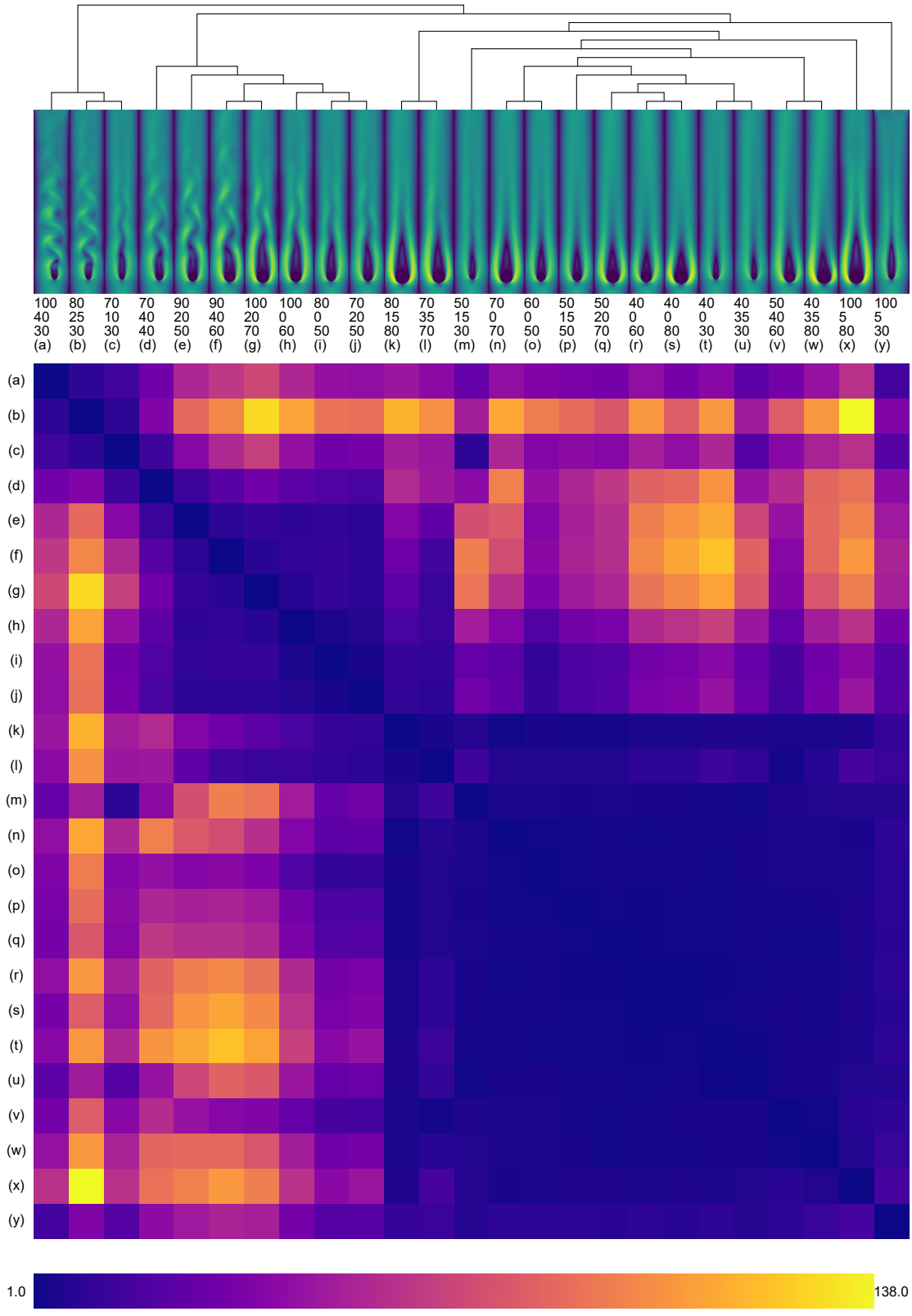


Figure 4.1 — Our dissimilarity measure for a flow simulation ensemble. The top row shows one timestep from each simulation. Groups of members that exhibit similar behavior (e.g., turbulent) have low dissimilarity (blue) to each other, but high dissimilarity (yellow) towards other members.

helps the models trained on them to perform better on the turbulent members than the laminar-trained models. Similarly, member (o) is also interesting in that it exhibits an in-between case: although it belongs to the group of laminar members, it has lower dissimilarity to the turbulent ones than the rest. During most of the simulation time it exhibits laminar flow, but turns turbulent towards the end. For this reason, the model trained on simulation (o) was more successful in predicting the turbulent ensemble members, akin to (k) and (l). Another special case is simulation (b) which is the most turbulent member of the ensemble with very distinct local behavior that leads to large prediction errors for most models. Simulation (c) is similar to it in terms of turbulence, however, it has a longer laminar setup phase, leading to lower dissimilarity values overall (laminar models can still predict the first part correctly). Finally, interesting outliers are members (a) and (y), which were sorted away to the left and right sides by the hierarchical clustering algorithm. Initially, they appear as a strongly-turbulent member and a laminar member respectively. However, closer inspection showed that the simulation files were corrupted, resulting in "flickering" that has not been observed in the ensemble before, but was identified via our visualization. These artifacts caused increased errors when predicting on the data, even for the model trained on it, thus resulting in medium normalized cross-prediction error for most other ensemble members. Overall, we found that our dissimilarity metric produces results that correspond to what is intuitively and qualitatively expected from comparing pairs of ensemble members, which allows us to identify prominent features of the ensemble.

4.3 Self-supervised Learning of Similarity

In this section, we present another learning-based approach to spatiotemporal similarity. In contrast to the cross-prediction method from Section 4.2 that computes similarity between ensemble members, the new method operates on spatiotemporal patches. Thus, it can better localize the behavior of interest and can also be applied to non-ensemble data. Moreover, the similarity is guided by user-constructed queries and can be computed quickly to perform the search interactively. We evaluate our approach both qualitatively with a domain expert and quantitatively by comparing to several baselines on both simulation and experimental ensemble data.

4.3.1 Overview

As described in the introduction of Chapter 4, machine learning can produce domain-agnostic methods that adapt to the data, which can be a significant advantage. However, they often require suprvised training data that is usually not available for scientific datasets, especially newly generated ones. Indeed, it would be cumbersome and unrealistic for a domain scientist to meticulously label every region of their dataset, which

can have high resolution in space and time, as well as numerous members in the case of ensemble data. This would completely defeat the purpose of a domain-agnostic visualization system.

To overcome the problem of missing labels, we propose to use self-supervised learning. The motivation for self-supervised learning comes from the fact that unlabeled data is unusable by typical supervised models, and yet most of the data is unlabeled. How can we utilize this large unlabeled data to improve our models? The answer of self-supervised learning is to define an artificial task (the *pretext* task) on unsupervised data that does not require manual labels. For example, predicting if two images are transformed versions of the same original image to learn transformation-invariant features. We can then train a supervised model on the pretext task and use it to help solve the target task, typically by fine-tuning it on a much smaller labeled dataset (Doersch and Zisserman 2017). However, we do not have any labeled data available. Instead, we directly use the feature space (representation) learned by the model and compute distances in that space as our similarity metric. To make sure that these distances correspond to similarity, we use the siamese architecture for our model.

Siamese networks (Bromley et al. 1993; Chopra et al. 2005) are a method of learning similarity. They consist of two identical sub-networks that are joined at the output. Each sub-network takes an input (e.g., an image) and encodes it into a feature space. Then, a distance is computed between the pair of encodings to output their similarity. This setup allows us to train the sub-network (which we call *the encoder*) and also encourages a feature space with useful distances, since we explicitly constrain the model to rely on distances when solving the task. Typically, a siamese model is trained on a labeled dataset of known similarities, but we do not have such data and instead train the model on a self-supervised pretext task.

The choice of the pretext task is critical to the learned representation and thus the final performance of our method. Most importantly, we need a task that is as similar as possible to our target task because similar tasks require similar information to be encoded by the model. One should also consider the invariances imposed by the task on the representation, as they will determine what information gets preserved or discarded. With this in mind, we designed our task to be a binary classification problem: "given two spatiotemporal regions of the data, are they nearby in space and time?". More concretely, the model is provided with two rectangular spatiotemporal patches (boxes) from the data and needs to determine whether they originate from the same ensemble member and within a certain spatiotemporal window from each other. This task is closely related to detecting similar behavior because nearby locations often contain similar behavior. Additionally, we explicitly encourage temporal and translational invariance of our representation, which helps detect similar processes that are not perfectly aligned. Note that other formulations are possible, e.g., predicting the distance of patches in space/time (regression) or using contrastive loss, but we prefer a

classification formulation for practical reasons. It is simple and produces an intuitive performance metric – accuracy, which allows us to make sure that the model is solving the pretext task during training. In contrast, raw MSE or cross-entropy values would be more difficult to interpret.

The result of the self-supervised training of our siamese model is the encoder that enables us to compute distances between data patches in the learned feature space. Of course, one cannot expect this distance to provide a perfect solution for the target task of finding similar behavior, but we demonstrate below that it is meaningful and is a significant improvement over distances computed directly on the raw data (Section 4.3.6, Section 4.3.7 & Section 4.3.8).

We can apply our technique to any type of a spatiotemporal dataset that is large enough for training, but we focus on ensemble data because it allows us to most effectively demonstrate the utility of our learned similarity. Ensemble datasets often contain many distinct types of behavior but are typically too large to manually search through. We will demonstrate that our approach can detect these behavior types without any domain-specific assumptions. More specifically, we describe and evaluate our approach with 2D+T ensemble data as it enables us to effectively present our results, but we also show that our approach works with 3D+T data (Section 4.3.7). As mentioned above, our similarity metric could be useful in a large variety of visualization applications. To exemplify one of these applications and better demonstrate the properties of the learned metric, we use it to perform search for occurrences of similar behavior. We developed a prototype system that enables the user to do this search interactively to explore an ensemble dataset (Section 4.3.5).

Our prototype's main feature is the example-based search over spatiotemporal patches, i.e., 2D+T rectangular subsets of the data. The user can select a few patches containing behavior of interest, and the system returns others containing similar behavior. The similarity is determined using our metric, encoding the query patches and measuring distances to other patches in the learned feature space. While the prototype has only basic features, it showcases our learned metric and its possible applications.

4.3.2 Pretext Task and Model

The key component of our approach is the model that we train on the pretext task. As indicated above in Section 4.3.1, the pretext task is to predict whether two spatiotemporal patches originate from nearby locations in space and time. As our model, we use a neural network that follows the siamese architecture. A typical siamese network takes two inputs (in our case, 2D+T patches), passes them through an encoder and computes the distance between the encodings. Crucially, both inputs use the same encoder, i.e., they share weights. This ensures that the inputs are projected into the same space in which distances will be computed.

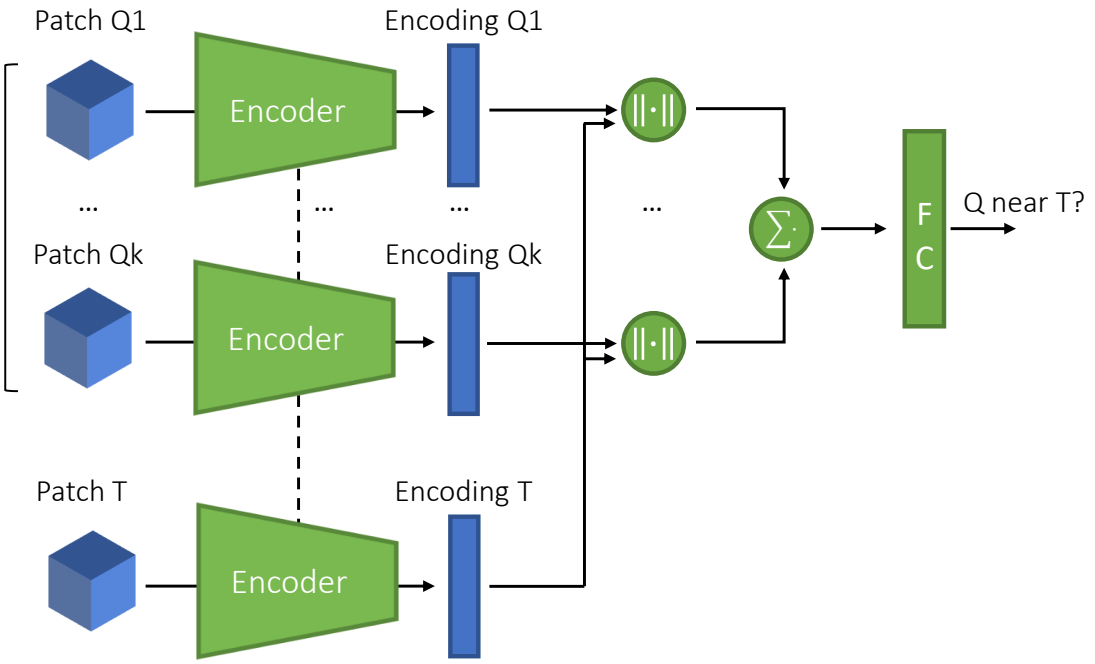


Figure 4.2 — Our model architecture. We encode the query (Q1, ..., Qk) and the test (T) patches using a convolutional encoder. Then L1 distances from each encoded query patch to the encoded test patch are computed. Finally, the average distance is used by the fully-connected layers (FC) to predict if the query and the test patches are located nearby.

In our search application, we will be computing distances to several patches provided by the user, not just one. To improve the performance in this regard, we deviate from a typical siamese model. Specifically, we replace one of the input patches with a set of k patches, here called *the query*. During the training, these patches are sampled from the same spatiotemporal neighborhood, and the network's task is to predict whether they also share the neighborhood with the other input patch, called *the test patch*.

An illustration of the model architecture is presented in Figure 4.2. Each input patch (the query patches and the test patch) goes through the encoder, which is a convolutional network. All invocations of the encoder use the same shared weights. Next, we compute an element-wise distance between each of the query patch encodings and the test patch encoding. These distances are then averaged (again, element-wise) and used as input for a fully-connected layer that acts as a simple classifier predicting whether the inputs came from the same neighborhood. We use element-wise L1-distance following Koch 2015. The element-wise aspect is particularly important in our self-supervised case: if instead we would have a vector distance producing just a scalar value, the pretext task will have to be solved in the encoder already, which would specialize the learned

representation to the pretext task and make it less general. Thus, we use the element-wise distance to let more information pass through the distance operator, shifting the pretext task decision towards the classifier.

4.3.3 Model Training

The model is trained on data sampled from an ensemble of spatiotemporal volumes. Each data point consists of several patches: the test patch and the set of k query patches. Half of the points represent the positive class, i.e., query and test patches coming from the same neighborhood, and half represent the opposite class. In either case, we first determine the location of the test patch by uniformly sampling a random ensemble member and then uniformly sampling a spatiotemporal location within it. This makes sure that ensemble members of different sizes are equally represented in the training data. For points of the positive class, the query patches come from the same member as the test patch, sampled uniformly from a spatiotemporal neighborhood around the test patch. The neighborhood is defined by the maximum spatial and temporal offsets o_s, o_t. This means that all query patches are at most o_s cells and o_t timesteps away from the test patch. For negative points, we first randomly choose an anchor location that is not in the neighborhood of the test patch. We then uniformly sample k patches around the anchor location with maximum offsets o_s, o_t, like the positive case but around a different location. Thus, the query patches always represent similar behavior, which makes them more suitable for distance averaging and improves our latent space. When choosing the offset parameters, we generally try to make offsets larger to make the pretext task more difficult while still training to high accuracy (in Section 4.3.8 we also show that the method is robust to the choice of parameters).

After the data is collected, we train the model using standard ML practices. We use 20% of the data as a hold-out validation set to monitor the model's generalization performance and perform early stopping. Overall we observed that higher accuracy and lower overfitting on the pretext task leads to better performance of the derived similarity metric. Implementation details of the model and its training follow in Section 4.3.5.

4.3.4 Similarity Metric

Once the model is trained, we use the encoder to compute patch similarity. With encoder f_e and two spatiotemporal patches p_1, p_2, our patch similarity metric d (technically, dissimilarity) is determined by the L1-distance between the encoded patches:

$$d(p_1, p_2) = \| f_e(p_1) - f_e(p_2) \|_1 . \tag{4.3}$$

As described in Section 4.3.1, we demonstrate the utility of this metric by performing interactive queries. To provide results for user queries, we use our similarity metric to

Layer	Units/Ch.	Activation	Kernel size	Strides
Conv3D	64	ReLU	(1, 3, 3)	(1, 2, 2)
Conv3D	128	ReLU	(1, 3, 3)	(1, 1, 1)
Conv3D	128	ReLU	(3, 3, 3)	(2, 2, 2)
Conv3D	256	ReLU	(1, 3, 3)	(1, 2, 2)
Reshape				
FC	256	ReLU		
FC	256	Sigmoid		

Table 4.1 — The architecture of our convolutional encoder, top-to-bottom. We use a sequence of strided convolutions followed up by a few fully-connected layers to encode each input patch.

construct a ranking score. A query Q consists of two sets of spatiotemporal patches: a set of positive examples Q_+ containing behavior the user is interested in; and an optional set of negative examples Q_- containing behavior that the user would like to exclude. The negative examples can help the user narrow down the query and filter out false-positive results. We define the ranking score r of some patch p to be the average similarity metric between the patch and the query, where the sum of distances of negative patches is subtracted from the sum of distances of positive patches:

$$r\left(Q_+, Q_-, p\right) = \frac{1}{\|Q_+\| + \|Q_-\|} \left(\sum_{p_+ \in Q_+} d(p, p_+) - \sum_{p_- \in Q_-} d(p, p_-) \right) \tag{4.4}$$

The ranking score is therefore low for patches that are similar to the positive part of the query and dissimilar to the negative, indicating a good match. The score is normalized to make the values more comparable across queries of different sizes. Note that during the model training we used only a single set of patches but sampled both positive and negative examples to learn a good similarity metric. This implicitly supports our ranking score that has the positive and the negative parts.

4.3.5 Implementation & Prototype System

Network implementation. The architecture of our convolutional encoder is depicted in Table 4.1. Since the ensemble datasets in the evaluation are all 2D+Time, we use 3D convolutional layers to perform both spatial and temporal convolutions, taking an input with dimensions $batch, time, height, width, channels$. For the 3D+T implementation in Section 4.3.7 we replace all the 3D convolutional layers (2D+T) of the encoder with 4D convolutions (3D+T), using the same strides and sizes. The 4D convolution takes a tensor with dimensions $batch, time, depth, height, width, channels$, and is implemented as a series of 3D convolutions along the time dimension. The output of the final fully-connected layer is used as the encoding of the input. After applying the encoder to

each patch, we compute element-wise L1 distance between each query patch and the test patches (see Figure 4.2). The distances are then averaged componentwise. Finally, the average distance vector is passed to the classifier. The classifier is a single fully-connected layer with one unit that has the sigmoid activation. During the training, we set the number of query patches $k = 2$ for all our models. Increasing the k further did not harm the final performance but also did not seem to improve it, so we chose the lower value.

The process of sampling training data is described in Section 4.3.3. For all datasets, we downsample spatially by the factor of two and then sample $500,000$ data points (50,000 for the non-ensemble Isabel dataset), each consisting of $k + 1$ input patches. We split the points 50:50 between the positive and negative classes, but other ratios could be explored. We also perform data augmentation: patches are randomly mirrored along the spatial axes, and their values are scaled with a random coefficient $\in [0.5, 1.5]$. We train the model using the Adam optimizer (Kingma and Ba 2014) with a learning rate of 10^{-4} and a batch size of 32. As a loss function we use binary cross-entropy and an L2-regularization term with a lambda of 2×10^{-4}. During training, we monitor the loss on the held-out validation set and terminate the training when the validation loss stops improving.

To perform search using the ranking score from Section 4.3.4, we first need to obtain the encodings of all the candidate patches. We can choose how densely to sample the candidates, trading off matching time for spatial precision. We extract patches that are non-overlapping in time. When using patches that do not cover the whole spatial domain (Section 4.3.6), the patches are extracted with spatial stride equal to the quarter of their size. This introduces some redundancy, since the patches overlap, but enables us to better study the spatial properties of our method.

Prototype system. To better demonstrate the utility of our ranking score and learned similarity metrics in general, we developed a prototype system for navigating ensembles of spatiotemporal data. This is a challenging task, especially for ensembles with a large number of members, making it prohibitive to manually search for the behavior of interest. Our system addresses the problem by allowing the user to find instances of the behavior with only a handful of examples. This reduces the effort for the user, as well as avoids rendering large amounts of data at once. It also reduces the risk of missing good matches compared to the manual search.

In our prototype, we limit ourselves to working with patches that spatially cover the whole domain, i.e., temporal slices (we investigate spatial aspects in Section 4.3.6). The front-end interface of the system is shown in Figure 4.3, with the cylinder dataset loaded (depicting flow around a cylinder at the bottom of the domain). The navigation panel (1) is the main component of the UI. Here we present each ensemble member as a row containing a list of renderings of its timesteps. The user can click on a timestep

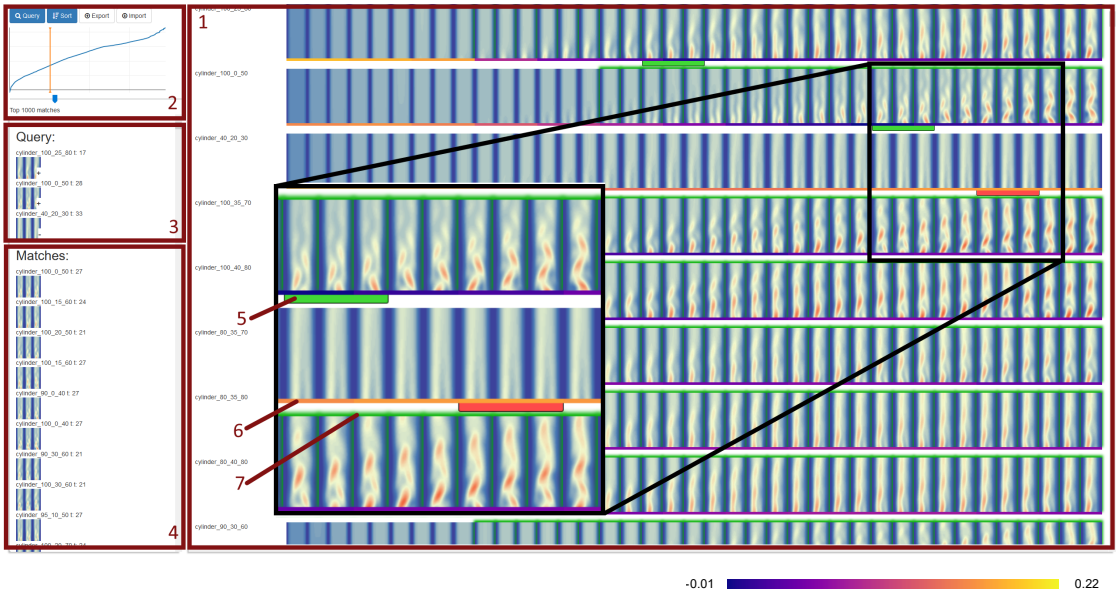


Figure 4.3 — Our interactive prototype for navigating an ensemble of spatiotemporal data, demonstrating query results on the cylinder dataset. In the navigation panel (**1**) we render a timestep rollout of all the members, showing markers for the query patches (**5**), ranking score (**6**, colorbar below) and highlighting the timesteps that match (**7**). In panel **2** we show sorted ranking scores of all patches in the ensemble. This gives an impression of their distribution and allows us to select which patches are considered to be a match. In panels **3** and **4** we show the contents of the query and the full list of matching patches.

to include a patch starting on this timestep into the query. We show the contents of the query using green and red marks (5) in the navigation panel, as well as a separate list in the query panel (3). The green marks correspond to positive examples and red to negative. Once the query is formed, the user can use the controls (2) to execute the query, loading the results from the back-end. The results are presented in several ways. First, we encode the ranking score (similarity) of every patch in the ensemble as a colored bar right below the timestep renderings (6, colorbar below). The best N matching patches are highlighted with green (7) in the navigation panel and are also shown as a list in the matches panel (4). To allow the user to configure the value N, we show the ranking score graph (2), plotting the sorted ranking scores of all the patches in the dataset. This gives an overall impression of the distribution of ranking scores and enables the selection of the cut-off line, i.e., how many top matches to display. Finally, the user can also sort the members in the navigation panel, according to the current query: members mentioned in the query itself are shown on top, followed by the members with the most matches. This makes it possible to quickly see the most relevant members for the current query and its results.

4.3.6 Qualitative Evaluation

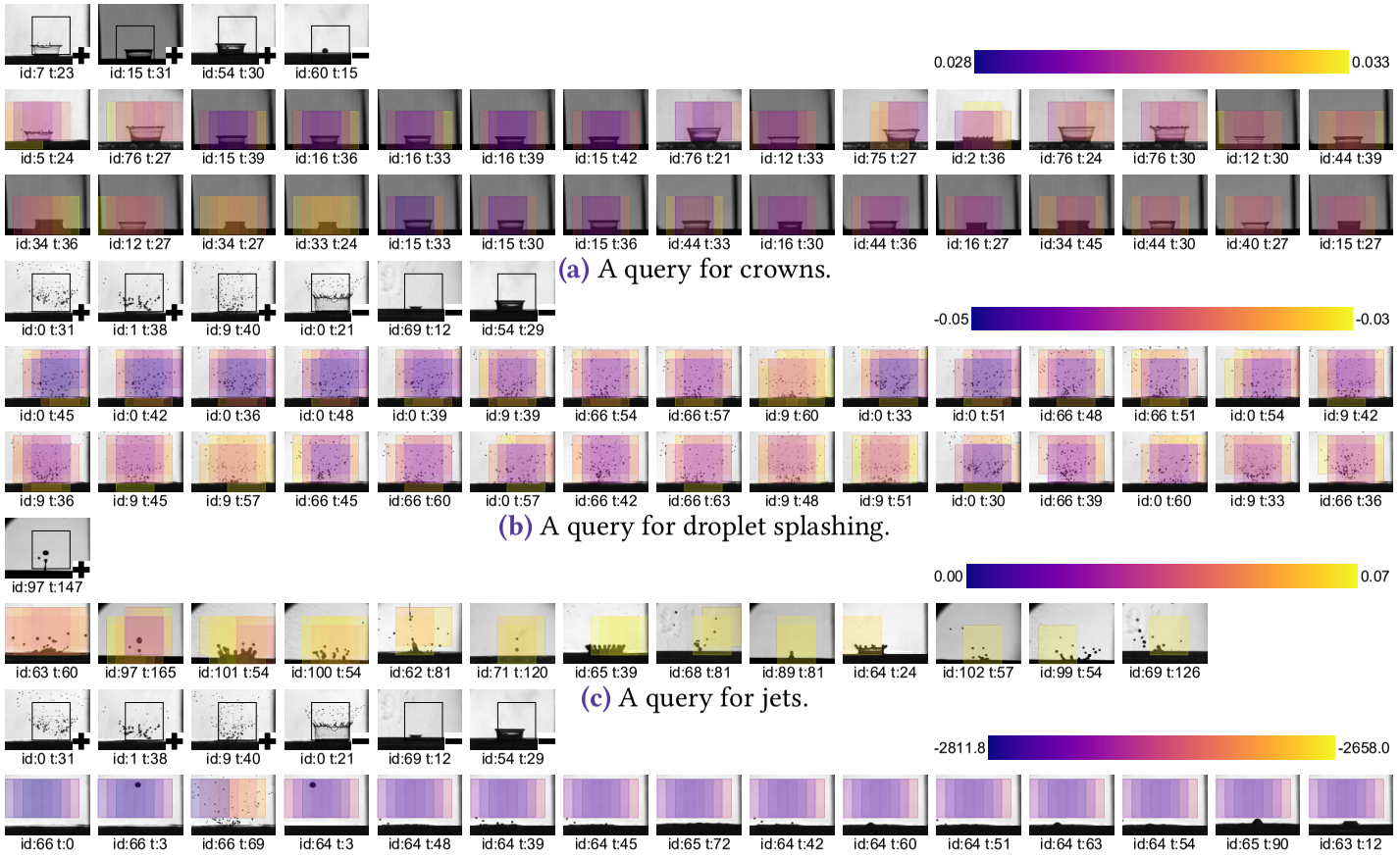
In this section, we perform a qualitative evaluation, demonstrating specific queries and their outcome (Section 4.3.6). We then show how our approach compares to results manually crafted by a domain expert (Section 4.3.6) and discuss her feedback (Section 4.3.6).

Query Results

First, we demonstrate the search results on the "droplet splash" dataset, which is an ensemble of experiments containing monochrome camera images of a single droplet impacting a thin liquid layer. Using different fluids, droplet size and impact velocity result in different impact regimes such as deposition, bubble formation or splashing, i.e., secondary droplets separating. There are 110 members, each ranging from 44 to 538 timesteps with a spatial size of 224×160. We used patches with a temporal size of three timesteps and a spatial extent of 50, and the neighborhood size is defined as $o_t = 9, o_s = 60$. This experimental ensemble is particularly challenging, as it contains images with differences in panning, zoom and illumination. Conventional similarity metrics often yield poor results under these conditions (see discussion below). However, since our model is trained on the dataset, it is able to learn some of these invariances and performs robustly.

In Figure 4.4 we present the results of different queries. First, we performed a query searching for fluid crowns (Figure 4.4a). The query contains three positive examples of crowns and one negative example with the initial droplet collision that occurs just before a crown is typically formed. In the results, we present the best matching patches rendered in the context of the data. Specifically, we took 500 best matching patches and grouped them by the timestep they begin on. We render timesteps that have the highest count of matches, highlighting the matching patch locations. The patch rectangles are colored according to their ranking score and rendered in worst-to-best order. Thus, in the locations where multiple patches overlap, we see the best-ranking score for that location. We observe that the results contain examples of crowns with varying shapes coming from many different members (specified by their ID). Despite the significant optical differences between the experiments, the method is able to detect relevant behavior. Furthermore, we see that beyond determining the member and the timestep where the behavior took place, we also receive spatial information: the best matching patches are well aligned spatially with the feature of interest. For an example of constructing queries interactively, see the supplementary video.

Another query is shown in Figure 4.4b. Here we search for instances of splashing, i.e., secondary droplets forming after the collision. We see that the results again contain a diverse set of patches matching our query. Even though direct differences between the patches with secondary droplets are large, our method can still find them. We found a



(a) A query for crowns.

(b) A query for droplet splashing.

(c) A query for jets.

(d) MSE results for the query from **b**.

Figure 4.4 — Query results on the droplet splash dataset. In the top left, we render the query patches (their first timestep). We color the matching patches based on their score, where blue means better matches. As we see, our method finds many diverse examples of the queried behavior, while MSE produces trivial results.

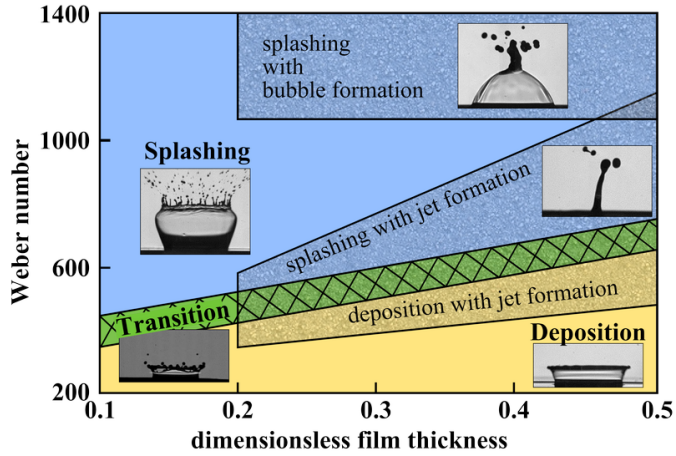
few examples that initially look like crowns (see also appendix), but on closer inspection, they turn out to be crowns that have just transformed into a spray of secondary droplets, thus matching the query. This hints at a property that follows from the training process: since we train the model by encouraging spatiotemporal coherence, we expect the matching to be somewhat "fuzzy", also having low distances to spatially and temporally neighboring behavior.

For comparison, in Figure 4.4d we show the same query but using MSE as the similarity metric for the ranking score. The search does not yield meaningful results, among others returning patches with empty space, since MSE heavily prioritizes the background and misses the droplets. It returns one valid timestep by matching the background, but this is only a rare outlier. In contrast, the top model-based results (Figure 4.4b) contain precisely the relevant patches with splashing droplets.

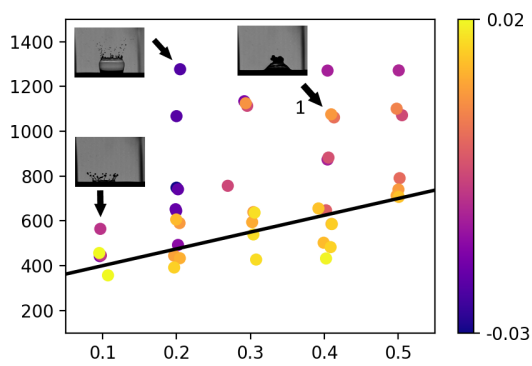
Although crowns and secondary droplets are the key features of this dataset, we also want to investigate if the method is capable of finding rare events. We constructed a query with just a single patch containing a fluid column with a droplet separation (Figure 4.4c). This figure, shows only the best frame of each matched member to conserve space (full result in the appendix). Here we note two types of matches. First, we see some crowns that have distinct fluid columns with droplets separating (id 63 and 101). Since separation occurs simultaneously in several locations, we get a lot of matches, and the corresponding timesteps get sorted to the top. Then, we also see some matches with a single fluid column. Some are from the query member (id 97), but some point towards other members with similar behavior (id 62, 71 and 89). Despite this event being much less typical for the ensemble, our approach was able to find similar instances with just a single example.

Next, we evaluate the temporal aspects of our metric on the "cylinder" dataset. This dataset is a CFD simulation ensemble of flow around a cylinder obstacle. Depending on the Reynolds number and the obstacle configuration, one observes different degrees of turbulence in the flow. We cropped the spatial domain, such that the obstacle is no longer visible (only the channel), so that the matching algorithm cannot "cheat" by comparing the obstacles. We used 300 ensemble members, where each member is a scalar velocity magnitude field with 39 timesteps and the spatial domain size 84×220. For this dataset, we focus on behavior extending over the whole spatial domain, so we use patches with the temporal size of three and spatial size equal to the data domain size, while the neighborhood size is $o_t = 9$. In Figure 4.3, we used our prototype to perform a query for turbulent members, providing two examples of turbulence and one negative example containing laminar behavior.

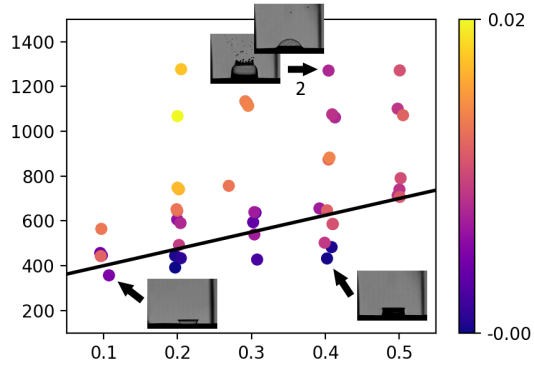
As we can see, after sorting the members by the number of matching patches, we find many other members containing turbulent behavior. Notice that the exact geometry of the flow can be very different, but the model still considers them similar due to



(a) Results by the domain expert (Geppert et al. 2017).



(b) A query for splashing.



(c) A query for deposition.

Figure 4.5 — Comparison of manual domain-specific results to our method. **a:** Parameter space map of droplet impact regimes manually constructed by the domain expert. **b:** Ranking scores of ensemble members for a splashing query (see Figure 4.4a). We find structure similar to (a), with best matching experiments (in blue) being located in the top-left quadrant of the parameter space and a similar transition region (the transition line from (a) plotted for comparison). We also found a poor-matching outlier region (marked "1"). A closer investigation led to the discovery of a region containing bubble formation and deposition that was previously unknown to the expert. **c:** Ranking scores for a deposition query. We find best-matching experiments to have lower Weber number or higher film thickness, in accordance with domain-expert results. A few experiments (marked "2") are matched moderately well by both queries because they display both bubble formation and splashing.

invariances learned during training. We also observe that the matching is successful temporally: we identify the point in time when the turbulence starts to occur.

Parameter Space Analysis

Next, we evaluate our results on the "droplet splash" dataset in comparison with the extensive manual analysis by a domain expert. This dataset was collected to study droplet impact regimes wrt. experimental parameters such as fluid viscosity, droplet velocity, film thickness, etc. In the previous analysis (Geppert et al. 2017), the expert has taken a subset of the ensemble depicting a particular fluid and has manually constructed a regime map of the parameter space, shown in Figure 4.5a. As we change the droplet velocity (Weber number) and the film thickness, we observe qualitatively different outcomes, with a thicker film and a slower droplet leading to cleaner deposition.

To compare our method's ability to detect different impact regimes, we performed two queries, one for splashing (also in Figure 4.4b) and another one for deposition. Then, we computed a ranking score for each ensemble member by simply taking the minimum score of all the patches from a given member. Thus, if a member contains a well-matching patch, the member itself is considered to be well-matching. We visualize the member scores positioned in the parameter space to compare them to the regime map that was constructed manually by the domain expert.

In Figure 4.5b we show the splashing query results. The best-matching members are located in the top-left corner (high velocity, thin film), and we can see that the score degrades as we move bottom-right, forming a transition region, which aligns well with the expert results. We plot the transition line from the expert map to make the comparison easier. Here we noticed a region with some poorly-matching outliers in the top-right quadrant (marked with "1"). After consulting with the domain expert and checking the experiment images, we found out that there is a bubble deposition subregime, which explained why the splashing query was not matching well. More importantly, its existence was previously unknown to the expert, highlighting one of the strengths of our method: being able to detect features of the data that the domain expert might have overlooked.

In Figure 4.5c we show results of a deposition query. Here, as expected, we get an inverse result: best-matching members contain clean deposition and are located in the bottom right corner of the parameter space (low velocity, thick film), and the splashing members (top-left corner) have high scores, again corresponding well to the results by the expert. We found one interesting outlier (marked with "2") that is matched reasonably well by both the splashing and the deposition queries. Upon inspection, we found that it contains a splashing phase, followed by a bubble phase, and since we use the minimum function to aggregate the patch scores for this figure, the member was matched well by both of the queries, each matching an appropriate time range.

Overall, we confirmed that our technique yields meaningful results when applied to data from this domain, successfully finding different types of behavior with a few simple

queries. The results exhibit a high degree of similarity to the manually-constructed regime map, which the domain expert also pointed out (Section 4.3.6).

Domain Expert Feedback

As part of our evaluation, we discussed our results on the droplet splash data with a domain expert in droplet dynamics. She collected and extensively analyzed the experimental ensemble data. We performed searches for several droplet regimes (e.g. Figure 4.4), which were previously studied by manually selecting relevant members and time ranges that were fed into ad-hoc Matlab scripts for further analysis. The expert noted very good agreement for detection of droplet splashing and deposition with bubble formation and fair results for queries of crown-forming deposition and jet formation. However, the query for splashing with bubble formation had a lot of unexpected matches. Here our method produced many matches containing crown-forming splashing, confusing bubbles with transparent crowns. Our parameter space comparison (Section 4.3.6) was viewed very positively: "In my opinion, the results are very good. The splashing limit is quite well reproduced in your maps.". The expert also pointed out that we found a small region that was previously inaccurately classified: "You detected a small region of bubble formation and deposition, which we did not recognize or to be precise which we counted as splashing".

She was very optimistic about the utility of the system in her workflow: "I think such a tool would have been and would be very useful for us because all the regime maps and splashing limits were derived by manually digging through every video and deciding what we see. [...] A tool like yours would make it much easier if we are looking for example at a special feature like jet formation." As a suggestion, she noted that the current results for splashing with a bubble could be further improved by performing iterative searches, i.e., first searching for a generic regime and then narrowing down the results with more specialized queries. Overall, we received very positive feedback about our technique, suggesting that our domain-agnostic method can act as a useful building block in addressing domain-specific problems.

4.3.7 Comparative Evaluation

In this chapter, we aim to demonstrate that machine learning can be used to construct an effective spatiotemporal similarity metric in a domain-agnostic fashion. This section discusses existing approaches to computing a spatiotemporal similarity metric and perform a qualitative comparison to some of these techniques. In the next section (Section 4.3.8), we also compare quantitatively to many more alternative methods.

Alternative Approaches

A similarity metric is an important component of many ensemble visualization approaches. Thus many other possibilities have been proposed. These can be roughly split into two categories. First, there are mostly generic vector, image or distribution distances, such as MSE (L2), L1, EMD, SSIM and local histogram differences. Such similarity metrics can be applied directly to the raw data and are a very common choice in modern ensemble visualization literature (see Table 2 in the survey by Wang et al. 2019). The other group of methods extracts domain- or problem-specific features from the data (e.g., vorticity or λ_2 for CFD datasets) and computes distances in that feature space.

The former metrics are the most appropriate to compare with our technique, as they are widely used in recent work and are similarly domain-agnostic. We compare our method to many of these metrics in our quantitative evaluation in Section 4.3.8. The comparison to the latter group is less appropriate since we are proposing a general domain-agnostic technique. While problem-specific solutions might outperform a general method, they cannot be applied to data from different domains, which is the core motivation for a learning-based similarity. Nevertheless, we still compare with two specialized methods in Section 4.3.7 and Section 4.3.7. We have chosen these approaches because they rely on a commonly used algorithm (SIFT), are somewhat data-agnostic (within their domain) and do not rely solely on basic distance metrics, improving the diversity of our evaluation. With this, we demonstrate that our generic metric yields similar performance while not relying on domain-specific features.

Comparison to SIFT

In this section, we compare the results of our similarity metrics to those computed with SIFT (Lowe 2004). SIFT is a computer vision technique for computing image keypoints and their local feature descriptors that are invariant to illumination, scale and orientation changes. Since SIFT is a robust matching algorithm that can, in principle, be applied to any scalar field, it provides an interesting comparison for our approach.

We applied the SIFT algorithm (which computes sparse image correspondences) to perform our patch-based queries as follows. First, we compute SIFT keypoints for all the timesteps and all the members of the ensemble dataset. By computing the keypoints over the whole spatial domain (as opposed to a given patch), we make sure that any large-scale keypoints would still be correctly extracted. Next, we define the SIFT distance between two patches as the minimum SIFT-descriptor distance between any keypoints lying within the patches. Given a query consisting of several positive and negative examples, we compute the ranking score according to Equation 4.4, i.e., the average SIFT distance from all the query patches to a candidate patch, where negative

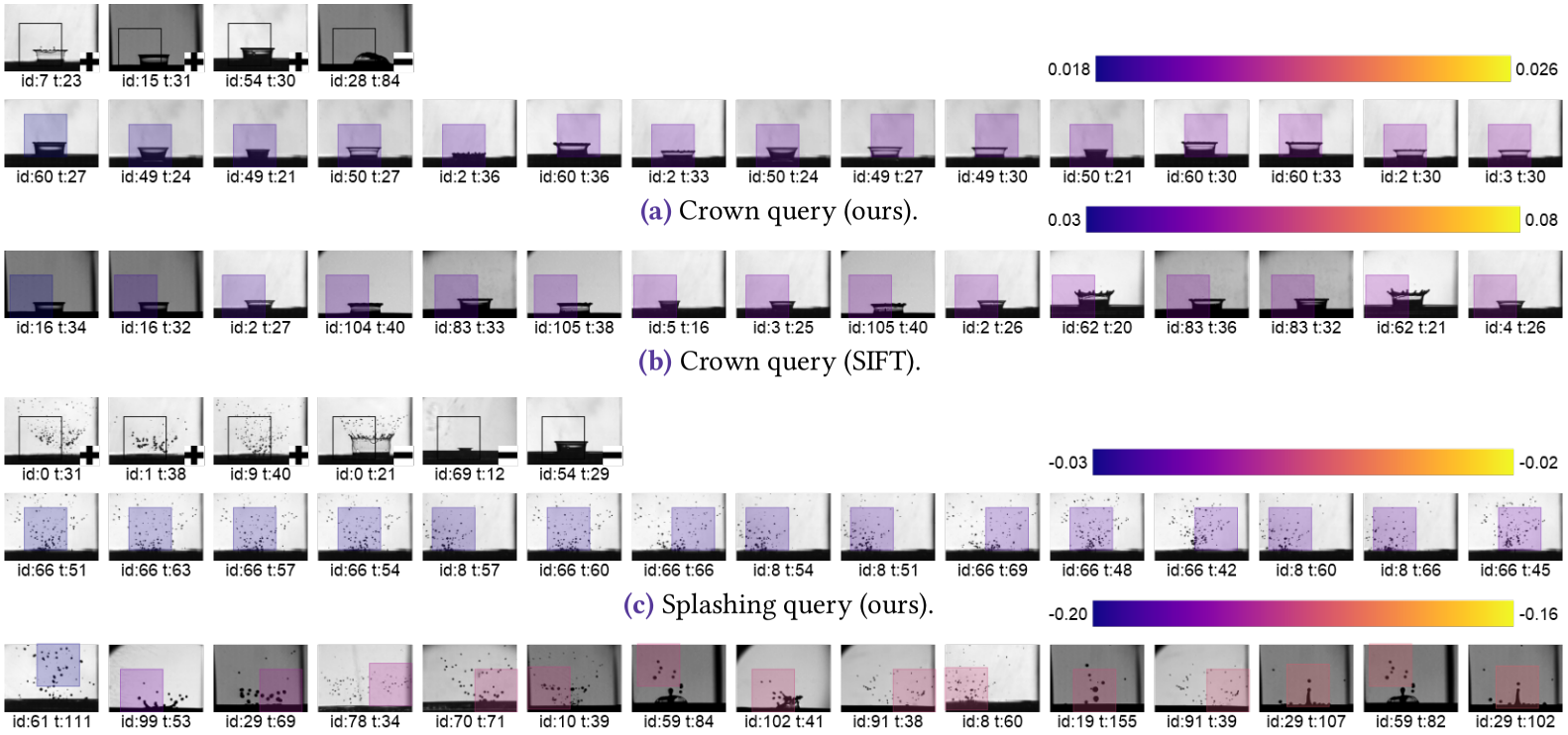


Figure 4.6 — Comparison of our similarity metric to SIFT on the droplet ensemble. **a, b:** we see that both methods produce good results on the fluid crown query, as it is very suitable for the SIFT descriptors. **c, d:** However, on the splashing query our method returns more robuts results, as SIFT is struggling to match small disperse droplets. An extended version of this figure can be found in the appendix.

patches contribute negatively. This way, we compute the SIFT-based ranking score from the query to every dataset patch and find the best matches.

The results for the "droplet splash" are presented in Figure 4.6. In Figure 4.6a and Figure 4.6b we show the results for the same crown query obtained using our model and the SIFT-based method. As we can see, SIFT produces a very accurate matching, which is not surprising: the dataset of camera images and the characteristic corners of the crowns provide an ideal application scenario for SIFT. Our model also successfully finds crowns in the data, though it also sometimes matches similar bubble deposition (this is discussed in more detail in Section 4.3.6).

Next, we perform a query for splashing (Figure 4.6c, Figure 4.6d), and here we see that SIFT performs much less robustly, also returning some crown deposition cases. The reason behind this is that a spray of small droplets does not yield robust SIFT keypoints, thus leading to worse performance. In contrast, our method has learned dataset-specific features and is thus able to return accurate matches.

Overall, we found that our method performs well compared to SIFT-based matching on this dataset of camera images and even outperforms it in some scenarios. Furthermore, we were unable to apply SIFT to our "cylinder" CFD ensemble (described in Section 4.3.6) because SIFT is not extracting any keypoints for most of the timesteps of this smooth dataset, while our method still performs robustly. This once again demonstrates the advantages of our generic similarity metric.

Comparison to Wang et al. 2016

In order to demonstrate that our approach can qualitatively match the results obtained with more traditional non-ML methods, we compare to the results from Wang et al. 2016. They introduced a method for finding similar regions in vector fields, which is used to return matches for a user-provided query region. First, they extract a pre-defined set of vector field features (called traits): divergence, the norm of Jacobian, etc. Then, SIFT-matches are computed in each field and used as match candidates. This serves to reduce the search space, as well as align the candidate regions with the query. Finally, they use the weighted L2 norm to evaluate the similarity of candidates to the query region. Notably, the method relies on a fixed set of features and can only be applied to vector fields. As discussed in Section 4.3.7, our approach prioritizes generality and thus only relies on a single raw scalar field. We also compare with applying SIFT to raw data in Section 4.3.7.

To compare to the results by Wang et al. we apply our method to the Isabel dataset, using the 3D+T version of our encoder (Section 4.3.5) and a 3D+T patch size $3 \times 15 \times 50 \times 50$, matching the proportions of the Isabel dataset. Unlike Wang et al., we are only using a single scalar field as input – the velocity magnitude. After training the model, we performed a query that aims to reproduce the result in their teaser image (Fig. 1 in Wang

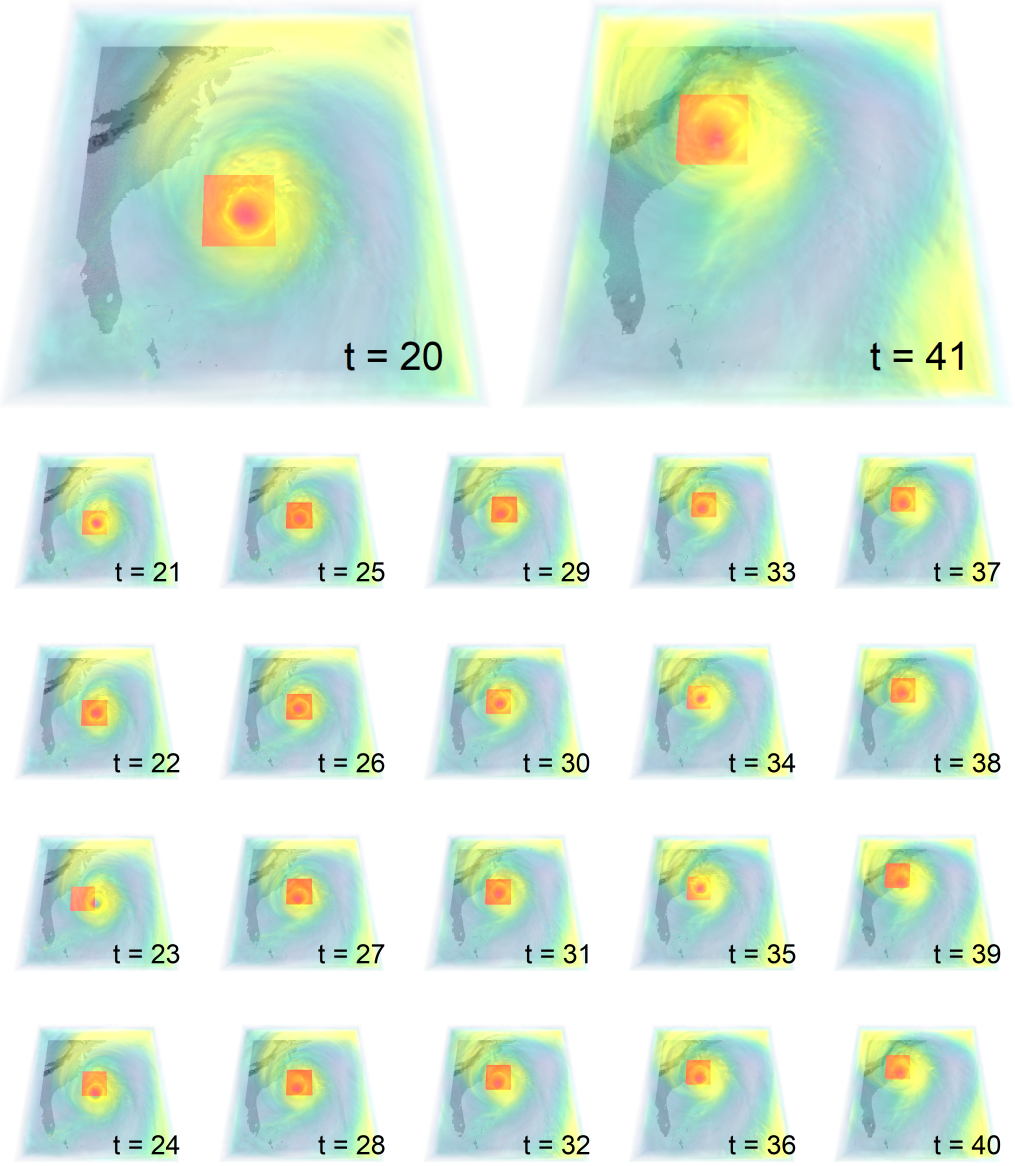


Figure 4.7 — Comparison of our method to Wang et al. 2016 on the example of the 3D Isabel dataset. The figure is analogous to Fig. 1 in their paper. We provided a single query patch of the hurricane eye in timestep 20 (left) and then render the best matching patch in the following timesteps. As we see, similarly to the method of ūthorg:wang:2016, we are able to track the eye of the hurricane.

et al. 2016). In the query, we provided a single patch near the eye of the hurricane at timestep 20, and in Figure 4.7 we visualize the best-matching patch in each of the following timesteps, rendering it as a transparent red box. Here we sampled candidate patches with stride equal to a quarter of the patch size to provide better spatial precision. As we can see, we are able to reproduce the result, effectively tracking the eye of the hurricane, while using only a single scalar field and not relying on any vector field features. This demonstrates the model's ability to learn relevant features during the self-supervised training.

4.3.8 Quantitative Evaluation

Next, we perform a quantitative evaluation of our method, aiming to assess its accuracy and compare it to alternative approaches. However, we do not have the ground truth describing what behavior occurs in different parts of the data or the similarity of different data patches. In fact, this is the very problem that we are addressing with our method. Thus, to enable the evaluation, we have manually labeled a small subset of the "cylinder" simulation ensemble (25 out of 300 members). This ensemble exhibits a range of different outcomes, however, there are two overall behavior types that we can approximately distinguish: turbulent and laminar flow. Importantly, this behavior occurs across the whole spatial domain, which means that we only need to label timesteps.

We assess accuracy using different search quality metrics (Section 4.3.8), both for our method and four alternative approaches (Section 4.3.8). Then in the following sections, we discuss results on manual as well as random queries, quantify model variance and its generalization performance.

Quality Metrics

Given a query, our method assigns a score to each patch in the data, effectively ranking the patches based on how well they match the query. Assuming that the positive patches in the query all contain some sought-after behavior A and all negative patches do not contain behavior A, we get the binary ground truth of patch relevance: patches with behavior A are relevant, and patches without behavior A are irrelevant. With this, we compute precision at three different cut-off ranks of 10, 50 and 100, e.g., Precision@50 is the percentage of relevant patches in the top 50 matches. In an interactive search scenario, it would be unreasonable to consider cut-off values above a few hundred for user investigation.

Aiming to also measure the result completeness, we designed a metric suitable to our problem. We compute timestep *coverage*, i.e., what percentage of timesteps exhibiting the queried behavior is included (covered) in the top N patches. And we determine N to be the minimum amount of patches needed to cover all the relevant timesteps.

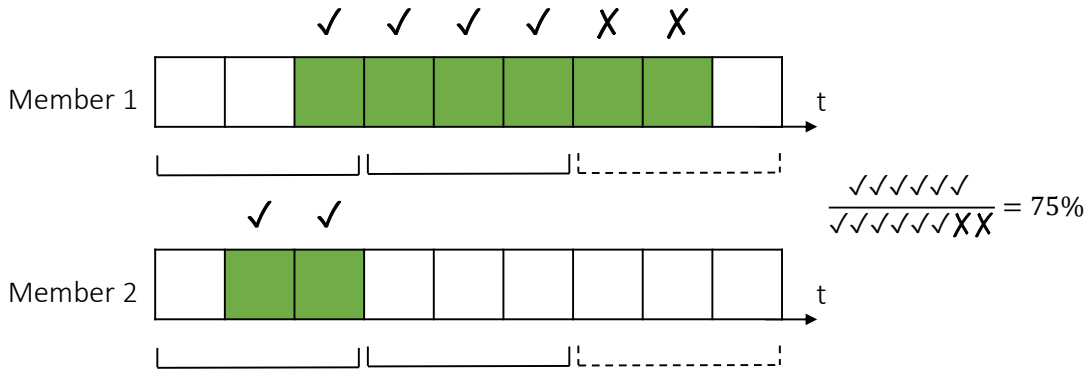


Figure 4.8 — Our coverage metric illustrated on a simple example. Two members of the ensemble contain the queried behavior (green) in some of its timesteps. To fully cover all queried regions with non-overlapping patches (brackets) we need four patches. Thus, we consider the top four matches according to our score (solid-line brackets). Due to a false negative in member 1 and a false positive in member 2, we miss two timesteps and get the total coverage of 75%.

See Figure 4.8 for an example. A ranking score that prioritizes patches with matching behavior will have higher coverage. The metric is less sensitive to the changes in the patch size, whereas precision may be more optimistic when the patch size is small and many candidates are available. Furthermore, the coverage metric considers how patches are positioned near the borders of feature regions: missing a patch that barely touches a relevant region has less penalty than missing a patch in the middle of the region.

Baseline Methods

We compare our model against many other similarity metrics: mean squared error (MSE), earth mover's distance (EMD), EMD applied to histograms (Hist-EMD) and a metric based on a pre-trained VGG16 model. In the appendix, we also show results for structural similarity (SSIM), L1 distance applied to histograms (Hist-L1) and fine-tuned VGG variants. To implement search using these metrics, we replace our model-based similarity metric d with the baseline metric when computing the ranking score from Equation 4.4. For the MSE metric, we compute the squared difference between the two spatiotemporal patches. We compute the EMD using the POT library (Flamary and Courty 2017) implementation of the Sinkorn algorithm (Cuturi 2013), computing the distance between the first timesteps, downsampled by the factor of four. Even with these simplifications, it takes around two days to perform the EMD computations for the evaluation on the "cylinder" ensemble. Hist-EMD is computed as the distance between the patch data

sorted into a histogram. VGG16 (Simonyan and Zisserman 2015) is a large computer vision model (100 million parameters vs. our 2 million) pre-trained on a big image classification dataset. VGG has been shown to generate generic features that are useful for estimating image similarity (Zhang et al. 2018). By comparing with it, we want to check if our encoder is learning useful problem-specific features or if it can be replaced with a powerful but generic computer vision model. The VGG metric is computed by putting the first frame of each patch through the VGG16 model and computing the distance in its learned feature space. Specifically, we take the output of the "fc1" layer and compute the Euclidean distance.

Query Results

We manually prepared two groups of queries containing typical laminar and turbulent patches. The quantitative metrics are presented in Table 4.2 and the rendered matches are shown in Figure 4.9. First of all, we see that overall the model-based search outperforms all the baselines, especially the non-ML methods. Specifically, the model shows both higher precision in the top results and higher coverage. The difference between the methods is generally less pronounced on the laminar queries because this scenario is more favorable for our baseline methods (especially MSE and EMD), since laminar members often have very small direct differences to each other. We also notice that the other learning-based method VGG achieves better coverage on the first three queries than our approach. This is because our model learns spatiotemporal aspects of the dataset and might, for example, consider slightly turbulent members similar to laminar, which is not accounted for by our binary labeling. Thus, in this particularly simple scenario of laminar behavior, it might perform slightly worse than a computer vision model. However, adding even a single negative patch to the query significantly improves the model's performance, again putting it above all the baselines. Another interesting observation is that the quality of the model's results generally improves as we expand the query. This is not always the case for other methods, e.g., MSE and EMD: especially for the turbulent queries, additional patches sometimes do not improve the result or even worsen the performance. Even a slight temporal shift or a difference in phase can result in very large MSE distances, worsening its results, while the model has learned these invariances during the training. For example, notice in Figure 4.9 that some of the matched patches have different phases (velocity peaks do not align). Furthermore, our model is more consistently benefiting from negative query patches, while other methods benefit from them only under certain circumstances.

Variance Quantification

Query sampling variance. We aim to evaluate the accuracy across a large number of possible search queries. For this, we defined 12 query types by specifying sought-

		Ours				VGG				MSE				EMD				Hist-EMD			
Feature	Patches	C	P10	P50	P100	C	P10	P50	P100	C	P10	P50	P100	C	P10	P50	P100	C	P10	P50	P100
turb	1+0-	**72.9**	**100.0**	84.0	**77.0**	65.0	**100.0**	**90.0**	72.0	42.4	90.0	50.0	45.0	38.5	**100.0**	62.0	39.0	56.5	70.0	56.0	54.0
turb	2+0-	**67.6**	**100.0**	88.0	**74.0**	64.1	**100.0**	**90.0**	73.0	39.4	90.0	50.0	42.0	38.5	90.0	60.0	41.0	60.0	90.0	48.0	54.0
turb	3+0-	**78.2**	**100.0**	**96.0**	**88.0**	66.8	**100.0**	94.0	77.0	36.8	70.0	44.0	41.0	35.0	80.0	42.0	36.0	59.1	20.0	52.0	56.0
turb	1+1-	**80.3**	**100.0**	**100.0**	**83.0**	69.4	**100.0**	94.0	75.0	53.8	90.0	56.0	54.0	52.1	50.0	64.0	54.0	54.7	60.0	56.0	56.0
turb	2+2-	**82.6**	**100.0**	**96.0**	**87.0**	69.4	**100.0**	92.0	76.0	52.1	90.0	56.0	52.0	47.6	50.0	64.0	53.0	57.4	50.0	54.0	56.0
turb	3+3-	**88.5**	**100.0**	**100.0**	**92.0**	76.5	**100.0**	**100.0**	81.0	52.1	70.0	58.0	53.0	48.5	50.0	64.0	53.0	56.5	60.0	54.0	56.0
lam	1+0-	63.7	**100.0**	**100.0**	**90.0**	**73.7**	**100.0**	94.0	89.0	64.2	**100.0**	76.0	77.0	65.1	**100.0**	94.0	86.0	60.9	90.0	54.0	49.0
lam	2+0-	66.2	**100.0**	**100.0**	**93.0**	**73.7**	**100.0**	**100.0**	90.0	64.2	**100.0**	**100.0**	87.0	64.2	**100.0**	**100.0**	92.0	60.9	90.0	54.0	49.0
lam	3+0-	64.6	**100.0**	**100.0**	**95.0**	**73.7**	**100.0**	94.0	90.0	65.1	**100.0**	98.0	86.0	65.1	**100.0**	94.0	91.0	62.8	90.0	56.0	59.0
lam	1+1-	**79.6**	**100.0**	**100.0**	**100.0**	71.7	**100.0**	90.0	88.0	62.3	60.0	72.0	68.0	61.7	50.0	58.0	71.0	63.6	90.0	52.0	50.0
lam	2+2-	**81.2**	**100.0**	**100.0**	**100.0**	76.4	**100.0**	98.0	91.0	64.0	70.0	86.0	75.0	63.1	70.0	66.0	70.0	63.1	90.0	58.0	49.0
lam	3+3-	**83.5**	**100.0**	**100.0**	**100.0**	81.9	**100.0**	98.0	93.0	68.3	**100.0**	90.0	77.0	64.9	60.0	72.0	73.0	79.8	**100.0**	92.0	94.0

Table 4.2 — Results of manually constructed queries on the cylinder flow ensemble. Each row depicts metric scores for one query. A query contains either patches with turbulent (turb) or laminar (lam) behavior. The 'Patches' column specifies how many positive and negative patches each query contains. Different queries are constructed by editing previous ones. For example, the turbulent query 'turb 3+3-' contains the same three positive patches as 'turb 3+0-', but we added three negative non-turbulent patches. We show four quality metrics: coverage (C) and precision (P) at three different ranks. We render query matches in Figure 4.9. Our model achieves significantly better results than the various baselines.

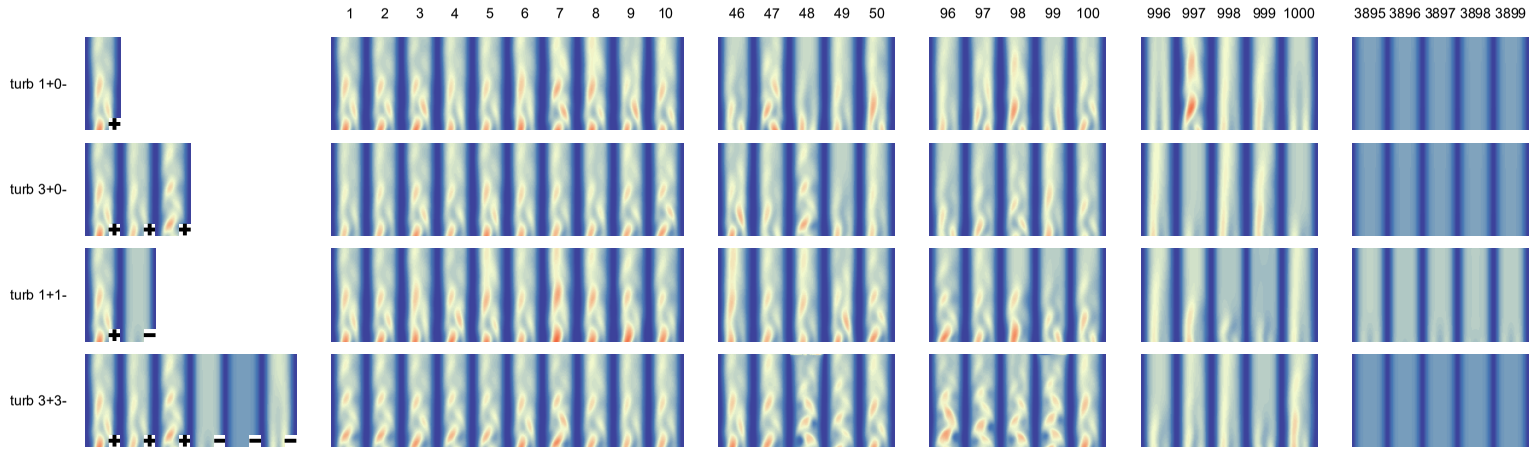
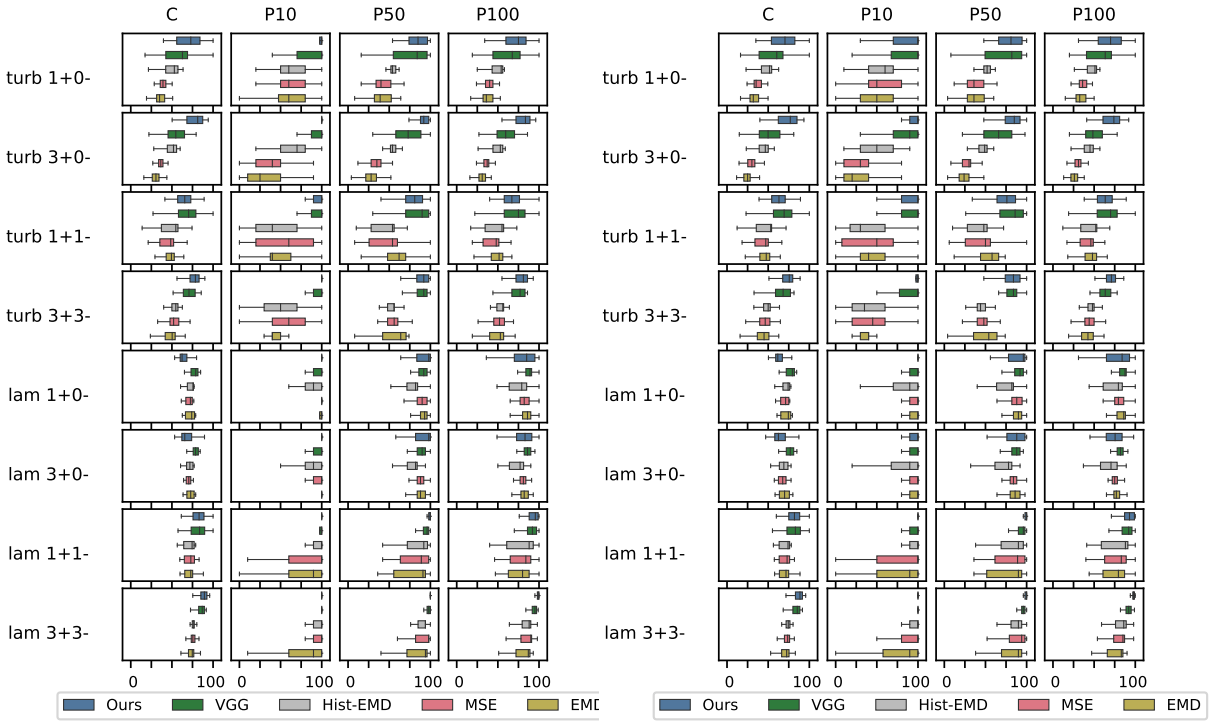


Figure 4.9 — Corresponding renderings to matches of turbulent queries from Table 4.2. Each row corresponds to a query. On the left are the query patches, and on the right, we show respective matches. We only render the first frame of each patch. We show the top 10 matches as well as some examples of lower matches. The icon on the query patches indicates whether a patch is a positive (+) or a negative (-) example.



(a) Including all results.

(b) Excluding query members.

Figure 4.10 — The distribution of query quality for "cylinder" wrt. randomly sampled queries. Each row represents a query type, e.g., 'turb 3+1-' means queries containing three random turbulent patches and one random non-turbulent patch. In the columns, we compute the same quality metrics as in Table 4.2, with all metrics ranging from 0 to 100. **a**: Our model shows better results than the baselines, especially on larger queries. We also see that the variance is reduced when more examples are used in the query. **b**: We evaluate the model's generalization by excluding patches from members mentioned in the query. Performance is slightly worse (as expected when removing the best matches), but the model is still successful. This suggests that the model generalizes beyond the pretext task and finds other instances of behavior.

after behavior (turbulent or laminar) and the number of patches in the query. Then for each type, we randomly generated 100 queries, utilizing the labels to ensure that each query contains appropriate patches. We present the results as a table of boxplots in Figure 4.10a. Most importantly, we again find that the model performs better than the baselines, and the effects described in Section 4.3.8 are still present when measured over a large sample of queries. However, we also see that non-ML results (MSE, EMD, Hist-EMD) sometimes have lower variance. This due to basic methods mostly finding nearby patches (due to temporal coherence), which gives consistent albeit poor results.

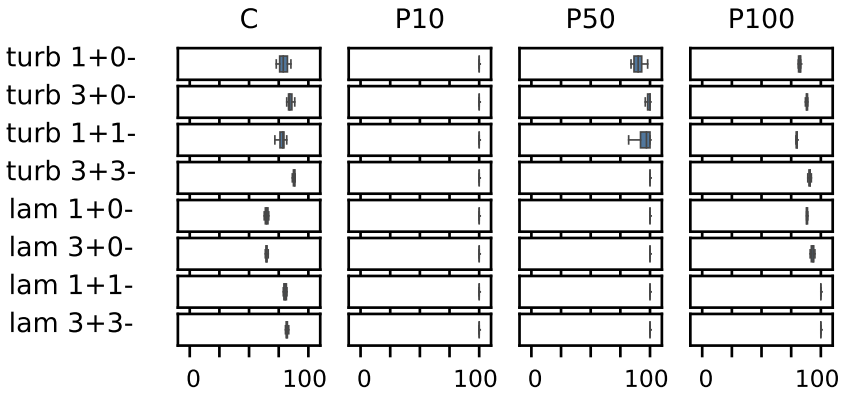


Figure 4.11 — The variance of query quality wrt. training process. Here we have trained our model ten times and performed the search with each one. We used the same manually constructed queries as in Table 4.2. While some variance is present, it is not significant and decreases with increasing query size.

Notice also that our results again tend to improve not only in accuracy, but also in terms of uncertainty as more patches are provided in the query.

Model training variance. One potential concern when using ML-models is the variance introduced during the training process, especially when transferring the model's representation to another task. To quantify the impact of model training, including training data sampling, weight initialization and data shuffling, we performed the training ten times and applied the resulting models to the twelve queries from Section 4.3.8. We observe that the model results indeed exhibit some variance, however, it is significantly below the query variance and stays consistently above the baselines (Figure 4.11). Additionally, we see that the results are the noisiest when using queries with few patches, but as more patches are added, the uncertainty gets considerably reduced. We believe that this effect is similar to ensemble averaging, as we effectively average several instantiations of our convolutional encoder.

Model Generalization

Another important aspect of a model's performance is its generalization properties across different tasks. Since we train the model on the pretext task, we want to make sure that we "get out" more than we "put in", i.e., that our model does not simply find patches from nearby locations. While finding similar patches from similar locations is useful, ideally the model should also generalize to other instances of similar behavior. To study this, we have used the same setup as in Section 4.3.8, sampling random queries and measuring search result metrics. However, we made a crucial modification: we remove from the list of candidate patches those patches that come from ensemble members mentioned in the query. This way, a model that simply solves the pretext task

would only find the members from the query and show poor results on the out-of-query data. The results are presented in Figure 4.10b.

Overall we can see that the performance is comparable to our previous experiments in Section 4.3.8, again demonstrating better results than the baselines. This suggests that the model indeed generalizes beyond finding patches from the same ensemble members. We observe some decrease in accuracy, but this is reasonable, since patches from members included in the query usually contain the most similar behavior and are expected to be among the top results. Thus, removing them from the pool of valid matches slightly lowers the quality metrics.

Parameter Study

Next, we perform a study of the patching parameters described in Section 4.3.3. In Figure 4.12, we present the results, where we vary the size of the spatiotemporal neighborhood (offsets o_s, o_t) and measure the resulting query accuracy. Despite significant changes to the parameter values, the model's performance remains consistent across several different queries, especially when considering larger queries.

Performance and ML Metrics

On the "droplet splash" dataset, the siamese model was trained with 45GB of data (500k points) on a desktop NVIDIA GTX 2070 graphics card in 4 hours and 20 minutes, achieving 94.7% training and 94.2% validation accuracy in 22 epochs. In our prototype system, the encodings of the candidate patches are precomputed (see Section 4.3.5), so the query performance is determined by how fast query patch data is encoded and distances to candidates are computed. A single-patch query in our prototype system takes 19ms to execute, where 3ms are spent encoding the patch on the GPU. And a ten-patch query takes 178ms to execute, where again only 3ms are spent encoding the patches (no impact due to parallelism), with distance computations taking up the most time. On the "cylinder", the model was trained in 4h 25min to 97.7% / 97.8% accuracy; a one-patch query takes 8ms, and a ten-patch query takes 41ms to execute. On the "Isabel", the model was trained in 3h 15min to 72.0% / 69.7% accuracy (if trained further, the model overfits on this smaller dataset); a one-patch query takes 5ms and a ten-patch query takes 31ms to execute. If we sample patches 64 times more densely (Figure 4.7), queries take 148ms and 1.3s, respectively. The above query timings are obtained while preloading the ensemble data into memory to remove the IO bottleneck of loading the patch data from disk. When this feature is disabled (or if the ensemble is too large), a naive implementation incurs an overhead of about 100-150 ms per ensemble member present in the query. Note that, in general, we did not heavily optimize our implementation for performance, and we believe there is potential for improvement. In particular, the performance of the prototype system could significantly be reduced with

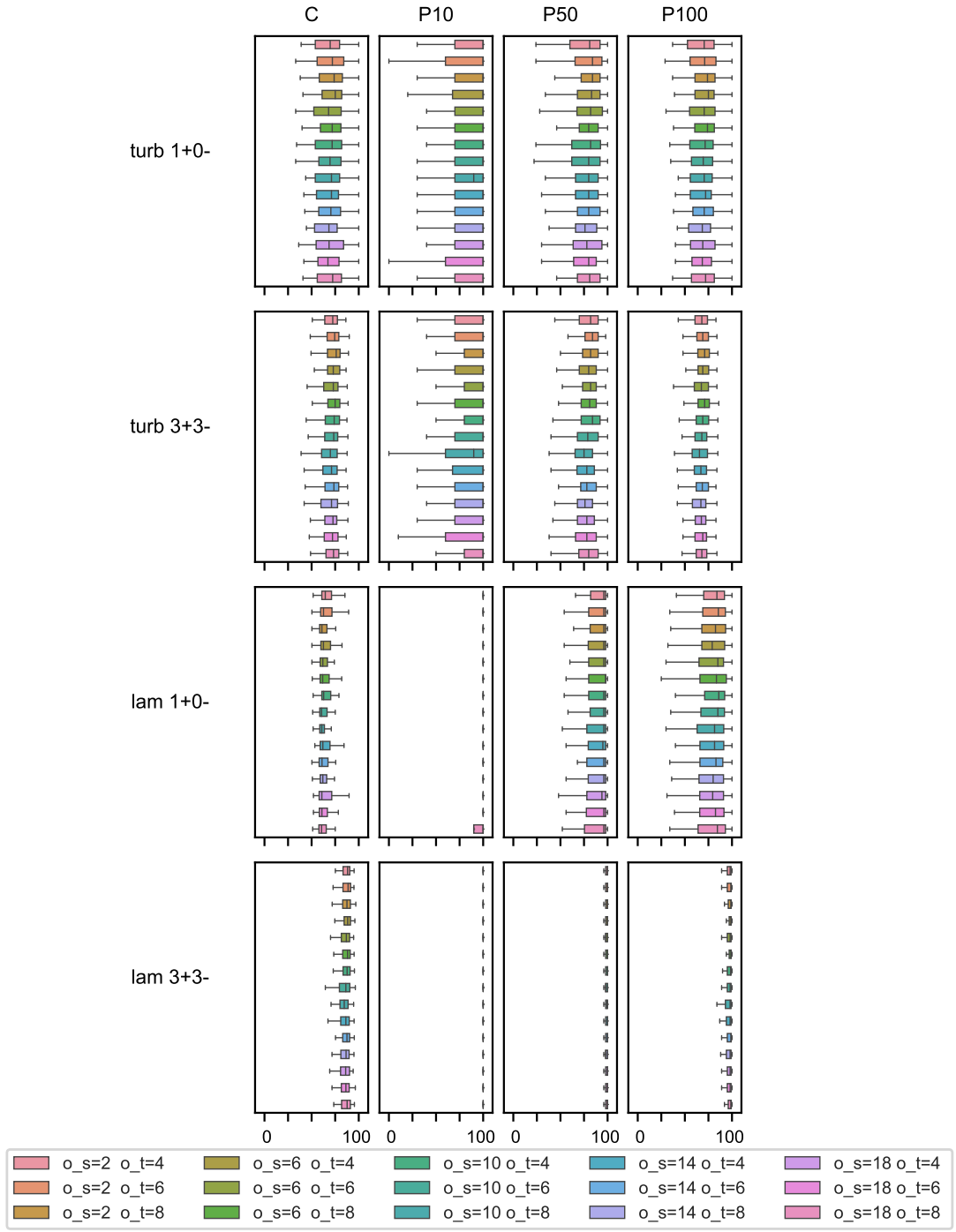


Figure 4.12 — Results of a parameter study on the cylinder ensemble. The experiment setup is similar to Figure 4.10 with metrics excluding the query members to better assess the generalization performance. Despite the changes in both the spatial and temporal offsets used during the model training, all models produce similar results. This indicates that our method is not overly sensitive to the parameters, which can be chosen with some minor knowledge of the dataset size.

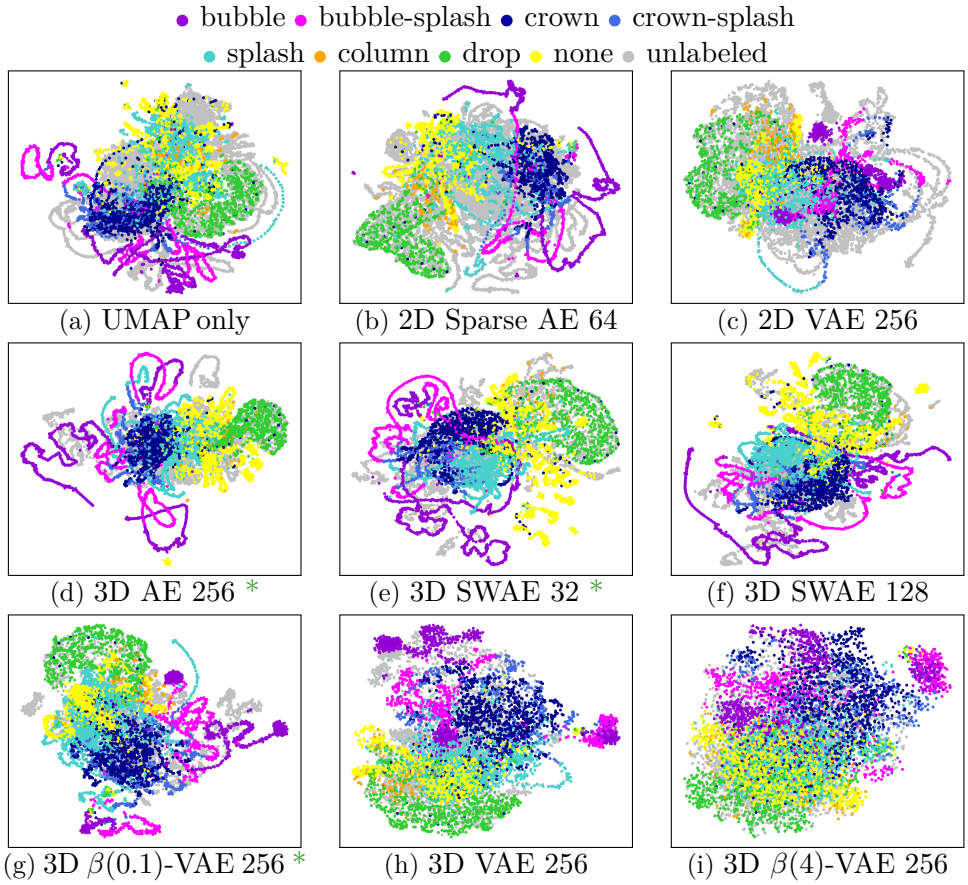
an optimized parallel implementation, a spatial data structure or using precomputed encodings for the query patches as well.

4.3.9 Discussion

The method presented in this section addresses an important challenge in visualization: data-driven analysis of large amounts of unlabeled scientific data. While here we focus on using our learned metric to perform search, our similarity metric can be useful for various visualization techniques. For example, clustering algorithms are generally based on quantifying distances, which our similarity metric could provide. Some algorithms, e.g., hierarchical clustering, also use the distance between a cluster and a point to compute clusters progressively. Here, our ranking score can be used to help form more meaningful clusters. Furthermore, a similarity metric can be utilized to perform projections, producing both a more meaningful representation of the overall ensemble and serving as a starting point for the search-based exploration presented in this chapter.

We demonstrated above that our model performs well on the search task, clearly outperforming other problem-agnostic approaches and even performing well compared to domains-specific methods (see Section 4.3.7). Nevertheless, there is room for improvement. While the model produces consistent results across different training runs, it has quite a high variance wrt. random queries (Section 4.3.8). There are several reasons for this. First, the problem itself is uncertain: the goal is to detect high-level behavior "types", which can have rather vague boundaries. For example, for some members of the cylinder dataset, it is difficult to say at what exact point the transition to turbulence occurs. Another reason is related: we are not certain that the random queries express the behavior that we are trying to find. Indeed, we see that some queries lead to better results than others, but we found that a common cause of poor results is an "unclear" query. The model's response might be reasonable, but it does not align with our expectations. Though, the user can improve the query in such cases by providing additional examples. Yet another aspect is that there is a disconnect between the evaluation and the target application. During the evaluation, we sample query patches independently, while in a practical scenario, the next patch included by the user is conditioned on the previous results. In other words, the user adjusts the query based on the intermediate results, e.g., to filter out a false positive. We do not model this effect in our evaluation metrics for simplicity, but it should be done in the future, iteratively constructing queries by including the incorrectly ranked patches into the query.

Overall, we see a lot of potential for self-supervised machine learning in scientific visualization, with many exciting directions for further research. We discuss these further in Chapter 7.



(a) UMAP only (b) 2D Sparse AE 64 (c) 2D VAE 256

(d) 3D AE 256 * (e) 3D SWAE 32 * (f) 3D SWAE 128

(g) 3D $\beta(0.1)$-VAE 256 * (h) 3D VAE 256 (i) 3D $\beta(4)$-VAE 256

Figure 4.13 — Projections of the latent space learned by different autoencoder variants. Behavior classes are color-coded. We see that varying the autoencoder architecture and parameters produces qualitatively different outcomes.

4.4 Autoencoders for Expressive Dimensionality Reduction

In the previous section, we described how one can construct a latent space for spatiotemporal data and then use it to measure similarity. We used a contrastive pretext task to learn this space (Section 2.2.6), but this is not the only way to approach (self-supervised) representation learning. Here, we will briefly present an approach where we learn a latent space with an autoencoder and use it to provide a visual overview of the ensemble data (Gadirov et al. 2021). This work is based on the Master thesis of Hamid Gadirov (Gadirov 2020), supervised by Thomas Ertl, Steffen Frey and me. In this project, I primarily advised on the machine learning aspects of the approach.

Autoencoders can be considered as a separate class of self-supervised learning ap-

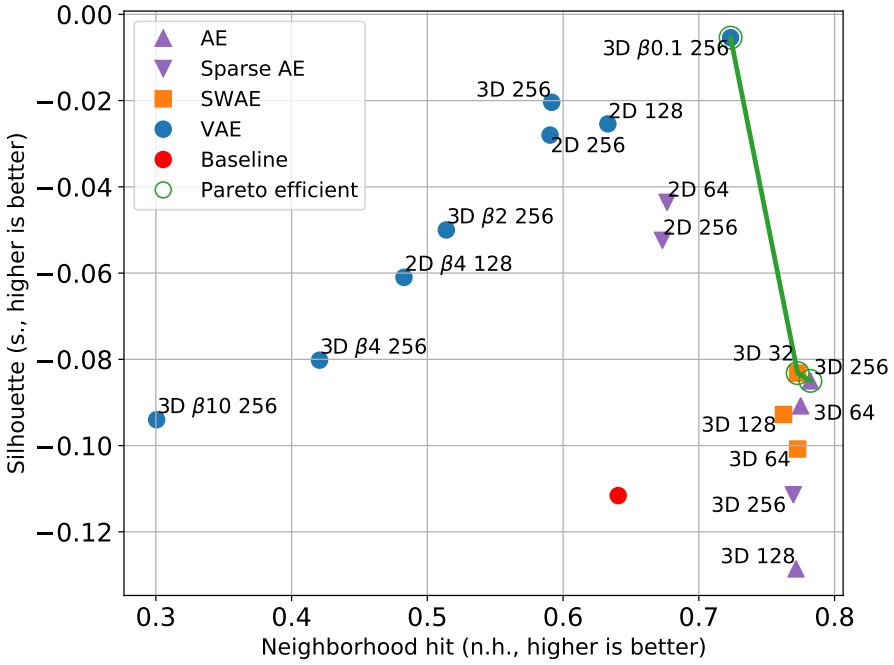


Figure 4.14 — Scatterplot of the quality metric values achieved by each tested autoencoder configuration. The Pareto-optimal configurations are circled (and marked with an asterisk in Figure 4.13). Again, we see that the choice of the model and its parameters has a significant impact on the projection. Moreover, no one model is superior across both metrics.

proaches, in addition to autoregressive (like our prediction-based method in Chapter 3) and contrastive (like the method from Section 4.3). Furthermore, many autoencoder variants were proposed in the years since their inception, with some demonstrating a superior ability to disentangle latent factors and smoothly interpolate between different input samples (Higgins et al. 2017; Kolouri et al. 2019). This property could be quite useful in visual analysis tasks, uncovering low-dimensional factors that could be used to construct an overview, separate different behavior classes and highlight outliers. The idea behind our project is to investigate which of these many autoencoder variants produce latent spaces that are most useful for visualization of spatiotemporal data.

We implemented several autoencoder architectures and trained them on spatiotemporal ensemble data, using a single timestep or a short sequence of timesteps as the input data. After training, we encode each timestep (or sequence) and use UMAP (McInnes et al. 2018) to project the latent vectors into 2D. To help with evaluation, we labeled a subset of the data into different domain-specific behavior classes. This way we can better understand the model's ability to separate the different classes.

The resulting projections are shown in Figure 4.13, along with the baseline of directly projecting the raw data. From the images we see that different architectures lead to noticeably different outcomes. For example, vanilla autoencoders and sparse autoencoders produce tight clusters but are worse at grouping together each behavior class. And (beta-)variational autoencoders provide an opposite trade-off.

We can better understand the differences by computing projection quality metrics, specifically, *silhouette*, which is higher for projections that tightly cluster each behavior class, and *neighborhood hit*, which is higher when the classes are separated from each other. In Figure 4.14, we see the value of these two metrics for all the tested autoencoder variants and parameter configurations. This view confirms what we see in the projections themselves – different models produce qualitatively different results. Worse, we cannot select the best model, as no single model is strictly better than the rest. Instead, we construct a Pareto frontier that consists of several models, each offering different trade-offs.

A similar situation is observed on another ensemble dataset, although a different set of model configurations become Pareto-optimal. This is likely due to the other dataset being a denser simulation dataset, while the droplet ensemble is fairly sparse and has a significant amount of empty space or unrelated features. On a positive side, we discovered that a small labeled subset is sufficient to determine an appropriate model and that this decision is consistent with larger labeled subsets. Therefore, we can use this approach to help explore large ensembles using only a small labeled subset of the data to choose the model.

In the context of this thesis, an interesting question is how do autoencoders compare to the contrastive approach from Section 4.3. Informally, it seems that the autoencoders are somewhat less reliable in the detection of similar behavior, likely due to being trained under a reconstruction loss. The model is forced to preserve the full information about the input (albeit efficiently represented), while a contrastive model can throw away irrelevant information such as the background, noise, etc. Still, in future work, we will perform a more thorough investigation, considering different contrastive and autoregressive pretext tasks. We discuss future work further in Section 7.5.

5

Metaphorical Visualization

This chapter is based on the following publication:

Gleb Tkachev, Rene Cutura, Michael Sedlmair, Steffen Frey, and Thomas Ertl (2022). "Metaphorical Visualization: Mapping Data to Familiar Concepts." In: *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI EA '22. Association for Computing Machinery, DOI: 10.1145/3491101.3516393

The mapping stage is the cornerstone of the visualization pipeline (Section 1.2). After all, mapping data to visuals is the essence of visualization. Not surprisingly, it is also the stage that (currently) benefits the least from ML applications. Designing the visual mapping requires significant human expertise and does not easily lend itself to automation. Nevertheless, this is an active area of research, in which the most prominent direction is to use ML models to recommend the appropriate chart type and axis assignments (e.g., Mutlu et al. 2016, Dibia and Demiralp 2019, Hu et al. 2018). We take a different route and use ML to extend the mapping beyond the assignment of visual attributes. The method is inspired by the similarity learning from Section 4.3, applying self-supervised models to connect vastly different domains.

In previous chapters, we adopted the traditional view of data visualization that presents the user as a trained specialist performing data analysis as part of their occupation. In this context, a visualization application is primarily evaluated by its efficiency, accuracy and scalability. However, there are many visualization "edge cases" that do not fit this description (Pousman et al. 2007). For example, ambient visualization might trade

accuracy and richness of encoded information for aesthetic qualities. And in science communication, the engagement of the audience can be one of the most essential factors (Borkiewicz et al. 2019; Ynnerman et al. 2018).

Similar scenarios are described under the umbrella of Personal Visualization (Huang et al. 2015), referring to both visualization of personal ego-centric data and data analysis in the personal context. Here, a user might have very different goals, background and expectations, when compared to a professional. Over the years, there were a number of applications that explored these research directions and were used by a much wider public for their personal goals and data. For example, Wordle was used to create more than 600,000 word clouds (Viegas et al. 2009). LastHistory allowed thousands of users to visualize their listening history (Baur et al. 2010). These and similar PersonalVis applications create an additional set of challenges that may rival or even eclipse the more classical evaluation criteria. Personal visualization should be accessible and address the personal goals of the user. But above all, in our opinion, it should be engaging and fun, as it encourages people to experiment with visualization and stay long enough to appreciate its rewards and more "serious" methods.

In this chapter, we invite the reader to consider data visualization through the lens of metaphors, as we believe this perspective can help us create more accessible and engaging visualizations. Cognitive linguists argue that metaphors are not only a poetic device, but are central to our language and cognition (Lakoff 1980). Every day we implicitly use *conceptual metaphors* like "time is a moving object" (time "flies", the time "will come", etc.) or "theories are buildings" (they are constructed, have a foundation, etc.) to help us structure complex ideas. Considering how pervasive metaphors are in our thinking, it is not surprising to find us using metaphors to understand our data. Visualization relies on many similar conceptual metaphors (e.g., "green is good", "up is more"), but is also itself a form of metaphor that helps us interpret abstract data entities in terms of visual experiences (shape, position, color, etc.).

What we propose is to extend the visualization metaphors beyond the visual. We can devise new ways of representing data by mapping it to concepts that are tacitly understood by humans. Of course, constructing mappings that are as abstract as "life is a journey" is not yet possible for automatic methods. But, if we have data from two domains, one of which is familiar to the user, we can map the entities (data points) of the unknown domain to the entities of the known one. For example, by preserving pairwise similarity, we can map data to words and learn that points X, Y and Z relate to each other like "dog", "house" and "chimney". Although the interpretation of relationships between words is somewhat ambiguous, we can leverage our knowledge of word similarity to explore the data, similarly to how we use metaphors in language to explain new ideas in familiar terms.

Conveying information in the form of metaphors is innately familiar to humans and

requires less expertise from the user, especially compared to the "expert" alternatives, such as glyphs, parallel coordinate plots and dimensionality reduction algorithms. Additionally, we can tailor the metaphor to the application context: if we are creating an infographic for an astronomy magazine, we could map our data to stars in the night sky, but we might use popular movies for the general public. Most importantly, it becomes possible to generate fun and vivid associations and provide an engaging way of communicating data that is suitable for more casual applications.

We call this approach *Metaphorical Visualization*. While we originally conceived of metaphorical mappings as a specific method of mapping between data and image embeddings, we quickly realized that the idea leads to many exciting applications. In what follows, we will present several diverse use cases, discuss their strengths and weaknesses, and outline how they fit together under the umbrella of metaphors.

5.1 Related Work

Metaphors for interaction and visualization. Metaphors have a long history in HCI, appearing as early as the first personal computers, where they were required for describing novel objects and interactions (Richards et al. 2009; Hartson and Pyla 2019). Finding effective interaction metaphors is also an important challenge for VR/AR applications (Mendes et al. 2019; Jerald et al. 2017). For example, designing an input mapping for 3D object manipulation often involves physical object metaphors: a balloon-on-a-string (Benko and Feiner 2007), corkscrew (Daiber et al. 2012), handlebar (Song et al. 2012), crank handle (Bossavit et al. 2014), and many others. Metaphors are also prominent in visualization where they are used for interaction, but also to construct visual representations. For instance, Havre et al. 2002 proposed a river metaphor to represent themes in document collections. And a city metaphor can be used to represent software architecture (Jeffery 2019). Overall, choosing an appropriate metaphor can have a noticeable impact on user performance (Ziemkiewicz and Kosara 2008). In this work, we extend the usage of metaphors for visualization from conceptual metaphors that structure the representation to computing mappings between concrete entities. So, for example, where ThemeRiver (Havre et al. 2002) would present topics as lines resembling rivers, we would map the topics to real rivers, using their properties to represent the data.

Metaphors can find applications in communicating information to wider audiences, where connecting to people and building their interest can be more important than conveying the raw facts. For example, in cinematic SciVis (Borkiewicz et al. 2020), metaphors are vital in conveying the subject to the audience. Metaphors also play an important role in creating infographics, where making the visualization memorable (Borkin et al. 2013) and aesthetically pleasing (Harrison et al. 2015) are important

considerations.

Image and style embeddings. Several of our implementations rely on image embeddings, which are most often constructed in the context of generative models and self-supervised pre-training for computer vision tasks. For example, Dosovitskiy et al. applied random transformation to learn a robust image embedding space, and Doersch et al. predicted positions of image patches to learn image features (see Doersch and Zisserman 2017 and Jing et al. 2020 for an overview). There have been a few works aiming to construct style embeddings or learn style similarity. Lun et al. used geometric similarities and supervised data to construct a model of style similarity, and Bell and Bala trained a siamese network to construct a style embedding to search for products with similar design. In this chapter, we use the SimCLR pipeline (Chen et al. 2020) and ideas from neural style transfer to construct our self-supervised image embeddings, but we focus on using the embeddings to explore relationships in other data.

Aligning word embeddings. One of our approaches to constructing metaphors is based on mapping between different embedding spaces. A related idea is utilized in natural language processing to facilitate machine translation. Given two word embeddings of different languages, a transformation (often linear) can be constructed to map the words of one language onto another. This can be done by using supervised word pairs (Mikolov et al. 2013a), finding similar strings in both languages (Smith et al. 2016), or more recently, in an unsupervised fashion by aligning the two distributions (Lample et al. 2018; Artetxe et al. 2018), see Ruder et al. 2019 for a comprehensive survey. We also aim to transfer knowledge by exploiting similarities between two domains, but unlike languages, our domains can share little similarity, making the linear (or any simple) mapping insufficient. Furthermore, there is no "ground truth metaphor" and many possible varied solutions could be valid, especially when the source domain is smaller. Most importantly, we present a conceptual approach to visualization through metaphors, where mapping between embeddings is just one of the many possible implementations.

5.2 Metaphorical Visualization

The main idea of our approach is to leverage the user's knowledge of one domain to learn more about another. Practically, this means that there is a dataset of interest and some data about another domain that is familiar to the user. We call the former *the data space* and the latter *the concept space*, where both are a discrete set of entities (data points or concepts). Our goal is to find a mapping of data onto concepts such that the relationships in the data space are preserved in the concept space. For example, we can express similarities between researchers (data) by mapping them to English nouns (concepts), such that related researchers are assigned to related words. This

mapping allows users to apply their knowledge of words to learn more about the researchers.

The key consideration in creating a metaphor is defining the relationships that should be preserved by the mapping. Depending on the structure of the data and the application, there could be several alternatives. We distinguish between distance-based, attribute-based, topology-based and hybrid methods of constructing the metaphor.

Distance-based mapping. In this type of mapping, we have a distance function in both spaces, which quantifies the pairwise similarity of points. We then compute a discrete assignment that aims to preserve distances, i.e., the distance between points in the data space should be as close as possible to the distances between their assigned concepts. With this mapping, the users can explore the pairwise similarities in the data by comparing the concepts, but also form groups of related data-concept pairs. The biggest advantage of this approach is its flexibility: we can use almost anything as the concept space, as long as it has a distance function. The richer the relationships captured by the distances, the more nuanced of a metaphor we can construct. We find this method to be most useful when we can apply machine learning models to construct distance functions for both spaces, allowing us to use complex entities, such as people, words, images, etc. In Section 5.3, we show how we can construct a distance function for researchers and then represent them with English nouns or with cat images, mapping between different ML embedding spaces.

Attribute-based mapping. This type of metaphor is applicable for tabular data with directly interpretable attributes. We can specify which data and concept attributes should have similar values. Unlike the distance-based mapping, we no longer define similarity among data/concept points, but instead specify similarity of data to concepts, i.e. across spaces. Therefore, the attribute-based method should be used when we have distinct features that can be conceptually related to each other. This gives us direct control over the metaphor and makes it easier to interpret, although it can lose some subtleties of a distance-based mapping. In Section 5.4, we use this approach to map between books, movies and games, such that their rating and popularity have analogous values, finding the "Twilight" among the games and the "Shawshank Redemption" of books.

Topological mapping can be pursued when the exact distances are not important or not available. For example, when constructing metaphors for network data or hierarchies, the relations in the data are modeled as a graph. Here, we need to build a mapping that preserves topology, e.g., adjacency for generic graphs or descendancy for trees. Interestingly, dimensionality reduction techniques like t-SNE or UMAP can also be considered a topological metaphor since they only preserve the local neighborhood and not the exact distances. We discuss a topological mapping for hierarchies in Section 5.6, where we map a taxonomy of sciences onto a taxonomy of industries while preserving

the parent-descendant relations.

The three conceptual approaches above are distinct from each other but are not mutually exclusive. They can be combined to produce *hybrid* mappings. This might be desirable, for example, when we want to control an aspect of a distance-based metaphor by explicitly aligning some of the data and concept attributes. Or, the concept space of an attribute-based metaphor might have an innate spatial layout that should be considered with an additional distance-based constraint. In Section 5.5, we use both distances and attributes to produce the mapping of movies to stars in the night sky. There, we match the movie rating to stars' brightness (attribute mapping), while also ensuring that similar movies are represented by nearby stars (distance mapping).

Overall, there are several approaches to defining and computing a metaphorical mapping, but they all share the overarching idea of representing data from one domain using another. In what follows, we present many concrete applications of metaphorical visualization and use them to demonstrate its advantages, discuss its limitations and outline important design considerations.

5.3 Distance-based Mapping

This section describes how to construct a metaphorical mapping by preserving distances in the data and the concept spaces. Here, we chose to focus on machine-learning embeddings for our data and concept spaces, but anything that has a distance function can be used for this approach. Throughout this section, we use the publication records of CHI, VIS and SIGGRAPH authors as our data. Initially, we constructed all results using their names and bio photos, but to avoid revealing personal information, we refer to each author using a randomly generated name and portrait.

5.3.1 Method

Given a set of data points X and a set of concept points C, our goal is to find a distance-preserving mapping from the data space onto the concept space. Because we are mapping to a discrete set of concept points (e.g., a set of images), we seek a discrete assignment, explicitly establishing correspondences between the data points and valid concept points. We can formulate this as an optimization problem, where we search for a map $M : X \to C$ between data points $x \in X$ and concepts $c \in C$ that minimizes the sum of squared differences between the distances d:

$$\min_{M} \; E(M) \text{ , with } E(M) = \sum_{i,j} \left(d(x_i, x_j) - d(M(x_i), M(x_j))\right)^2. \tag{5.1}$$

Depending on the application, M can be constrained to be injective (i.e., mapping to unique concepts). We specify the same distance function d for both spaces, but different

functions could be used if normalized appropriately. Note that Equation 5.1 is very similar to the Multidimensional Scaling objective (Mead 1992), with the key distinction that we perform a discrete assignment instead of mapping to a continuous space. This is a necessary complication because most interesting concept spaces are discrete (e.g., words), even if we construct continuous embeddings to obtain the distance function.

Unfortunately, this assignment problem is very challenging. Notice that the cost of each individual assignment depends on all the others: if we change the assignment of a single data point, it will change the distances in the concept space to all the other points, thus affecting their assignment costs as well. Because of this, the resulting problem is non-linear and difficult to solve optimally. Formally, it could be represented as a Quadratic Assignment Problem (Koopmans and Beckmann 1957) in Lawler's formulation (Lawler 1963), but the problem would be very large and intractable.

Instead, we use simulated annealing (Kirkpatrick et al. 1983) to compute an approximate solution. We initialize the algorithm with a random assignment of concept points. At every iteration, a candidate neighboring solution is generated by randomly changing the assignment of one of the points. The candidate's cost E' is computed (Equation 5.1), compared to the current cost E and with probability $p = P(E, E', T)$ we update the current solution to the candidate, where P is the acceptance probability function defined as:

$$P(E, E', T) = min\left(1, e^{-\frac{E'-E}{T}}\right). \tag{5.2}$$

Here T is the temperature parameter set to be initially high, encouraging early exploration, and then gradually decreased to converge to better local solutions. After a fixed number of iterations has elapsed, we obtain the final solution to our assignment problem.

5.3.2 Authors to Words

In this first example, we illustrate how VIS authors could be explored metaphorically by mapping them to English nouns. The data space is an embedding of authors, and the concept space is an embedding of English nouns, both learned from data. We train the author embedding using a self-supervised model similar to word2vec (Mikolov et al. 2013b) on the VisPubData dataset (Isenberg et al. 2017), which contains 3108 papers and 5415 unique authors The model is provided with a pair of authors and is tasked with predicting whether they are co-authors. Each author is passed to the model as an integer index used to look up a corresponding 32-dimensional embedding vector. Then, a dot product is computed between the vectors, followed by a single sigmoid output unit. The model is trained with a cross-entropy objective to perform the classification, learning an embedding in the process, and achieves 91% accuracy on a held-out validation set.

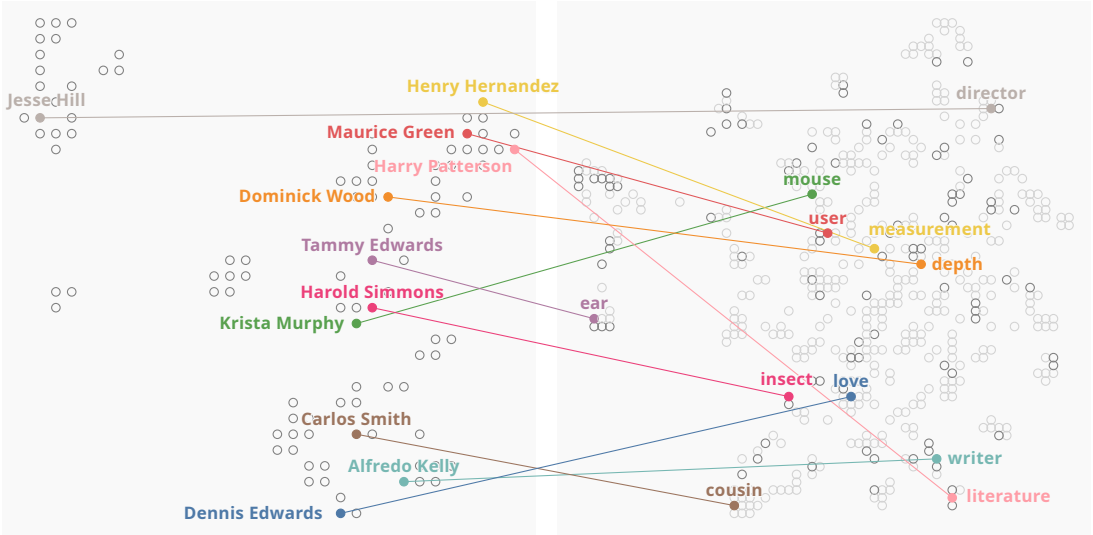


Figure 5.1 — Mapping VIS authors to English nouns. We show a UMAP projection of the author embedding (left) and the word embedding (right), and plot lines to visualize a few pairs from the resulting mapping. Some similarity relationships are present in the author projection (Maurice Green →'user', Henry Hernandez →'measurement'), but some are seen only in the concept projection (Harry Patterson →'literature', Alfredo Kelly →'writer'). And some of the more subtle similarities can only be noticed from the word themselves, e.g., Harry Patterson →'literature' and Dennis Edwards →'love'.

Our concept space consists of 500 common English nouns, which we passed to a pre-trained word-embedding model. We used the "en_core_web_md" model from the *spaCy* toolkit (Honnibal and Montani 2017) for the embedding, producing 300-dimensional embedding vectors. Both of the embedding models use the dot product, and accordingly, we also use the normalized dot product (cosine similarity) as our distance function d for both spaces.

Results. For our metaphor, we map the top 100 IEEE VIS authors to English nouns. We focus on the authors with the most entries in the dataset because they have the most co-authorship information and are likely to produce interesting relationships in the data. Again, we used randomly generated names to warrant the privacy of authors.

We visualize the mapping in Figure 5.1. We present each of the two spaces as a scatterplot of the UMAP-projected points (McInnes et al. 2018). Here, we also connect data points to their assigned concepts with lines. While showing the projections is not necessary for our approach, it helps us compare the positional and the metaphorical mappings. The common co-authors were assigned to strongly related words, e.g., Maurice Green → 'user' and Henry Hernandez → 'measurement' (cosine 0.61), similarly Krista Murphy → 'mouse' and Harold Simmons →'insect' (cos 0.64). Points that are not related are

appropriately mapped to unrelated words, for instance Krista Murphy → 'mouse', Jesse Hill → 'director' (cos -0.45). There are also pairs, e.g., Harry Patterson → 'literature' and Dennis Edwards → 'love', that share similarity (cos 0.53), but it is lost in the projections, revealed only through the metaphorical mapping.

Importantly, the mapping works not only for points highlighted in Figure 5.1 but for the whole dataset. If we now consider the 50 most frequent authors, we can "wander" through the space, following pairs of related authors (about 0.4-0.7 cosine), e.g.: Maurice Green → 'user', Adam Varma → 'engine', Harold Taylor → 'mixture', Kellie Jackson → 'salad', Harvey Hill → 'ratio', Ben Patterson → 'efficiency', Marilyn Huang → 'interaction' and back to Maurice Green → 'user'. A single word like 'mouse' (Krista Murphy), can express relationships to 'user' and 'device', but also to 'insect' and 'bird'. This flexibility of the word metaphor allows it to preserve some of the global relationships. In our study (Section 5.7), people reported that the word space requires time to interpret, but they were generally able to find thematic word clusters and sometimes the finer connections between the words.

This example is meant to introduce the idea of the metaphorical mapping, but also to illustrate some of its strengths and weaknesses. Of course, mapping data to words is not as accurate and reliable as traditional visualization. However, words can be concise and engaging, giving us the ability to describe a person's research interests with a single term. This can be advantageous when the main goal is not to convey facts as accurately as possible, but to engage the audience in casual data exploration. For example, imagine printing a single keyword on badges at a conference social event, providing a fun way of encouraging interaction and guiding people to others with shared interests.

5.3.3 Authors to Cats

In our second example of a distance-based mapping, we construct a space of CHI authors and use cat images as our concepts. For the data space, we obtained publication data from Microsoft Academic, loading 14k authors who have published at CHI and their 19k keywords. This author-keyword matrix underwent a sparse Singular Value Decomposition (SVD) to compute 30-dimensional author embedding vectors for the 100 most frequent authors according to our data. To construct a cat embedding, we took the cat images from the "Dogs-vs-Cats" dataset (Kaggle 2014) and trained a model using SimCLR (Chen et al. 2020), with ResNet18 (He et al. 2016) as the encoder architecture. In SimCLR, the model is trained in a self-supervised fashion to find identical images under random cropping, color distortion and blur transformations. The images are passed through the encoder that constructs a transformation-invariant representation, producing a 256-dimensional embedding vector. Then, cosine similarity is computed between the encoded images to predict which of the images were identical prior to the transformation. Deviating from the original SimCLR approach, we take the feature

Figure 5.2 — Mapping CHI authors to cat images. Groups of similar-looking cats were mapped to related authors, e.g., the black-and-white cats (1: Bruce Sanchez, Jesse Hill, Brian Lee, etc.) or the black cats (2: Peter Butler, Simon Zink, Jeremy Ramirez, etc.). See also Figure 5.3.

vector after the projection head because, in our application, we are interested in a space with meaningful distances rather than an information-rich representation for fine-tuning. After training the model, we used a sample of 1000 images and their feature vectors as our concept space.

Results. In Figure 5.2 we show the assigned cat images for some of the frequent authors in our data (names randomly generated). Although cat similarity can be more ambiguous to interpret, there are several interesting clusters. For example, there is a set of black-and-white cats (1): Bruce Sanchez, Jesse Hill, Brian Lee, etc. that contains related authors working in visualization and visual analytics (cos 0.62-0.83). Similarly, there is a large group of black cats that feature many similar authors sharing a connection through mobile and ubiquitous computing (2): Peter Butler, Simon Zink, Jeremy Ramirez, etc. (cos 0.57-0.90). The mapping also utilizes other features, producing a different but related cluster of black cats with a cage background (3). In this cluster we

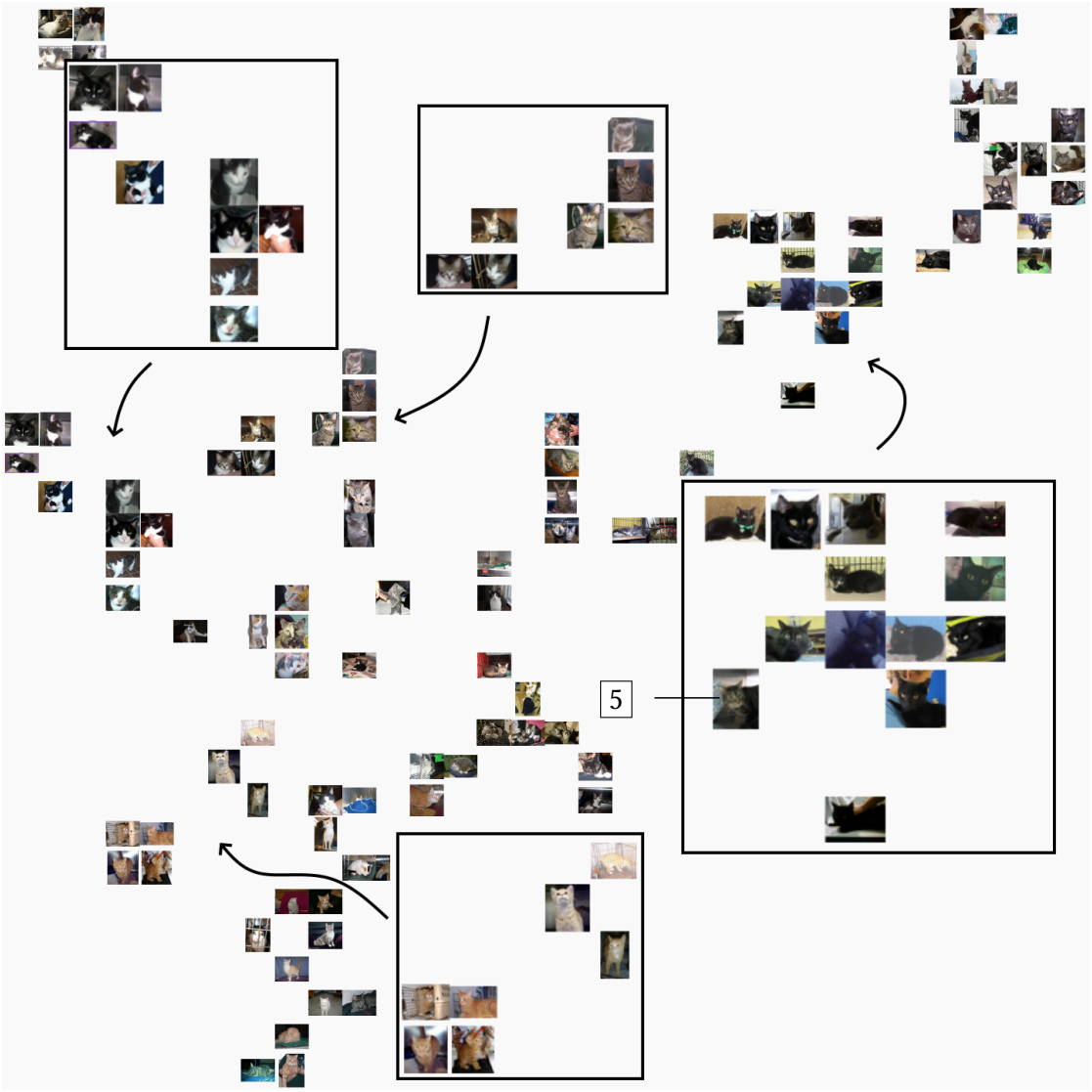


Figure 5.3 — Mapping CHI authors to cat images. A UMAP projection of the frequent author vectors, drawn as cats. We see many clusters of similar cats (cf. Figure 5.2), while some outliers (5) also hint at further global relationships.

find researchers whose topics commonly include user interfaces and multimedia: Ramon Brown, Sara Hall, Darrin White, etc. (cos 0.45-0.79). Strongly dissimilar to the above are the white and ginger cats. The latter (4) represent some of the authors working in psychology and sociology, e.g., Phillip Wright, Mary Barnes, Sam Baker (cos 0.63-0.69). We show the author projection in Figure 5.3, replacing the markers with the cat images. On this scatterplot, we can also confirm that strongly-related authors from the same

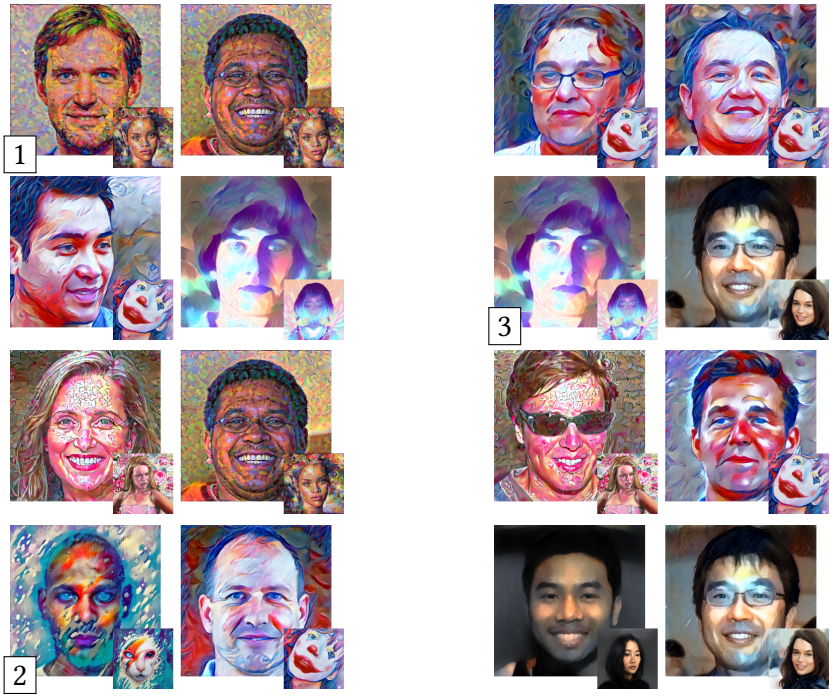
cluster are assigned similar cat images. And in a few cases, we can see an "outlier cat" that hints at the author's global relations, e.g., Peter Kelly (5) sharing similarity with the cluster of striped cats.

Overall, we found that our metaphorical mapping produces meaningful results for image embeddings. Our user study (Section 5.7) indicated that people are generally able to find similar cats, with the color being the most prominent feature. This example is meant to demonstrate that the idea of metaphors can be applied to many types of data, and we hope that it can spark other creative applications. In fact, in the next section, we continue to build upon this image mapping method to stylize author photographs to implicitly encode author similarity.

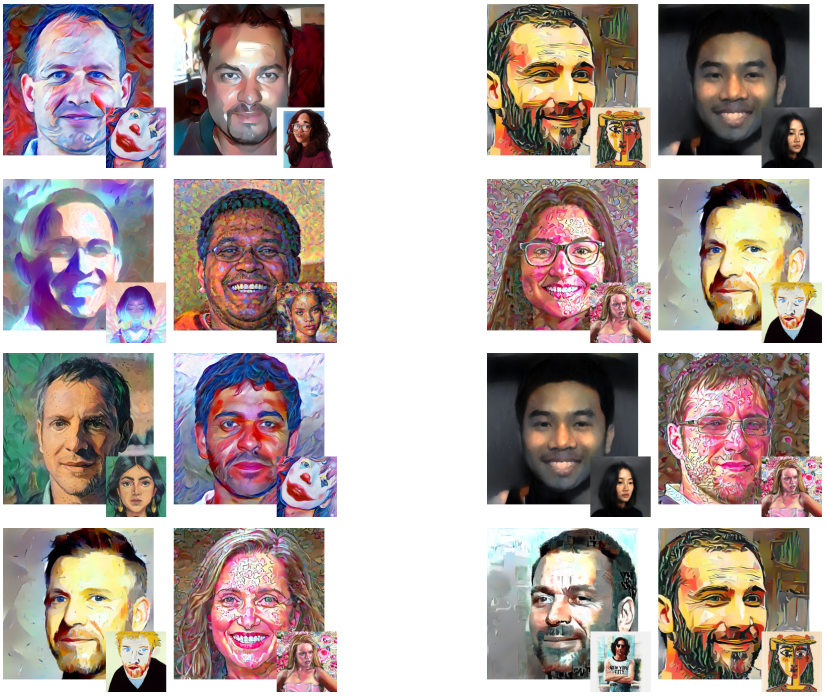
5.3.4 Authors to Visual styles

Continuing with the idea of image metaphors, we can not only assign a specific image to each author, but use just some of its properties to encode the metaphor. In this example, we will use neural style transfer to encode the similarity of SIGGRAPH authors into the artistic style of their portrait images. We use images generated by StyleGAN2 (Karras et al. 2020) to anonymize the authors images. The author embedding vectors are learned from a dataset of 1090 SIGGRAPH papers and 2008 authors, using the method from Section 5.3.2. Similarly to our cat metaphor, we perform a mapping between the author embedding vectors and an image embedding of style donor images. However, to construct an image distance metric that emphasizes the style of the image (rather than its content), we make several modifications to our model from Section 5.3.3. We are again using SimCLR, but replace the encoder with a pretrained VGG16 model (Simonyan and Zisserman 2015), which has its weights frozen during training. Instead of using the output of the encoder directly, we extract the style information as the activations after the convolutional layers ('conv11', 'conv21', …, 'conv51', see Simonyan and Zisserman 2015) and compute the Gram matrix for each layer's activation (we follow (Gatys et al. 2015) in how the style information is extracted). Concatenated Gram matrices are used as the input to the projection head. The idea is to constrain the encoder to only extract the stylistic features, and train the projection head to map them to a 256-dimensional vector that describes the style. We follow the SimCLR procedure as usual, but use an aggressive cropping setting (10-20% of the image size) to further encourage the encoding of the style and not of the content. The model is trained using a dataset of 11,000 digital art images (Srujan 2020).

Then, we construct a distance-based mapping between the author and style vectors, mapping 100 most frequent authors to a small sample of 16 style images. We deliberately use a small number of style images and allow duplicate assignments to make it easier to distinguish style similarity. Once the style images are assigned, we perform the style transfer for each author image with the method of Gatys et al. 2015.



(a) Similar pairs.



(b) Dissimilar pairs.

Figure 5.4 — Mapping SIGGRAPH authors to visual styles. We map each author to a style donor image, such that similar authors are mapped to similar (or even identical) styles. We then transfer the style onto the author's portrait (we use artificial images) to seamlessly encode their interests. In **(a)**, we show a sampling of related author pairs, where similarity can be encoded by using an identical style (1), styles with similar colors (2) or strokes (3). And in **(b)** we show pairs of unrelated authors, which were assigned significantly different styles.

Results. The styled author portraits are presented in Figure 5.4. In the top half, we show samples of similar author pairs (75th percentile and above). Authors with stronger similarity are assigned identical styles (1), making them particularly easy to distinguish. But other similar authors are also distinguishable through the similarities in the color scheme (2) or brushwork (3). And in the bottom half, we see that the most dissimilar authors (25th percentile and below) were mapped to significantly different visual styles.

Compared to the pure image metaphor from Section 5.3.3, mapping to visual styles allows more control over the final representation. Here we can fix the content of an image and use the metaphor to only alter its style, creating an implicit visualization of the metadata. Once again, the main strength of this approach is that it can be adapted to the application at hand and can provide a seamless way for users to engage with the data. For example, imagine generating stylized avatars for participants of an online conference. The users could be provided with a few options to tweak the result to their liking and then implicitly communicate their research topics to connect with the other participants.

5.4 Attribute-based Mapping

Now we demonstrate another method of constructing a metaphorical mapping. The idea of the attribute-based mapping is that when we have tabular data with directly interpretable attributes, we can explicitly define which concept attribute should represent which data attribute. For example, we could map movies onto stars, such that the star's apparent brightness represents the user rating of a movie. Compared to the distance-based mapping, this requires additional design choices to construct the mapping, but provides more control over the result and leads to a more transparent metaphor. Note, that we use the term *attribute* as opposed to a more common *feature* to avoid confusion with ML feature spaces from Section 5.3.

5.4.1 Method

We compute the attribute-based mapping by solving a Linear Assignment Problem (LAP). First, we take each data and concept attribute and normalize it to get zero mean and a standard deviation of one. Then, we define the cost of assigning a data point x_i to a concept point c_j as the MSE between the data and concept attribute vectors. More formally, the cost matrix C for the LAP is:

$$C = \left[c_{ij} = \left(\bar{x_i} - \bar{c_j}\right)^2\right] \in \mathbb{R}^{|X| \times |C|}. \tag{5.3}$$

Here $\bar{x_i}$ and $\bar{c_j}$ are the vectors of normalized data and concept attributes, constructed according to which data attribute should be mapped to which concept attribute. The

final mapping is obtained by solving the LAP $min_M \sum_i c_{i,M(i)}$, which can be done optimally and efficiently.

As our cost function, we opted to use MSE between the normalized attribute values, but many other options are available. The advantage of our choice is that we encourage relative scales to be preserved, so if a data point a is twice as far from the mean as b, this relationship will be maintained in the concept space. This works well in our examples, e.g., user ratings across various websites can be given in different ranges (1 to 5, 1 to 10 or 0 to 100) and have different voting patterns, so they should be normalized, but it is important to preserve their relative scales. An outlier in the data space should remain an outlier in the concept space.

5.4.2 Books to Movies and Games

We demonstrate our attribute-based mapping with a metaphor between popular books, movies and video games. All three domains are represented by tabular data with directly interpretable attributes such as user rating, release date, etc. Furthermore, we make the metaphor even more intuitive by mapping between similar attributes, e.g., matching the book's user rating to the movie's user rating.

We use a dataset of books from Goodreads (Dasgupta 2019), the movie dataset comes from IMDb (Leone 2019) and the game data is from Steam (Antonov 2019). For all three domains, we take the 500 entries with the most user ratings (i.e., the most popular). We construct two mappings, one from books to movies and the other from books to games. In the former, we assign the IMDb user rating to the Goodreads user rating and the number of ratings on IMDb to the same attribute from Goodreads. Similarly, we map the two book attributes to the user rating and the number of ratings on Steam.

The results for mapping books to movies and games one-to-one are presented in Figure 5.5, where we show the book-movie-game triplets as their cover images. In this example, we jointly explore these spaces via the 16 most popular books, ordered left-to-right, top-to-bottom. We see that the first match is "Twilight", "Batman v Superman" and "PUBG" (1), and it corresponds to items that are very popular but have an average, even controversial user rating. Note that by average, we mean average among the 500 most popular items. A similar case, albeit slightly less popular, can be seen in the "Lord of the Flies", "Star Wars: Episode I" and "Day Z" triplet (3). "HP and the Half-Blood Prince" is both very popular and very favorably rated, which results in it being appropriately matched to the IMDb's #1 rated "Shawshank Redemption" and the "Factorio" game (2). In general, we find that this metaphor is easier to interpret than the distance-based mapping and produces meaningful associations between the different media. Now one could explain to a person familiar with movies that "The Da Vinci Code" is the "Iron Man 2" of books.

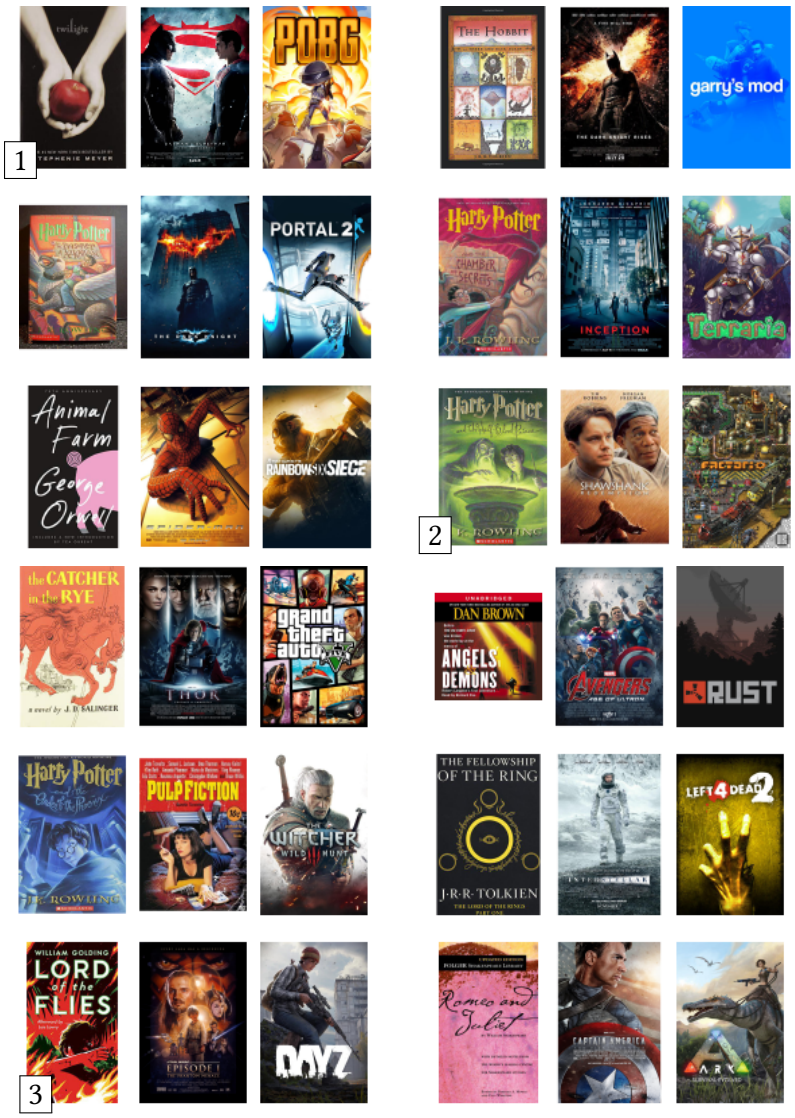


Figure 5.5 — Attribute-based mapping of books to films and video games. Here we explicitly map the books' rating and vote count attributes to similar film and game attributes. As a result, one can interpret the metaphor more directly since all three items in a book-film-game triplet will have similar rating and popularity. The user can rely on their sparse knowledge across all three domains to learn more about the unknown items. For example, top-left we see the "Twilight", "Batman v Superman", "PUBG" triplet (1), where all three are very popular and have a mediocre rating. While "HP and the Half-Blood Prince", "Shawshank Redemption" and "Factorio" (2) are connected because all three are popular, but are also rated very favorably.

Another interesting way of applying attribute-based mapping is to perform clustering first. The mapping is then performed between the clusters, producing both a multi-way assignment and generating distinct "categories" of analogous items across the three domains. For this metaphor, we apply k-means clustering with 64 clusters to each domain and then compute the mapping between the cluster centroids. We also add a third attribute, matching the book's page count to the movie duration and the average playtime of a game.

The results are presented in Figure 5.6. In each row, we display three clusters (one from each domain) that were mapped to each other. For each cluster, we show three of its elements as examples and plot where the cluster centroid (blue marks) is located in the overall distribution of each attribute (gray outline). In the first row (1), we see a cluster with works that are quite popular, positively rated and have above-average length. For books, we have "The Shining" and "Outlander" with 600+ pages, matched to the movie "The Shining" and popular games like "Rocket League" and "Arma 3". We observe that the mapping is preserving all three attributes well, resulting in a richer metaphor. Next (2), we have the opposite situation, with items that are not so popular, poorly rated (compared to the other 500 items) and are on the shorter side. Unsurprisingly, many of them are sequels. We also find a "short and sweet" cluster (3) of short and well-rated items, like "The Tale of Peter Rabbit", "Casablanca", "Monty Python and the Holy Grail", "Hellblade", and so on. Another interesting example is an outlier cluster of popular, very well rated and very long works (4): George R.R. Martin's "Clash of Kings", "The Lord of the Rings" films, "The Godfather", "Team Fortress 2" and "Warframe". All well-known, beloved and very long (or played a lot in the case of games). And the last cluster (5) has even longer works that have positive but not an outstanding rating. Here we see "War and Peace", "Titanic" and "Rust", all extremely long and with favorable user ratings, but not the highest possible. These last two clusters are outlier cases, resulting in the higher assignment cost and fewer elements since it is harder to find analogous items for the outliers. Nevertheless, we are still able to construct an appropriate metaphor, even when dealing with the extremes of all three distributions.

In summary, we see that despite the simplicity of the attribute-based mapping, we are able to generate accurate and engaging associations between different domains. Its particular strength lies in the control that we have over the metaphor and its resulting transparency. In the above examples, we have opted to use the most popular works to reach the widest audience. So when we say that a book or a film has a below-average rating, it is not awful, but only mediocre. Showing films with the IMDb rating of 5/10 would result in many unknown movies. However, the ultimate application of this metaphor could be, for example, in recommendation systems. With the knowledge of which works the user has read/watched/played, we can generate a personalized metaphor, also taking into the account the user's own ratings. So if you liked that one book that everyone else hated, we can use it as a metaphor to describe an unpopular

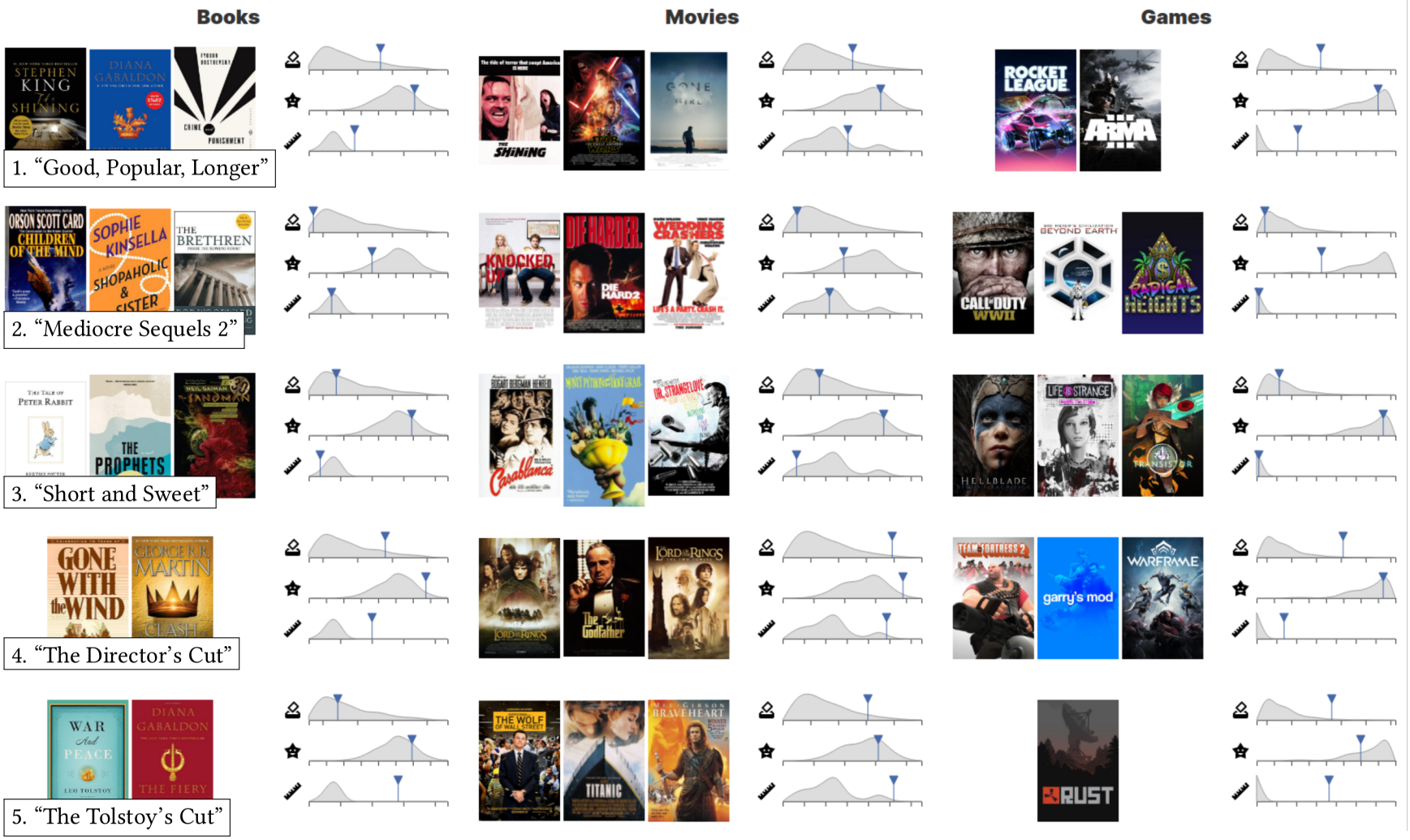


Figure 5.6 — Attribute-based mapping of book clusters to clusters of movies and games. The works with similar relative popularity, rating and length are matched together. Each row shows a triplet of matched clusters, and where each cluster falls in the overall distribution of popularity, rating and length. This metaphor successfully connects similar archetypes across all three domains. For example, we see a “short and sweet” category (3) with “The Tale of Peter Rabbit”, “Casablanca” and “Hellblade”; or the extremely long, but moderately rated match of “War and Peace”, “Titanic” and “Rust” (5).

movie that you personally might enjoy.

5.5 Hybrid Mapping

So far we have discussed two methods for computing metaphorical mappings: a distance-based approach (Section 5.3) and an attribute-based approach (Section 5.4). The former is very flexible, as it requires only distance functions and enables us to map across many different domains. The latter is more manual but provides additional control over the mapping, making it more interpretable. In this section, we will combine the two methods and preserve both the distances as well as the relative attribute values. One scenario where preserving both could be helpful is when we want to control a particular aspect of a distance-based metaphor, e.g., to assign frequently used words to the more frequent authors in our metaphor from Section 5.3.2. Another important use case arises when using attribute-based mapping with concepts that have inherent spatial information. Visualizing the spatial structure of the concept space will inevitably imply similarity among nearby data points, and so it is important to make sure that this similarity exists also in the data space. We describe such a scenario below, where we map movies to stars in the night sky.

5.5.1 Method

The hybrid metaphor mapping is a straightforward extension of the distance-based method from Section 5.3. Now the total cost of an assignment from Equation 5.1 also needs to include the linear attribute cost from Equation 5.3 and becomes:

$$E(M) = \sum_i \left[c_{x_i, M(x_i)} + \lambda \sum_j \left(d(x_i, x_j) - d(M(x_i), M(x_j)) \right)^2 \right]. \tag{5.4}$$

Here the first term is the attribute-based cost of mapping each data point x_i to a concept $M(x_i)$. And the second term is the distance-based cost from Equation 5.1 that captures the difference between the data and the concept distance for every pair of data point. The coefficient λ is used to control their relative importance. We solve this extended optimization problem using the same simulated annealing algorithm described in Section 5.3, replacing only the cost function.

5.5.2 Movies to Stars

We demonstrate the hybrid mapping by assigning popular movies to bright stars. With this metaphor, we generate an illustrated map of the night sky, inviting the users to explore the data and build connections between movies and stars.

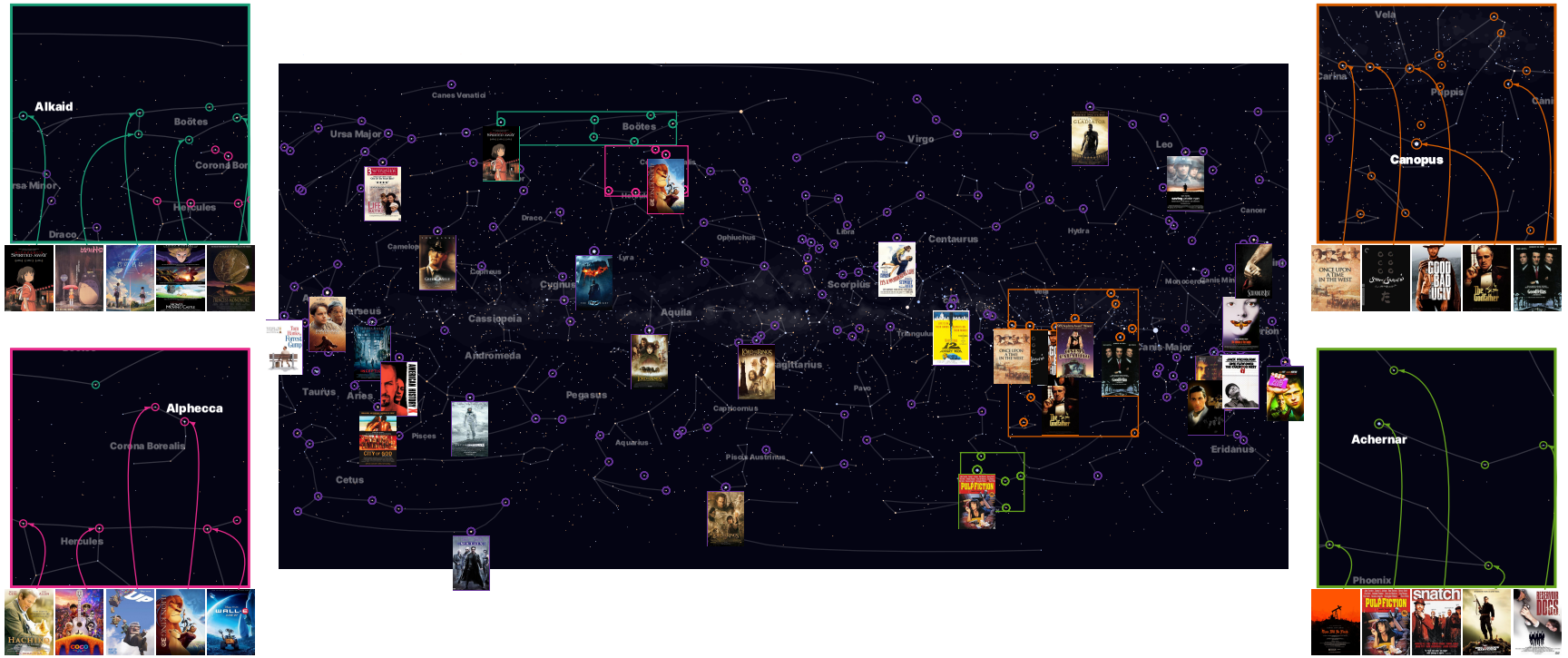


Figure 5.7 — The metaphorical mapping of popular movies to stars in the night sky. We assign well-rated movies to brighter stars, while also attributing related movies to neighboring stars. Users can explore this engaging infographic to build connections between stars and movies and to find out more about both in the process. For example, in the top-left corner, we see Miyazaki's animated films (in green, "My Neighbor Totoro", "Howl's Moving Castle", etc.) mapped to the Boötes constellation. Here the brightest star Alkaid was assigned to "Spirited Away", suggesting that it is the highest-rated of the films. Just below, the animated film classic "The Lion King" (in purple) became Alphecca – the jewel of the northern crown (Corona Borealis). And the exceptionally positively rated "The Godfather" became the second-brightest star in the sky – Canopus, surrounded by the mafia, western and samurai movies.

For our data space, we use a list of 200 movies from IMDb with the highest number of votes (popularity), cross-referenced with the MovieLens Tag Genome Dataset (Vig et al. 2012). With the tag data, we build a movie-tag matrix and perform SVD to construct a movie embedding. We then use the left singular vectors to represent the movies, and for the concept space, we use a list of 400 brightest stars in the night sky.

Next, we need to define the mapping, and here we must consider how it will be presented to the user. The stars have natural spatial positions that we can exploit to encode more information into the visualization. We accomplish this by using our hybrid mapping approach to add additional distance-based costs. For the stars, we compute the Euclidean distance in the display space, using the equirectangular projection and the galactic coordinate system. An area-preserving projection like Aitoff is also an option, but we prefer a rectangular projection for an infographic. And for the movies, we take their singular vectors, apply UMAP to first project them into a two-dimensional space and then compute the Euclidean distance. We apply this pre-projection step rather than compute the distances in the high-dimensional space because a neighbor-preserving projection works better when mapping to the low-dimensional space of 2D positions. As the last step, distances in both spaces are normalized to mean zero and standard deviation of one to match their scale. For the attribute cost, we map the average movie rating to the star brightness (apparent magnitude) so that highly-rated movies will correspond to the brightest stars. Finally, we compute the mapping with simulated annealing, using the cost function from Equation 5.4.

The results are presented in Figure 5.7. We render the stars and the constellations with D3-Celestial (Frohn 2017) and mark all the stars that were assigned a movie. The posters are shown only for the most popular films to avoid clutter. We also show magnified images for several clusters.

First, we observe that the movie similarity was properly encoded as the star distances. For example, we see a cluster of Disney/Pixar animated films (purple, “The Lion King”, “Up”) next to the Boötes constellation of Miyazaki animes (teal, “Spirited Away”, “My Neighbor Totoro”); a cluster of older western and mafia movies (orange, “Once Upon a Time in the West”, “The Godfather”); and a constellation of Tarantino and similar crime films (green, “Pulp Fiction”, “Snatch”). Inspecting different regions of the sky, we also see that the films with the highest rating are mapped to the brightest stars (relative to the other popular films). The highly-rated “The Godfather” is mapped to the brightest star of the region – Canopus, while the nearby “Goodfellas” is assigned to the dimmer Adhara.

Overall, we believe that the mapping captures the metaphor of the “movie night sky panorama” and generates some memorable connections, like Boötes being the anime constellation and Leo representing films about war. Informally, we found it much more engaging to explore both stars and movies under a joint metaphor than as separate

datasets. It shows that sometimes the metaphors could be even more fun when the user has some knowledge of both spaces, telling stories and making associations across them. This could be used, for example, for science communication or to connect to a particular audience. Furthermore, together with the Authors-to-Styles metaphor (Section 5.3.4), this application also demonstrates how the metaphors can influence the visualization itself, mapping to visual attributes (positions, in this case) as well as abstract data (ratings) to construct the final representation.

5.6 Topological Mapping

In this section, we present a prototype of topological mapping. We use it to map a taxonomy of sciences (taken from Wikidata) to a taxonomy of industries (taken from Eurostat), i.e., between two trees.

5.6.1 Method

We define a hierarchical dataset as a directed tree with edges oriented from parents to children, where each vertex of the tree is associated with a vector of attributes. For example, this could be a file system tree, with each vertex having a size and a creation date. Our goal is to map vertices of the data tree to vertices of the concept tree, such that the difference between their attribute vectors is minimal according to some cost function (e.g., MSE). This is the attribute cost from Equation 5.3. Additionally, the map M must satisfy the hierarchy constraint: if vertices x_p, x_d in the data tree are connected by a path $(x_p, \ldots, x_k, \ldots, x_d)$, then their assigned concepts must also be connected by some path $(M(x_p), ..., c_k, ..., M(x_d))$. In other words, the parent-descendant relationship must be preserved.

We solve this problem with the simulated annealing algorithm described in Section 5.3. However, generating random neighboring assignments is no longer trivial due to the hierarchy constraint, so we must make some modifications. The most important modification is that we do not require all of the data vertices to be assigned, so that we can compute mappings for data trees that do not fit topologically within the concept tree. Instead, we add a loss term that penalizes unassigned vertices, making it a soft constraint. This simplifies the search of the solution space since we can move through "partial" solutions to more easily find low-cost regions. And it also makes the algorithm much more practical to use with real datasets that do not align well with each other. We initialize the search with a random assignment computed in the following way. The data root is assigned to the concept root. Then, we traverse the data tree breadth-first, and assign each vertex x_c with parent x_p, to a random descendant of $M(x_p)$, satisfying the hierarchy constraint. If concept $M(x_p)$ has no unassigned descendants, we leave x_c unassigned. After initialization, we proceed as described in Section 5.3, stepping over

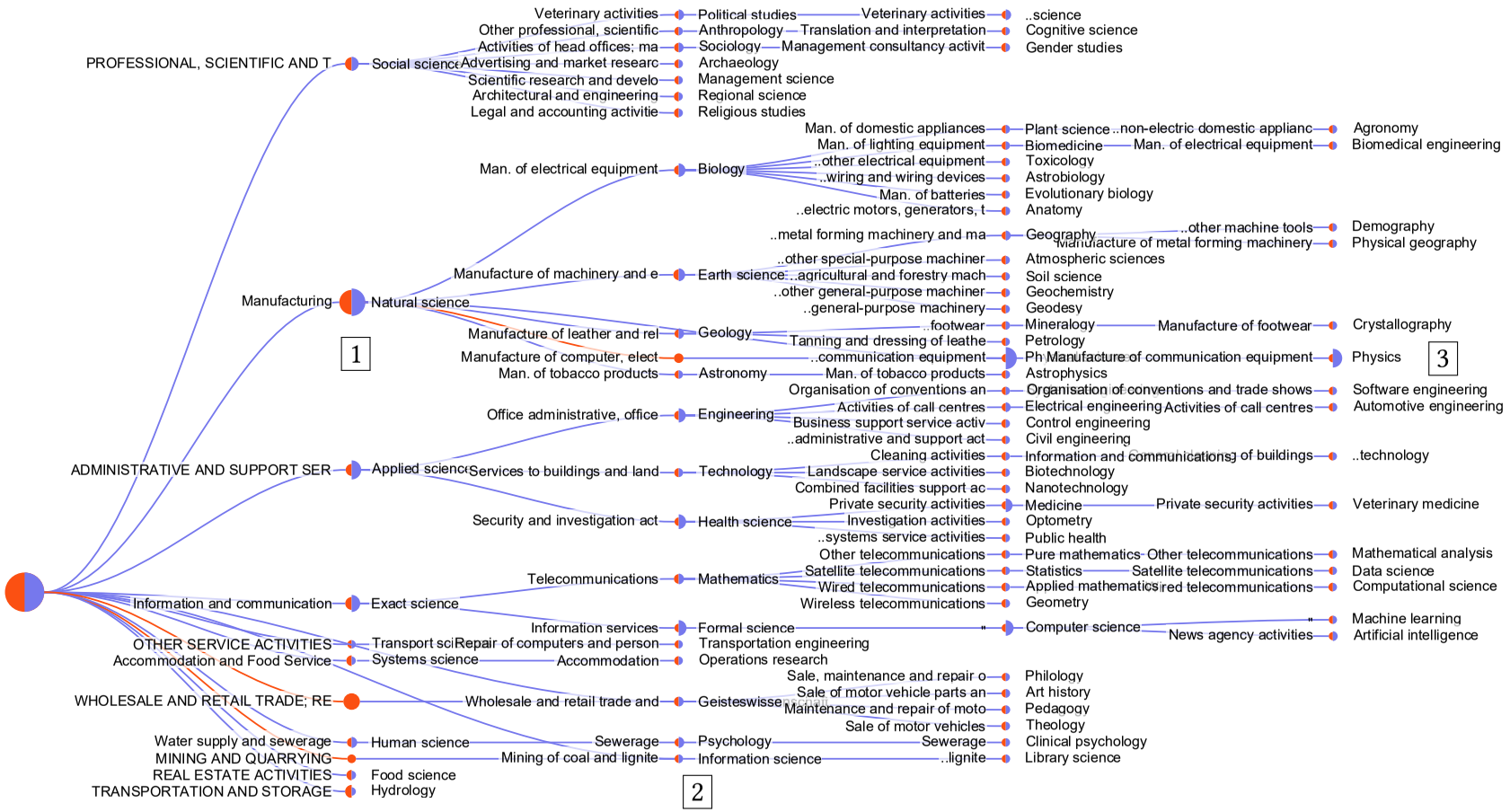


Figure 5.8 — Mapping the science taxonomy (blue) to industries (orange). The area of the (half-)circles encodes the size of each field or industry. The mapping preserves both the parent-descendant relationships and the size of the nodes. For example, natural science (1) is assigned to the largest industry – manufacturing, while biology and astronomy are mapped to different types of manufacturing. Fields are allowed to "skip" levels of hierarchy to better match their attributes, like in the case of information science (2). And for physics (3) there is no type of manufacturing that is sufficiently large to represent it.

random neighboring assignments. The sampling of the neighboring assignments also needs to be adapted. First, we sample a random data vertex x_c that has an assigned parent x_p. Then, we assign x_c to a random descendant c_d of $c_p = M(x_p)$, where c_p must be connected to c_d with a path of unassigned concept vertices. If there are no such vertices c_d, we simply unassign data vertex x_c. Finally, we unassign all descendants of x_c and c_d. These procedures guarantee that after changing the assignment of the data vertex x_c we still satisfy the hierarchy constraint.

5.6.2 Sciences to Industries

In Figure 5.8, we show the mapping of scientific fields onto industries. It matches the topology and one attribute: the number of publications to the number of employees (size of a subfield to a size of an industry). We render the resulting mapping as a joint tree, where sciences are colored blue and industries are orange. We use the area of the (half-)circles to represent the number of publications/employees in each field/industry but set a minimal value to prevent the nodes from getting too small. Overall, we are able to satisfy both constraints: the natural sciences are assigned to the largest economic sector – manufacturing (1), preserving the size attribute across the two spaces. And the specific natural sciences are mapped to the descendants of manufacturing, e.g., biology becomes electrical manufacturing. Similarly, mathematics and computer science are mapped to subtypes of the information and communication industry. We see that some scientific fields, e.g., information science (2), appropriately skip a hierarchy level to better match the attribute. Another interesting case is physics (3), which is a very large field and cannot be matched well to any subindustry of manufacturing, because there is no type of manufacturing that is so much larger than the others. Nevertheless, finding good solutions requires us to introduce multiple loss functions and constraints, leading to an inelegant algorithm. We present it here for the sake of completeness, and in the future, we will pursue a specialized algorithm for tree mapping, e.g., searching for assignments hierarchically and providing better initialization by aligning nodes with similar local topology.

5.7 Qualitative Study

To learn more about how people perceive and interact with metaphors, we conducted a user study. We opted for a qualitative study because it is better suited for our rather unconventional idea of metaphorical visualization and allows us to study aspects that we might not have anticipated.

5.7.1 Study Design and Analysis

We recruited 10 participants who are doctoral students working on visualization and HCI at a local department (3 female and 7 male, aged 27-36). Aiming to study metaphors in a personalized context, we purposefully collected data and built metaphors about the people at the department, which included the participants themselves. We obtained the data from Microsoft Academic, retrieving 13k authors that published in top visualization venues and 18k topics (keywords) attributed to them. Similarly to Section 5.3.3, we extracted embedding vectors for 50 authors at the department. The authors were then mapped to words (Section 5.3.2), cat images (Section 5.3.3) and visual styles (Section 5.3.4).

All three metaphors could be explored in a web-based tool that we created for the study (Figure 5.9). The tool displays all 50 authors as draggable notes on a digital corkboard so that the users can perform affinity diagramming. This setup provides the participants with a simple task that encourages them to explore the metaphor. It also allows them to better illustrate the perceived similarities and clusters. The users can also create new metaphors by dragging concepts from the left panel onto the authors, thereby providing initial assignments for some authors. The tool would then compute the concepts for the remaining authors, continuing the metaphor.

A session with each participant lasted around 45 minutes in an one-on-one video call, the screen and the audio were recorded. It began with a short introduction to metaphorical visualization. Then, the participants were shown their personal Microsoft Academic page and the dataset origin was explained. Next followed the main part of the study, where the participants used the web tool to interact with the metaphors. This usually began by asking the participant to find themselves and a few people with similar concepts. Afterward, they were instructed to search for any other similarities and arrange the notes while thinking aloud. Participants were occasionally prompted to comment on why they considered grouped concepts to be similar, or whether the groups aligned with what they know about their colleagues. After about 5-10 minutes, the participants were explained how they can change the metaphor and prompted to try it. They would then comment on their existing groupings and continue to explore and edit the metaphor for about 5 minutes. This process was repeated for all three concept spaces in a different order, and the session was concluded with a 5-10 minute semi-structured interview. During the interview, the participants were asked to comment on which spaces they found easier to interpret, and which they liked better. They were asked about any expected or unexpected groupings that stood out and any particular assignments that they remembered. Also, we asked if they had any ideas for other concept spaces or applications of metaphorical visualization. Finally, the demographic data was collected.

During the analysis phase, the recordings were transcribed and annotated to include

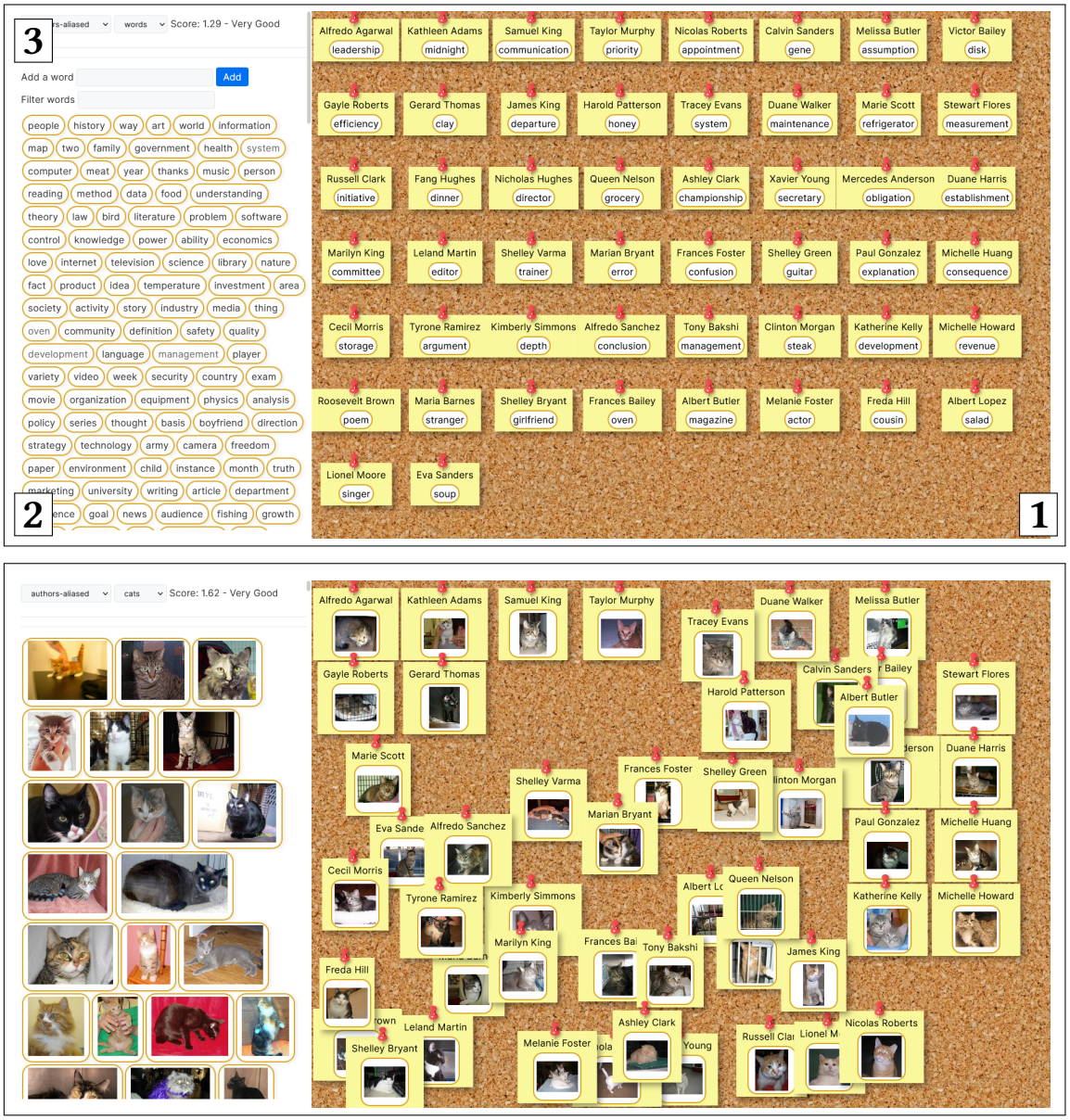


Figure 5.9 — The metaphor tool used in our study. All the names are anonymized. **Top:** The initial state of the tool displaying the word metaphor. **1:** The main area used for affinity diagramming. Each draggable note displays a person's name and the word that they are currently mapped to. **2:** A list of concepts that can be assigned to any person by dragging the concept onto a note. **3:** The controls for switching between metaphorical spaces. **Bottom:** The tool during the exploration of the authors-to-cats metaphor.

the groupings formed by the users in the tool. Then, the transcripts underwent an iterative coding process using NVivo, eventually generating 65 codes that were grouped into 10 categories.

5.7.2 Findings

Perception of similarity in different spaces. During the study we observed the participants explain how they reasoned about similarity in different spaces, and during the interview, they were asked which they found to be easier. When working with words, participants most often grouped them based on the topic, for example, technical terms (application, system, data), art-related (guitar, singer, poem), business (investment, contract, client), and so on. **P6:** *"*Groups 'data', 'system', 'control', 'database', 'application'* This is kind of software-ish."* But some more subtle and multi-faceted connections were also made, showcasing the flexibility that words can offer. **P7:** *"'Explanation' [and 'poem'], poems always need explanation or interpretation."* **P1:** *"'Honey' maybe comes with 'girlfriend', it depends if honey is honey [food] or honey [endearment], like *chuckles*."* It seems that the word space requires thinking and can be harder to interpret, but can also be more flexible and interesting, a point brought up by 4 participants. **P6:** *"Words need a lot of parsing, and thinking about 'does this work?'"* **P2:** *"*Matches 'celebration' and 'football' to 'music'.* I really like this, because it seems sooo infinitely dimensional. *laughs* It works, because there are so many directions that terms can be similar."* **P1:** *"If the domain is for exploration, maybe I use the words."* **P3:** *"For words you have to really look harder, I think. But it's also fun."* We believe that the flexibility of word similarity can be especially useful in fostering playful and creative exploration of personal data.

When looking for cat similarity, people most commonly relied on the color of the cat: black, white, orange, striped, etc. **P2:** *"Oh! We can go for black cats right away. *Groups 5 people.* Yeah, this works out nicely."* But sometimes they also used other traits, such as kittens or cats in cages. **P10:** *"I'm just looking for some baby cats."* **P3:** *"Apparently, [Person A] and [Person B] and [Person C] are similar because they all have cats with this type of head."* **P8:** *"I put these two together because they are both in cages."* In general, the opinions on how easy it was to perceive similarities in the cat space seemed to diverge, with some people saying that it was easier and others that it was harder than words. **P10:** *"You don't have to interpret it so much. You just see black cats."* **P6:** *"Cats were the hardest because I didn't really know what were the primary features."* **P7:** *"And cats were funny. I liked the cats as well, but I couldn't find much."* This suggests that the cat images, being visual, are compared more quickly, but some of the finer features may require time to interpret.

The style metaphor was regarded as the easiest for finding similarities by the majority of our participants. **P9:** *"And for the style transfer it was super intuitive, because I look at it, and it looks similar or not, this is like a no-brainer."* **P8:** *"The style visualization made me*

fully understand the connection between the researchers." This was somewhat surprising to us, the advantage appears to be that the content of the image can be ignored, with only the color and the texture encoding the similarity. **P2:** *"[Person], well, he fits color-wise, and also texture I think is similar, that's cool."* Another contributing factor is that we used 16 styles, making clusters of similarly-styled people more easily detectable. Nevertheless, participants were able to not only find the clusters of identical style, but also in-between cases of similar texture or color. **P10:** *"And these two are something in-between. Here you have really smooth area, and here something's just blurred."* **P9:** *"[Person A] is like a weird case. It feels like he's between [Person B] and probably this group above here."* **P5:** *"Oh and look here, a small [Professor] group cluster, [Professor] is the same as us, between us and here."* This is quite encouraging and we think that such "data-enriched avatars" might make for an interesting future study.

Expected and unexpected findings by participants. We observed our participants construct many similarity groupings, both around themselves and involving others. In total, we coded around 180 similarity clusters and pairs being mentioned. In 106 cases, the participants indicated whether the discovered similarity aligned with their knowledge of their colleagues. Among those instances, we counted 90 that were expected and 16 that were not. The latter cases, where the perceived metaphor does not correspond to the user's expectation are the most interesting to us, so we have manually reviewed them by computing the underlying similarity between the researchers, i.e. the data points. Interestingly, in 12 out of 16 cases, the data similarity was above the 75th percentile or 0.27 cosine, i.e. the cases were not false positives, but rather represented similarities in the research topics that were unknown to the participants but reported by Microsoft Academic. **P2:** *"My cluster doesn't make sense. ['meat', 'chocolate', 'potato']. Maybe, I don't know. [Person A] does a lot of HCI, [Person B] as well. [cosine 0.55]"* **P5:** *"I think this one is similar to these ones, but I don't see the connection between these two and [Person]. [cosine 0.32]"*

We also performed a similar investigation for the 18 instances where participants said that they expected people to be similar, but did not find the expected similarity. We found 12 cases where the perceived lack of similarity was explained by the data, i.e. similarity was below the 75th percentile. **P5:** *"But I expected that [Person] is closer to our topics. [cosine 0.17]"* **P1:** *"Interviewer (I): Who should be similar? – [Person], for example? But she's not similar. [cosine -0.04]"* **P4:** *"[Person A with 'organization'] I would put here [to 'potato', 'honey'], but the word doesn't fit. [cosine -0.02]"* Overall, we observed that most of the cases where the user's perception of the metaphor did not fulfill their expectations were actually in the data. Our prototype did not provide access to the underlying data, and it should have, in retrospect, because these cases indicate opportunities for users to verify and extend their knowledge of their colleagues. **P2:** *"I would've like to have [...] the papers, the keywords for the authors, to see how this could make sense. Because for some pairings it was surprising to see."* **P4:** *"[...] it would be*

interesting to know what types of topics are behind these images."

Metaphor creation feature underused. One thing that our study setup did not encourage enough is changing the metaphor by manually defining assignments. We structured our tool around affinity diagramming, which worked well for understanding how similarity is perceived and data connections are made, but an unintended consequence was that people focused too much on finding similarity groups, which is logical in hindsight. A few participants were even confused that the whole mapping would be recomputed once they added an initial assignment and needed additional clarification. **P7:** *"Oh, 'temperature' is different. *Tries to fit it somewhere* Ooh, now everything's different."* **P8:** *"I try a white cat for [Person]. And now, what? Did mine also change?"* In future work, we'd like to have a tool that is more suited for building fun metaphors, e.g., by presenting only a few people at a time, prompting the user to define assignments, and automatically presenting some of the interesting outcomes.

Fun and personalization. One of our main goals for this qualitative study was to see if metaphorical visualization can provide a fun and casual way of exploring data. And throughout the study we observed participants find amusing assignments and associations. **P10:** *"*Laughs* Good, [Person A], 'error'. I have to make a screenshot. [And later after the study:] *Chuckles* The [Person A] error. I still have to send it to him."* **P2:** *"Let's see if there are more baby cats around. Oh [Senior researcher] does not, *chuckles* this is like an old cat."* **P9:** *"But then it's pretty funny that [Person A] has 'maintenance' and [Person B] has 'system' it's like a maintenance system they do together *chuckles*."* **P7:** *"['cousin' is put next to 'marriage'] *Laughs* and I don't marry my cousin. But yeah, *chuckles* maybe we need the police here."*

And even more interesting were the many cases where people connected the concepts to their associations about themselves and their colleagues. **P7:** *"Uh.. *chuckles* so I like .. *laughs* I'm one of the small cats because I'm one of the youngest here."* **P1:** *"I like this [Person] 'appointment'. – I: Why? – Like, when I remember [Person], the first thing that pops into my mind is the [Seminar] timing thing. *laughs*"* **P3:** *"He looked tired like this cat *chuckles* this morning when I saw him."* **P5:** *"Look! All [Project] people are easily "confused" or arguing with each other and explaining to each other things. *Groups 'depth', 'confusion', 'argument'.*"* We believe that creating such associations between the data and their personal experience, and especially making assignments that reflect them, can allow people to introduce their personal knowledge and connect with the data metaphor.

Of course, the participants are from our own department, but still, we were pleasantly surprised that many people spontaneously expressed that they enjoyed the experience, without being prompted. **P9:** *"I'm surprised that it worked so well, really. – I: Really? – It's crazy, yeah. I mean, if I would've see a graph layout of those, just for reference, I would argue that you would have to do some major trickery to get the amounts of freedom you*

need to describe something like this." **P1:** *"This was fun."* **P5:** *"*Prompted to finish* Sorry, I'm obsessed now, I feel like I finally acclimatized to cats. *Continues to group cats.*"* **P2:** *"This was really cool."* **P4:** *"It's like playing "Memory" – I: Where you find similar pairs? – Yeah, exactly."* **P3:** *"I really like it. And I kind of would use it for memorization. I think if you connect vocabularies to funny images and so on, would be easier for learning."* **P7:** *"This is like that game where you have to insert words for a sentence and then something funny or politically wrong comes out. – I: Cards Against Humanity? – Yes *chuckles*."* This quality of data metaphors to be fun and enjoyable in themselves could be an important advantage when bringing data to casual users and applications.

5.8 Discussion

Our goal in this chapter was to demonstrate the flexibility and creativity of metaphorical visualization, so we focused on constructing a wide range of application examples. Here, we discuss our general takeaways from building data metaphors and future improvements for our concrete examples.

We see the most promising applications of metaphors to be in personal visualization, science communication and data storytelling. From this standpoint, one of the most important design considerations is making the metaphor clear and intuitive. For example, in our books-movies-games metaphor (Section 5.4) we map between semantically identical (user rating) or analogous (length) attributes. Mapping the user rating to the number of actors would be possible but would likely feel unintuitive. The metaphor should also avoid "metaphorical artifacts", which occur when we attribute to data some concept properties that are not a part of the metaphor. In our movies to stars mapping (Section 5.5), we originally did not consider distances between stars in the metaphor, but this led to interpreting movies mapped to nearby stars as related. As a response, we incorporated the distances into the metaphor and avoided false associations.

Another important aspect of metaphorical visualization lies in how the metaphor is visually represented. Unlike the traditional visual primitives like points in a scatterplot, words or artistic styles do not visually aggregate. And so, presenting the metaphor across the whole dataset at once could be challenging. When necessary, scalability should rather be pursued through interaction. This also aligns well with personal applications, where we present only small parts of the data and where interaction can improve engagement even further. And in general, metaphors designed for informal applications should strive to be visually pleasing and fun to use.

In this work, we focused on the overall approach of using metaphors and therefore opted for simple and generic methods in our implementations. In the future, many of them can be refined with more specialized techniques. For example, our study showed that using generic image similarity for cat images might not be optimal. It can be improved by

using a customized measure, e.g., eliminating the background, restricting the dataset to certain poses and using a more robust similarity function (Zhang et al. 2018). Another interesting future direction is to use generative image models to directly generate suitable images, instead of relying on fixed dataset. For our visual styles metaphor, we can apply style transfer methods that specialize on portraits (e.g., Selim et al. 2016). The distance-based metaphors are computed using simulated annealing, which provides robust and unbiased solutions, but can struggle when scaling to thousands of points. We aim to address this by first mapping a subset of "landmark" points (e.g., most frequent authors) and then using them to define linear costs for the rest of the data. Finally, we would like to conduct another study with a tool that is more interactive and encourages users to build more personalized metaphors.

We are excited about the idea of metaphorical visualization and believe that metaphors could find their usage in many informal and personalized applications. Overall, the goal was to explore the idea in its breadth and to inspire many more data metaphors and use cases.

6

Performance Prediction for Parallel Volume Rendering

This chapter is based on the following publication:

Gleb Tkachev, Steffen Frey, Christoph Müller, Valentin Bruder, and Thomas Ertl (2017). "Prediction of Distributed Volume Visualization Performance to Support Render Hardware Acquisition." In: *Proceedings of the 17th Eurographics Symposium on Parallel Graphics and Visualization*. EGPGV '17. Eurographics Association, pp. 11–20

Rendering is the final step that is present in any visualization pipeline (Section 1.2). And although it may appear to be a mere technicality compared to the feature extraction or the design of visual mappings, this is not the case. Rendering is the primary surface of interaction between the system and the user. Thus, it both constrains the previous stages and has a major impact on the overall effectiveness of the visualization system.

When considering the potential benefits that machine learning can bring to rendering in visualization, the most straight-forward applications include augmenting or replacing the rendering function with a learned model, typically with the goal of performance or fidelity improvements (see also Section 7.1). In this chapter, we consider a different angle and apply ML models to assist in the implementation of rendering algorithms via performance prediction.

Performance prediction has a variety of useful applications in complex visualization

systems. First, it supports decisions made during systems design: a performance model can be quickly used to estimate how hypothetical changes to hardware or software will affect the performance of an application, without implementing them. Second, it can be used for recognizing the need and driving the optimization process of a newly created application. The implementation of rendering algorithms is a difficult, error-prone task, and an existing model may help find issues in application performance. Finally, a performance model is useful during equipment procurement, allowing potential performance gains to be estimated before purchasing expensive hardware, helping to achieve an optimal performance-to-price ratio. While there is a significant amount of research on performance prediction for generic workloads, visualization applications have a set of special, challenging properties.

In this chapter, we will focus on volume rendering, which is a powerful method for visualization of three-dimensional data obtained through measurements or simulation. It is therefore one of the most important techniques in scientific visualization. Volume rendering is not only computationally expensive, but also an embarrassingly parallel problem. Therefore, it is often solved on (multi-layered) parallel architectures such as GPU clusters. In such scenarios, many different factors influence rendering performance, which makes the prediction of the overall rendering performance a challenging task. Furthermore, it is an inherently interactive technique, with user input having a large impact on the performance. Prediction approaches assuming stable and repetitive performance (like in many HPC codes) are inapplicable to this case, warranting specialized performance models. Finally, visual computing applications use highly parallel discrete graphics hardware, complicating the overall prediction problem with an additional node-local level of parallelism and transfer bottlenecks, both in each GPU and between GPUs of a node.

While machine learning-based approaches are widely used to capture the properties of complex systems, a general model of cluster performance should ideally be trained using data from a wide range of possible variations. In our target application scenario dealing with render hardware upgrades for GPU clusters, this means including measurements from reasonable combinations of different sizes of clusters with varying node hardware (CPUs, etc.), different number and types of GPUs, and different network interconnects. However, interchanging node hardware of the whole cluster to perform each set of measurements is impractical in most cases as it would require an enormous amount of equipment and effort.

To address this, in **Tkachev** 2017 we proposed a two-level approach to predicting parallel volume rendering performance using neural networks. Neural networks are a flexible data-driven tool that can be used without manually putting application and hardware-specific knowledge into the model. Training is performed using synthesized performance histograms, which allow for acquiring more training data from a single cluster. These can be used to emulate different node hardware by simply stalling the

nodes during local rendering for a predefined amount of time. This way, local rendering (which is practically limited by GPU performance) is decoupled from compositing (which is dominated by the network speed). In this chapter, we present the original method from **Tkachev** 2017, and further extend the work with clearer exposition, improved prediction accuracy and a comprehensive study of neural network configurations.

6.1 Related Work

Volume visualization. An overview of the current state of the art in GPU techniques for interactive large-scale volume visualization was given by Beyer et al. 2015. We also reviewed the basics in Section 2.1.2. Popularly, parallel rendering is classified into three classes: sort-first, sort-middle and sort-last (Molnar et al. 1994). In our work, we use the latter, i.e., we parallelize over volume data (in object space) by having each GPU create a local image from its own data. In a second step, local results are composited to yield the final image. The two phases differ in that the first one can be done completely independently, while compositing is done collectively and begins only after finishing all rendering tasks.

Rendering performance of large-scale systems has been the subject of several studies in the past. Peterka et al. 2008 implemented, tested and analyzed performance of parallel volume rendering on an IBM Blue Gene/P, while Howison et al. 2012 investigated the benefits of a hybrid parallel approach to volume raycasting in their work. Focusing on small to medium scale GPU visualization clusters (similar to the one used in this work), Müller et al. 2006 as well as Fogal et al. 2010 (among others) presented respective performance characteristics for volume rendering. Bruder et al. 2020 and Bruder 2022 have presented a thorough analysis of factors influencing GPU volume rendering performance and provided a set of recommendations for robust performance evaluation.

Performance prediction. Assessment, modeling and prediction of application performance in distributed environments is an active field of research in high-performance computing. Different approaches for performance modeling have been proposed in recent years. Those include performance skeletons (Sodhi et al. 2008), regression (Barnes et al. 2008), micro benchmarks (Escobar and Boppana 2016) and neural networks (Singh et al. 2007). We use neural networks as a component in our prediction model. Ipek et al. 2005 also predicted performance of a large scale parallel application applying a multilayer neural network. They trained their model with data from executions on the target platform, to capture full system complexity. Lee et al. 2007 extended the neural network approach with additional statistical techniques for preliminary data analysis and added a comparison to a piece-wise polynomial regression approach in their work. They both applied their models on well known HPC benchmarks, namely SMG2000, a semicoarsening multigrid solver (Brown et al. 2000), and High-Performance

LINPACK (Petitet 2004). However, those benchmarks only consider CPUs and have significantly different characteristics in comparison to distributed volume rendering. Unfortunately, CPU-focused techniques are typically inadequate for modeling GPU performance. In contrast, we not only consider GPUs, but also crucially have a different focus on predicting performance for hardware upgrades.

GPU performance models for guiding application optimization have been proposed by Baghsorkhi et al. 2010. They developed a compiler-based approach for analyzing GPU kernel code and modeling its performance. Zhang and Owens 2011 took a different approach but with a similar goal. They utilize micro-benchmarks to accurately measure various aspects of GPU performance, using the results to construct a performance model. Artificial neural networks for performance and power prediction of GPU applications were applied by Wu et al. 2015. They formed a collection of representative scaling behaviors by employing k-means clustering. Using neural network classifiers, they mapped those scaling behaviors to performance counter values. However, their performance estimation model was designed to predict how applications scale as a GPU configuration changes, i.e. their main objective as well as the scope differs from ours (in that they focus on single GPU performance).

In particular for parallel volume rendering, Rizzi et al. 2014 constructed an analytical model for predicting of the scaling behavior on GPU clusters, separately considering different rendering phases. Larsen et al. 2016 developed a performance model for rasterization, ray tracing and volume rendering algorithms in the context of in-situ visualization. They constructed an analytical model for the performance of every application being executed on a single machine, and used statistical methods to determine constants. Then the authors extended the model to parallel execution by introducing a related model for image compositing performance. Similarly, Bruder et al. 2017 used a combination of an analytical and a statistical performance models to help perform dynamic load balancing. The last two methods are "semi-empirical" (Hoefler et al. 2011), i.e., a combination of empirical measurements (e.g., execution times) and an analytical performance model. In contrast, we take a solely empirical approach, training a neural network and intentionally abstracting away hardware and application-specific details from the model. This has the benefit of implicitly capturing the interplay between application and hardware, without the need for manually adapting the model to a given scenario. Also, we can faster adapt to significant hardware changes via (automatic) training rather than re-modeling. Most importantly, our objective differs in that we focus on hardware procurement as our main use case rather than the question of feasibility in the context of in-situ rendering.

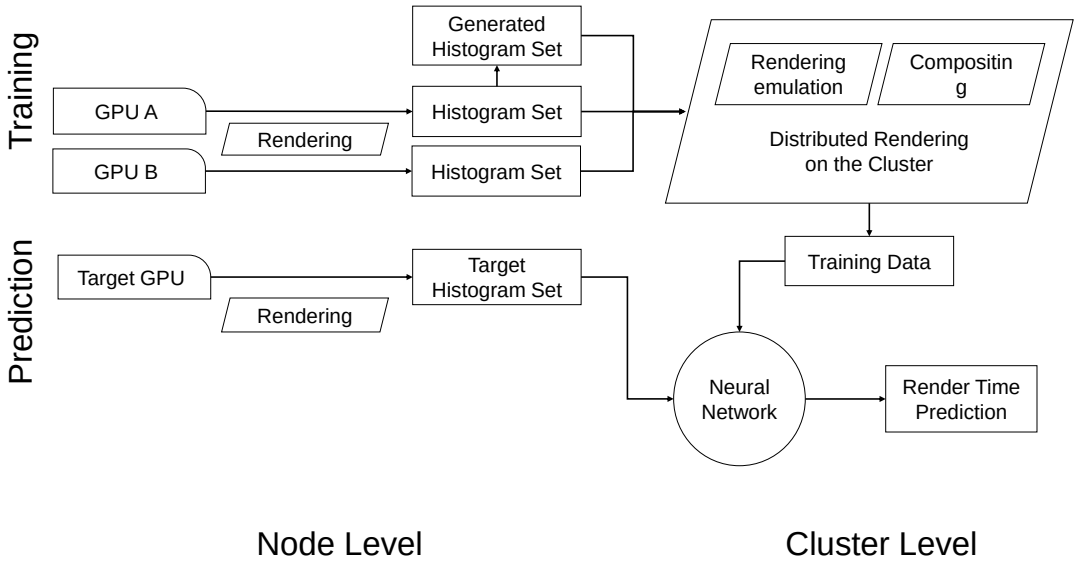


Figure 6.1 — An overview of our performance prediction approach. First, we measure local rendering performance of our training hardware, obtaining two sets of histograms (top-left). We generate an additional set of histograms by scaling our measured data. Then, we use the histograms to perform rendering emulation on the cluster (top-right), collecting the resulting cluster frame time for training our neural network. We can then predict cluster performance by measuring a histogram set on the target GPU (bottom-left), and using it as input to the model (bottom-right).

6.2 Overview

Objective. The main objective of the approach is to predict the total render time T_c of a frame, based on the size of the resulting image (I), the data set and its size (D), view parameters (V), the node hardware (H) and the cluster size (C). More formally, we are looking for a model that achieves the following mapping:

$$(I, D, V, H, C) \rightarrow T_c. \tag{6.1}$$

In particular, our focus lies on predicting the performance impact of changes to the node hardware H.

Parallel volume rendering. To develop our prediction technique, we implemented a parallel volume renderer. It can be classified as a sort-last renderer (Section 2.1.2), using object-space partitioning to parallelize the computation. Using this technique, the rendering of each frame basically consists of two major phases: local rendering and inter-node compositing. During the local rendering phase, each node independently renders its own partition, performing raycasting on GPUs. During the second phase, inter-node compositing combines the individual images into the final rendering using the 2-3 swap compositing scheme (Yu et al. 2008). In 2-3 swap, as explained in Section 2.1.2, the

compositing is performed in steps, during which nodes exchange and compose data in small groups of up to four nodes. Initially, each node has a full image with only its own volume partition rendered on it. As data is exchanged each step, the image that the node holds shrinks, while its contents become more complete, i.e., contribution of other volume partitions is taken into account. And in the end, each processor is left with a small complete chunk of the final image.

Two-level prediction approach. A distributed renderer running on a GPU cluster is a complex system exhibiting parallelism at multiple levels and featuring intricacies like network congestion. We can use neural networks to capture this complexity. To obtain acceptable results from a neural network, it needs to be trained with a lot of training data on a wide range of hardware (we evaluate the benefits of additional training data in Section 6.5.3). The more general the model should be, the more hardware, datasets and different output resolutions we need for collecting the training data to prevent the network from overfitting a particular configuration. However, the number of GPU clusters available for training is limited, and exchanging all GPUs of a cluster for extensive testing is unfeasible in practice.

We reformulate our model by splitting it into two levels similar to the two phases of an object-space distributed renderer. First, we render an image of a subset of the data on each node (local rendering). Second, a compositing phase follows, which exchanges data over the network and assembles the final image on the CPU. In our setup, the GPU only affects the local rendering phase. This allows us to abstract this part of the hardware and application parameters as a *local render time* T_n, which specifies how long it takes a node of the cluster to finish its local rendering. With this, the model from Equation 6.1 can be split into two models:

$$(I, D, V, H) \rightarrow T_n \tag{6.2}$$

$$(I, C, T_n) \rightarrow T_c \tag{6.3}$$

The first model (Equation 6.2) represents the local rendering phase, where image size (I), data set (D), view parameters (V) and node hardware (H) define the local render time (T_n). The second model (Equation 6.3) covers the compositing phase, mapping image size (I), cluster size (C) and local render time (T_n) to the final cluster frame time (T_c). An advantage of this reformulation is that Equation 6.3 does not rely on information about the hardware used for rendering, in contrast to the original model (Equation 6.1). Therefore, we can emulate different rendering hardware on a single cluster by simply stalling the nodes according to T_n. This allows us to gather more performance measurements from the cluster by emulating different local GPU rendering times without actually installing different graphics cards in the nodes (see Section 6.3). The data gathered this way can then be used for training the neural network to predict Equation 6.3. The model we eventually obtain is only tied to the hardware used for compositing and the network hardware and topology. This means that our model can make meaningful

predictions on the basis of local render time measured on a single node equipped with target hardware. A graphical overview of our approach is presented in Figure 6.1.

6.3 Emulation of Local Rendering Performance

A key technique for acquiring sufficient training data for the cluster performance model is the emulation of local render times T_n (Equation 6.2). This section covers our approach to suitably representing local rendering performance via *performance histograms*.

Motivation and objective. The main issue when taking a machine learning-based approach to performance prediction for clusters is that typically only one (or very few) specific hardware configurations are available for obtaining training data. Therefore, the training data is conceptually restricted to this one configuration, and allows prediction only in terms of scaling with the number of nodes. To circumvent this, we propose to remove local rendering from the model to only capture how communication (hardware and network topology) affects the final time, training the model to predict the cluster performance already given local performance. We represent this local performance using performance histograms, which can either be measured from arbitrary hardware available on individual machines, or even be generated artificially. This allows us to emulate different local rendering performance on a single cluster and collect more training data. Additional training data helps the cluster model (discussed in Section 6.4) to better learn the dependency between local rendering performance and overall render time, and prevents the model from implicitly adapting to particular node performance.

Performance histograms representing local render time. An *emulation run* works similarly to real rendering, but during the local rendering phase we simply stall the nodes without performing any actual computation. Once all the nodes have finished waiting, the inter-node compositing phase proceeds as normal. The nodes exchange and compose random data, but the amount of compositing and communication remains unchanged, allowing us to measure how a given local render time affects the overall cluster performance (Equation 6.3). By varying the local render time, we can simulate rendering with different combinations of hardware and application parameters, producing more training data from a single cluster. Note, that during compositing we always consider the full image, and do not make any optimizations based on footprints.

The local render time T_n obviously does not represent the time it takes to render the whole volume on a single node, but rather the time it takes for a single node of a cluster to render its partition. In our implementation, we use a static, uniform partitioning of the volume into bricks, i.e., each partition has an approximately equal number of voxels. However, render times still can differ significantly between bricks due to the perspective

projection and the early ray termination. This results in dynamic load imbalance with different nodes becoming the bottleneck under different camera orientations.

Thus, instead of using a static, predefined local render time for each node, we use a representation of node performance that can capture dynamic load imbalance. To do so, we use a distribution of local render time, which describes a probability of a node taking a certain amount of time to perform local rendering during a frame. This distribution can be expressed as a histogram, which is randomly sampled to decide how long a node needs to stall when emulating local rendering. This way, we can replicate the varying character of local render time, while still maintaining the same average performance, resulting in a reasonably accurate emulation of volume rendering performance (Figure 6.4b).

Obtaining performance histograms. As outlined in Section 6.2, our goal is to use rendering emulation to collect more training data from a single cluster. To perform more emulation runs, we require more performance histograms that describe the node performance to be emulated. The histograms can be obtained either by measuring them on various node hardware, or by generating them artificially, representing some hypothetical hardware. We explore viability of both approaches, measuring some histograms on existing hardware and generating additional histograms to get more training data.

To measure the histograms on existing node hardware, we execute a run of the volume renderer for each combination of image size, volume size and cluster size parameters (this influences the size of the volume partitions). We record the local render time for each node and frame and sort the measurements into a histogram. By measuring a histogram for each combination of the parameters, we obtain a set of histograms that captures performance of the tested node hardware. Additionally, the histogram approach allows us to uniformly represent both single and dual GPU configurations. Two GPUs act together to render the node's partition, and from the standpoint of local render time are viewed as a single faster rendering device. Thus, we can measure two sets of histograms: one in single GPU mode, and another in dual GPU mode.

Then, we generate modified histograms that are similar to those measured during actual volume rendering to produce a larger variation of training data (Figure 6.2). This noticeably improves the results of our cluster model (discussed later in Section 6.5.3). To generate artificial histograms, we take the values from a histogram previously measured on hardware under the same rendering parameters and perturb it slightly using a uniform distribution. Without perturbation, all generated histograms would be the same, making the network biased towards their particular shape. To define the domain of the histogram, i.e., the minimal and the maximal local render time, we take the values from the measured histogram and scale it by a constant, effectively imitating slower/faster hardware with similar scaling behavior. This way, we generate

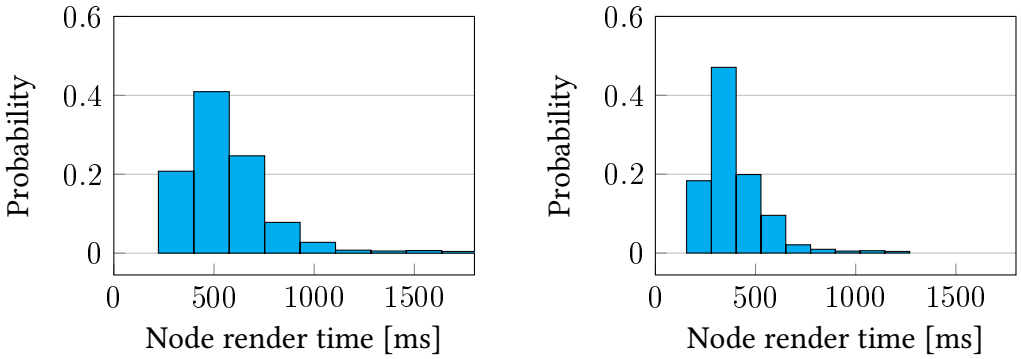


Figure 6.2 — An example of a performance histogram created from measurements (left), and artificially generated (right). They represent local rendering performance of a node of a 24-node cluster for image size 6144^2 and volume size 1024^2.

two additional sets of histograms. Note, that while the primary purpose of modifying histograms is to generate a larger variation of training data, the concept could also be applied to basically simulate arbitrary combinations of hardware and volume rendering characteristics. However, investigating this more closely is beyond the scope of this work.

6.4 Cluster Performance Model

In this section, we present our cluster performance model. For this, we train a neural network using data acquired through our rendering emulation technique (Section 6.3).

Model input and output. For our model we train a neural network that has the following input data:

Image resolution (I). Although image size is an application parameter that affects local rendering and is implicitly captured in a performance histogram, it also defines the amount of data exchanged over the network, and thus, it is useful for predicting the cluster frame time.

Cluster size (C). The number of nodes affects both the amount of data exchanged and the communication pattern.

Performance histogram (T_n). The performance histogram is fed into the network using two sets of features. The average, minimum and maximum local render time features define the domain of the histogram, which is a rough estimate of node performance. Ten bin features represent the distribution of local render time, which encodes the load imbalance of the rendering application (Section 6.3). By choosing to use ten histogram bins, we aim to maximize the resolution of the

histogram to provide more data for the network. However, a further increase of the bin number causes the histograms to have occasional gaps, introducing undesirable noise into the input data.

Cluster frame time (T_c). As output, the model provides a prediction for the cluster frame render time. Specifically, we predict the average frame time recorded over the camera path. The choice of the target variable is made in line with our emulation technique (Section 6.3): our histogram-based emulation is designed to match the average performance (Section 6.5), so by training the model to predict the average performance of the emulation, we transitively predict the average performance of actual volume rendering.

Model and training. Our architecture is a series of fully connected layers with the ReLU activation function (Glorot et al. 2011). The last layer is the output layer and consists of a single unit, since the render time is our sole output variable. Generally, the number of units and layers can be varied to control the complexity of the network. Too little complexity typically means that the network is too simple to represent important relations, while too much complexity induces the risk of overfitting. Because a good choice is very hard to make a priori, we determined the adequate structure for our network experimentally as follows. We started with a simple single-layer network, and increased the number of neurons and layers while the accuracy of prediction was improving. Using this procedure resulted in a network with two hidden layers of 16 and eight neurons, since further increase in the network's complexity yielded no improvement in the accuracy of prediction. See Section 6.5.6 for details.

For training the network, we use both measured and generated histograms. Each point of the training data maps a combination of image size, cluster size and a corresponding histogram to the resulting cluster frame time (Equation 6.3). The model is trained with the MSE loss and an L2 regularization term, controlled by the regularization parameter λ. We train a network for the values of $\lambda \in \{10^{-3}, 10^{-2}, \ldots, 10^{5}\}$, and choose λ with the smallest loss on held-out validation data.

6.5 Results

In this section, we evaluate the prediction accuracy of our model. We begin by describing the training data for the neural network. Then we assess the quality of our prediction in a cluster upgrade scenario, and also on a different cluster with a similar network configuration.

6.5.1 Implementation

Our renderer can utilize both one and multiple GPUs per cluster node. In a case of multiple GPUs, the raycasting results are locally composed among the GPUs. The raycasting uses front-to-back compositing with early ray termination and is implemented in CUDA. Once all nodes have finished local rendering, the intermediate results are exchanged among the nodes and composited into the final image using the 2-3 swap compositing scheme (Yu et al. 2008). We used asynchronous operations provided by the MSMPI implementation for our inter-node communication. The volume is partitioned using a k-d tree, which implicitly provides ordering for compositing. We statically assign a partition to each node during initialization, without performing runtime load balancing. If a node has multiple GPUs, the partition is split further, assigning a sub-partition to each GPU.

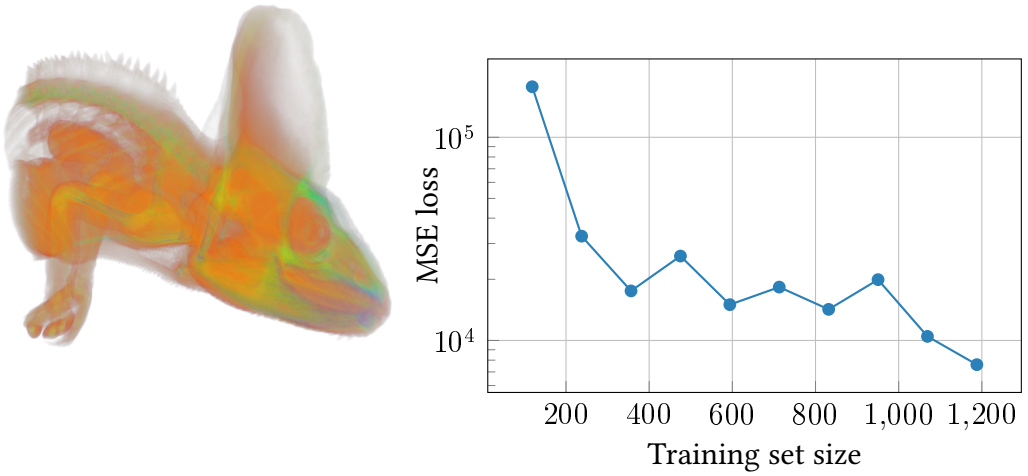
With four nodes on a 1024^3 volume with 6144^2 image size the renderer achieved an average frame time of 3545 ms, with 2084 ms spend in local rendering phase and 1461 ms in compositing. As we increased the cluster size, the frame time started to decrease, with compositing having a larger impact on performance. For example, with 32 nodes the average frame time was reduced to 2081 ms, where local rendering and compositing took 394 ms and 1687 ms respectively. Overall, both local rendering and compositing had a significant impact on the cluster performance, which showed the necessity of a two-level approach to cluster performance modeling.

6.5.2 Collecting Training Data

We acquired the training data for our model on a 33-node GPU cluster. Each node was equipped with two Intel Xeon E5620 CPUs, 24 GB RAM, two NVIDIA GeForce GTX 480 GPUs and DDR InfiniBand. The InfiniBand network had full bisectional bandwidth. For evaluation, we used a render run consisting of 72 frames while orbiting the camera twice around the volume on the XZ-plane. We recorded the local render time for each volume partition and the overall cluster frame time. We considered every combination of the following parameters (a total of 594 configurations):

- Image size $\in \{1024^2, 2048^2, 3072^2, 4096^2, 5120^2, 6144^2\}$
- Volume size $\in \{256^3, 512^3, 1024^3\}$ (scaled version of the Chameleon, Figure 6.3a, with the original size being 1024^3 voxels)
- Cluster size $\in \{1, 2, \ldots 33\}$

We measured histograms for both single and dual GPU mode for the 594 configurations mentioned above. Furthermore, from the sets of measured histograms, we artificially generated two additional sets. We did this by perturbing the histogram bins with a



(a) Chameleon **(b)** Learning curve

Figure 6.3 — (a) Test data rendering. (b) Learning curve showing improving test loss with larger training sets.

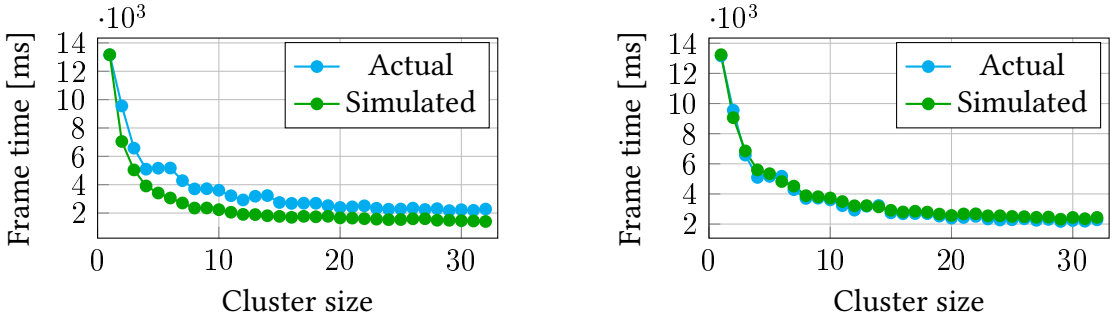
uniform distribution and scaling the domain of the histograms by a factor of 0.7. The rationale behind this factor is that it significantly changes the local rendering performance, while the result still remains within the same order of magnitude as measurements. This produces variation in the training data that helps prediction (Section 6.5.3).

Finally, we used measured and generated histograms to perform rendering emulation on the cluster, obtaining the data for training the network (see Figure 6.1 for an overview):

- S_{m1}: Measured-histogram data set, single GPU mode
- S_{m2}: Measured-histogram data set, dual GPU mode
- S_{g1}: Generated-histogram data set, derived from S_{m1}
- S_{g2}: Generated-histogram data set, derived from S_{m2}

6.5.3 Emulated and Actual Render Timings

An important aspect of our approach is that we used rendering emulation to obtain training data for our model (as discussed in Section 6.3), making it possible to train a full cluster model while measuring only on a single cluster. The benefit of additional training data can be seen in Figure 6.3b where we depict the learning curve, plotting MSE loss on the test data against the training set size. The initial steep drop corresponds to the drastic improvement made after having almost no training data. (Note, that to show more details a logarithmic scale is used on the y-axis.) As more training data is added, one can see an improvement in accuracy on the test data set (the model becomes better



(a) Simulation using simpler local render time vectors.

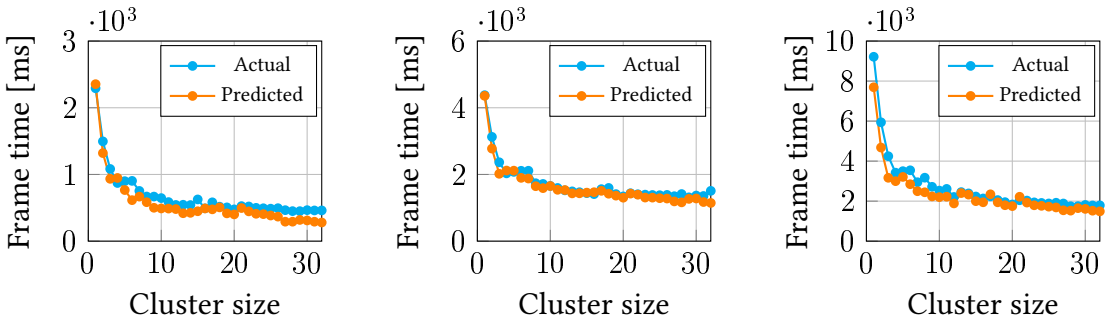
(b) Simulation using our performance histograms.

Figure 6.4 — Comparison of actual and simulated performance for 6144^2 image size and 1024^3 volume size. 6.4a: Simulation using local render time vectors, with each node stalling for the average amount of time taken by this node during an actual rendering run. 6.4b: Simulation using performance histograms, with each node sampling the distribution during runtime to determine the amount of time it should be stalling.

at generalizing to previously unseen data). This shows that the additional training data acquired through usage of performance histograms and rendering emulation improves the prediction accuracy of our model and makes the approach practical.

We investigated the accuracy of our distribution-based rendering emulation technique by comparing its performance to that of an actual rendering application. For this, we performed a set of normal rendering runs, recording not only the cluster frame time, but also the performance histograms (Figure 6.4b). We can see that the performance emulated with the histograms closely matches the performance of an actual render run.

To demonstrate the importance of using a distribution of values for our emulation, we further compared it against a simpler vector-based technique, which assigns a predefined local render time to each node. For this, local render times were averaged for each node over all frames to acquire a local render time vector. Figure 6.4a illustrates that the vector-based approach allows imitating a general performance trend, but has a large deviation from the actual performance. The vectors capture the static load imbalance, i.e., nodes having different computational load overall (e.g., due to some parts of the volume being more opaque than others), but it does not capture the dynamic load imbalance, which refers to the effect of nodes having different computational load every frame, with different nodes becoming the fastest/slowest under different camera orientations. However, our histogram-based approach covers both types of load imbalance. Therefore, it has a significantly smaller performance deviation, making it suitable for obtaining better training data.



(a) 3072^2 image size, 1024^3 volume size

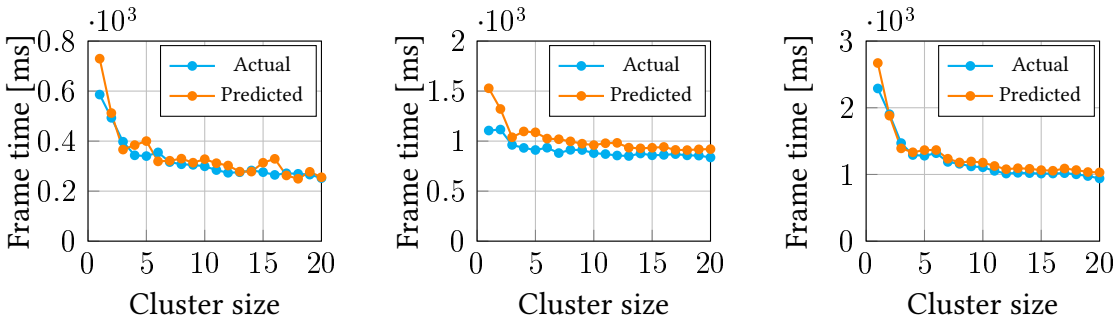
(b) 6144^2 image size, 512^3 volume size

(c) 6144^2 image size, 1024^3 volume size

Figure 6.5 — Evaluation of our model in the cluster upgrade scenario for different image and volume resolutions. The model is trained using data obtained in single GPU mode, and used to predict performance of the cluster with two GPUs per node. All three subplots represent different 'slices' of the same data, for different fixed values of image and volume size.

6.5.4 Predicting Performance of an Upgraded Cluster

We evaluated our approach in a cluster GPU upgrade scenario. Specifically, we investigated benefits of upgrading single GPU nodes to dual GPU nodes (same model). We trained our model using two data sets that were obtained from measurements in single GPU mode: a measured-histogram data set S_{m1} and a generated-histogram data set S_{g1}. No data collected in dual GPU mode was used for training. To make a prediction, we collected a set of histograms on the target hardware, i.e., on a single node equipped with two GPUs. To test our prediction, we executed the volume renderer on the whole cluster running in dual GPU mode, without any rendering emulation, thereby comparing the prediction to the actual rendering performance and not just the emulation performance. The results are presented in Figure 6.5, with our model achieving an MSE loss of $3.826 \cdot 10^4$ and an R^2 score of 0.95. The R^2 score measures how well the model predicts the data: the best score of 1.0 means a perfect prediction, and a score of 0.0 is achieved by simply predicting the mean. It can be seen that our model works well both for a smaller and larger number of nodes. For instance, in Figure 6.6b we see how the prediction matches both the steep descent in the beginning of the graph, and the "tail" of the graph, where the communication overhead prevents further performance gain. Furthermore, the model also reproduces the smaller details of the scaling curve, predicting which cluster sizes are more favorable: in Figure 6.5c both the prediction and the actual performance have a local minimum at even cluster sizes.



(a) 3072^2 image size, 1024^3 volume size

(b) 6144^2 image size, 512^3 volume size

(c) 6144^2 image size, 1024^3 volume size

Figure 6.6 — Evaluation of our model on a different cluster. The model is trained using data collected on one cluster, and used to predict performance of a different cluster with similar network configuration. All three subplots represent different 'slices' of the same data, for different fixed values of image and volume size.

6.5.5 Predicting Performance Across Different Clusters

While the main focus of our approach is on predicting the outcome of upgrading node rendering hardware, it can also be used to predict the performance of a different cluster having a similar network configuration. For this application case, the test was performed with data from a different cluster. Therefore, we trained our model with measured data sets S_{m1} and S_{m2}, using data acquired both in single and dual GPU mode. The generated data set S_{g2} was used to automatically choose a value of the regularization parameter λ during training (Section 6.4). For testing, we performed measurements on a 20-node cluster with two Intel Xeon E5-2640 v3 CPUs, 128 GB RAM, an NVIDIA Quadro M6000 and FDR InfiniBand (this is a faster network interconnect than in the cluster used for measuring the training data). As mentioned before, no rendering emulation was used for the test data, so we compared the prediction of the model to the actual rendering performance and not just the emulation performance.

The model achieves an MSE loss of $7.603 \cdot 10^3$ and an R^2 score of 0.93. It exhibits some jittering when predicting performance on smaller image and volume sizes, producing an initial spike in Figure 6.6a and Figure 6.6b. In this case, the exact shape of the curve changes between different executions of the neural network training (due to random initialization of the network's weights), but the overall trend remains the same: on a small number of nodes, the model predicts lower performance compared to measured results. An explanation to this deviation is that the GPUs of the training cluster are significantly older and slower than the ones of the test cluster, and, unlike them, benefit from parallelization even for smaller image and volume sizes. Thus, the network has

Layers	Neurons	Lambda	Test score	Val. score
1	64	1	10207.09	5949.29
1	64	10	12801.47	6200.41
1	128	0.01	8178.46	6233.40
2	16+8	1	10241.14	6289.66
		...		
2	128+64	0.01	11632.76	14039.96
1	8	1	30517.56	15841.59
3	64+32+16	0.1	17013.04	17065.43
		...		
3	8+4+2	0.01	106556.08	302147.11
3	8+4+2	1	111366.93	303968.76
3	8+4+2	0.1	138458.65	341182.30

Table 6.1 — Comparison of neural network sizes. Performance of 75 networks ordered by the validation score, best (top) to worst (bottom).

observed a steady decline in frame time, as we add more nodes to the cluster. However, the test cluster has a different scalability trend, which results in a large deviation from the model's prediction for a small number of nodes.

In all three cases we can see that the model's prediction has some bias as the model consistently predicted slower performance than what was actually achieved. We attribute this deviation to the different network interconnects, as the lower network bandwidth of the training cluster is implicitly captured in the model. Hence, when the model is used to predict performance of a cluster with a faster network, it consistently underestimates the performance. This implies, as expected, that our approach is not suitable for predicting performance of an arbitrarily different cluster. While changes in rendering hardware are covered by our histogram approach, the communication/compositing is learned by the model from the data of the training cluster, and changes in that respect are not accounted for in any way. However, our results show that we can still yield reasonable results for clusters with similar network configuration (see also Section 6.6).

6.5.6 Hyperparameter Study

To further validate our empirically chosen neural network architecture (see Section 6.4) we trained 75 different networks, varying the number of layers (1, 2, 3), the number of neurons (8, 16 ... 128) and the regularization parameter ($10^{-2}, 10^{-1} \ldots 10^{1}$). In Table 6.1 we present the results for best-, medium- and worst-performing networks, chosen based on their validation dataset score (average of 10 runs). We can see that the best-performing networks show similar results, however our '16+8' architecture

has significantly fewer neurons and hence, lower training time. Among the networks with medium performance, we can find both complex architectures (e.g., 128+64 neurons) with low regularization, and simple architectures (e.g., 8 neurons) with stronger regularization. The former suffer from overfitting the data, while the latter are too regularized to capture some of the details of the training data. Finally, the worst-performing networks are deeper networks with fewer neurons (e.g., 8+4+2 neurons) that attempt to reduce the data to a very compact representation (just 2 neurons), which is insufficient to meaningfully represent the training data, leading to poor accuracy.

6.6 Discussion, Limitations and Future Work

In our approach, we used performance histograms as a representation of node performance. However, these depend on the exact rendering scenario used for their measurement (e.g., the camera path). Choosing a different camera path for performing the rendering may have an effect on the local rendering performance (Section 6.3). Although our model conveniently abstracts away such details of local rendering, using a model trained under a certain scenario to predict performance of a significantly different scenario may potentially worsen the prediction accuracy. In this case the neural network might not have observed this significantly different performance histogram, and might give a less accurate prediction. We performed some testing of these conditions, predicting rendering performance for a camera orbiting in a different plane and with a different transfer function, observing no significant drop in prediction accuracy.

Furthermore, our histogram construction technique exerts a limitation on the complexity of the camera trajectory used for performance measurement. In particular, we construct the histograms by measuring the performance of each node during each frame of a rendering run and sorting all the measurements into bins. Thereby, we aim to capture both the dynamic load imbalance (i.e., nodes having different local render time during different frames) and the variation in the overall computational load caused by the volume being faster to render under certain camera orientations. This results in mixing together two different distributions: one characterizes load imbalance among the nodes, the other the overall computational load, which varies between the frames. For example, if we measured a histogram in a scenario where camera distance to the volume is changed significantly, the resulting histogram could be interpreted by our rendering emulation as “some nodes being slow, some being fast”, instead of “all nodes being slow during certain frames, and all nodes being fast during other frames”. Eventually, this would lead to a different emulated rendering performance, making our method less accurate. This limitation could be addressed by using a distribution of distributions, representing local render time distribution during each frame separately. For emulation, one would first choose a distribution for the current frame, and then use it to determine the local render time for each node. However, this extension has

two major challenges that require further investigation: sorting distributions into a histogram, and finding a vectorial representation suitable for training neural networks.

Furthermore, our statistical approach would benefit from a larger amount of systematically gathered empirical data of volume rendering performance. This way, one could learn about how render times are distributed in general and use this information to build more representative emulations.

For the sake of simplicity, we did not implement any advanced optimization techniques (empty-space skipping, interleaved rendering and composition, etc.) in our volume renderer. However our approach is in principle capable of handling these methods, and we would like to investigate its prediction accuracy in future work.

Our performance prediction approach is best suited for supporting equipment procurement, allowing cluster performance to be estimated by only performing measurements on one of its nodes, without purchasing the whole set of hardware. However, the model implicitly captures network hardware and topology of the training cluster and therefore has some limitation in general applicability and re-usability. Optimally, it needs to be trained on a cluster with similar conditions. This presents no problem in the case of node hardware upgrade, but can become cumbersome when building a completely new GPU cluster. Possible future extensions are therefore the emulation of the composition phase performance and a suitable representation of communication patterns. This could be used to abstract away cluster-specific details, similar to how we use performance histograms to abstract away node hardware.

One could argue that an analytical model for estimating local rendering performance, e.g., in the form of a cost-per-sample calculation, could be a better approach. However, our usage of histogram poses the advantage of being possibly a much more universal method, in terms of variations in the volume rendering technique or even applicability to other applications. Furthermore, (GPU) hardware algorithms such as caching, swizzling, 3D memory etc. can be the cause of significant performance deviations (Bruder et al. 2016).

In summary, with our approach we were able to obtain accurate predictions for our main application scenario – the upgrade of GPUs in an existing cluster. Furthermore, when using it in the context of a faster, but similar network configuration, we still achieved an adequate accuracy with a consistent error reflecting the difference in network performance. For such application cases, we intend to extend our model of cluster performance to become network-agnostic. Finally, we can gather a wider variety of measurements for various hardware to improve the model on the local rendering level as well.

7

Machine Learning in Scientific Visualization

In the chapters above, we have presented a diverse set of new methods that apply machine learning to scientific visualization. Here we would like to reflect upon our experiences and summarize some of the themes and insights developed during our research.

7.1 Types of ML Applications in (Scientific) Visualization

When it comes to applications of machine learning in visualization, we find it to be generally useful to categorize them based on where in the visualization pipeline the ML methods are applied. The visualization pipeline is a well-established concept but it exists in many different variations. In Figure 7.1, we show a simplified version of the pipeline, distinguishing four stages: data analysis, visual mapping, rendering and user interaction. Let us briefly describe the individual stages and enumerate a few examples of relevant ML applications to get an overview of the sub-field.

The *data analysis* stage encompasses operations like data loading, pre-processing and filtering. Feature extraction also occurs in this stage, which is of particular interest in the context of ML applications. For example, He et al. 2020b supported comparative visualization of a pair of climate ensembles by training a model to distinguish between their members. They effectively extracted an additional scalar feature that helps to

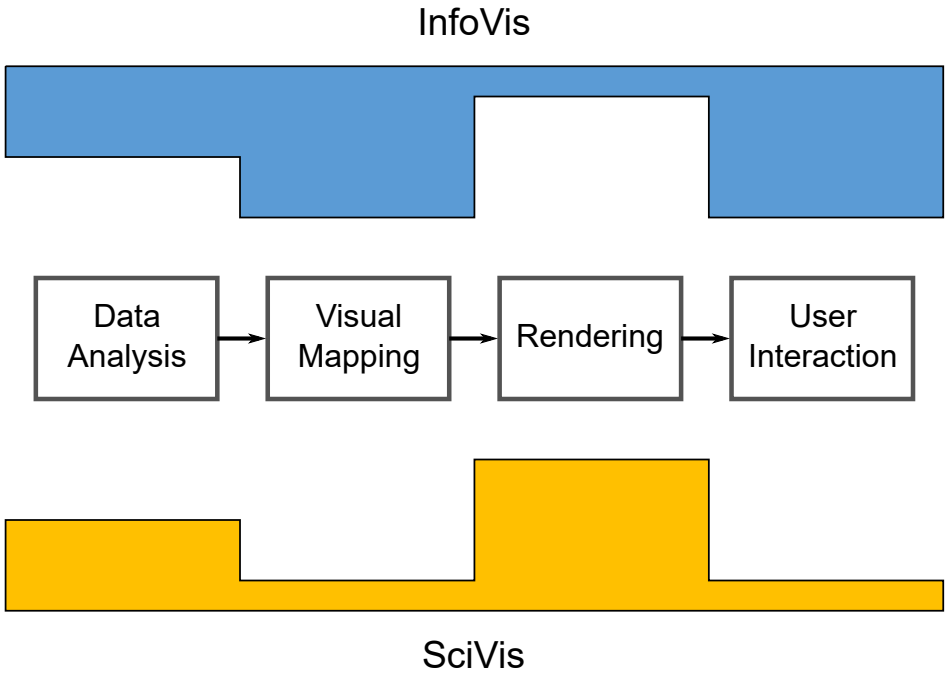


Figure 7.1 — An approximate comparison of how many ML-based methods are found in scientific and information visualization. The former is characterized by a large number of rendering applications, while the latter focuses more on visual mapping and user interaction.

group similar ensemble members and easily compare the member distributions of the two ensembles. Shi et al. 2022 proposed a graph-neural-network surrogate model for ocean simulations, which can generate data for new simulation parameters, facilitating faster parameter space analysis. Another related application in this category was proposed by Lu et al. 2021, where they used a neural network to approximate volume data, effectively compressing the data. Other data analysis applications include timestep selection, clustering, classification and similar tasks where the data is filtered, converted between different representations or additional features are extracted.

The next stage of the pipeline is *visual mapping*. Here, the prepared data from the previous step is converted to visual objects and attributes such as polygons, positions and color. Interestingly, there are very few ML applications of this kind in scientific visualization, and we turn instead to information visualization, where it is a more common occurrence. The primary topic for such applications is automated/recommended visualization. The goal is to automate the visualization *design* process, suggesting or directly producing an appropriate mapping of data to visuals. Dibia and Demiralp 2019 developed one such approach, mapping data to a formal description of an appropriate

visualization using a supervised model. And Cui et al. 2020 generated basic infographics from text, using a combination of a natural language model and a rule-based visuals generation. These are clear cases of visual mapping applications, as they neither generate additional features nor use an ML model for rendering, focusing specifically on choosing a visual representation.

The *rendering* stage of the pipeline is where the visual attributes are used to generate the final image. Rendering can present significant challenges in scientific visualization, and so many of the ML applications are concentrated here. In contrast, there are no InfoVis works in this category, since rendering is rarely a major concern for abstract data visualization. The typical ML applications of the rendering variety include learning the whole or some parts of the rendering function, performance prediction, workload optimization and so on. To name some concrete examples, Guo et al. 2020a; Han and Wang 2020 trained super-resolution models for field data. Although super-resolution, being data interpolation, can be considered a data analysis application, the evaluation focuses on rendering, and thus we attribute this family of applications to rendering. A more clear case is the work of Engel and Ropinski 2021, who demonstrated a model for fast generation of volume ambient occlusion maps, replacing a part of the rendering function with an ML model.

The final stage of the pipeline is less easily defined, but for our purposes, we describe it as having to do with the *user interaction*. Both in the usual sense of user inputs, but also the user's perception and comprehension of the visualization itself. Once again, this area contains no applications from scientific visualization and is of the primary concern in InfoVis. On the side of interaction, Fan and Hauser 2018 proposed to use a neural network to refine the brushing selection in scatterplots. The model considers the data distribution to adapt the user's input and improve the selection. An example of perception-related application is the work of Bylinskii et al. 2017, who presented an ML model of user attention for infographics. Such a model can be useful for both visualization design and summarization of existing images.

Overall, it is quite interesting that both scientific and abstract data visualization publications showcase ML applications in data processing, but specialize in the further pipeline stages. SciVis authors concentrate on rendering, while ML methods in InfoVis are applied primarily to visual mapping and user interaction (Figure 7.1). On the one hand, this is not unexpected, since this aligns well with the topics of each respective subfield. Nevertheless, we see an unexplored opportunity in the SciVis ML research, since interaction and visual representation are both of interest to the field, and yet the potential of ML models was not yet properly explored in those directions.

Type I and Type II applications. If we categorize the scientific visualization publications, we may note that when it comes to ML applications, many of them focus on rendering and data interpolation. This preference hints at a different classification facet

that can be useful in understanding the current research space. Specifically, we propose to distinguish between two groups of ML applications, which we call *Type I* and *Type II*.

The Type I methods are characterized by employing ML models to create a surrogate of an existing visualization system component, e.g., replacing the data interpolation function, the rendering function or its parts. This is done typically to improve the performance of the system, the quality of the renderings or to save storage/compute resources. The essential criterion of Type I applications is that they do not generate qualitatively different information, instead pursuing quantitative improvements. The outputs of a Type I method could be generated with a traditional non-ML algorithm, given enough computational resources, storage and time. In this sense, the Type I applications can be roughly referred to as quantitative.

In contrast, Type II applications seek to produce additional information that qualitatively differs from the products of the original pipeline. This can come in the form of feature extraction, summarization or providing new ways of interacting with the visualization. Instead of quantitative improvements, Type II methods seek to qualitatively enhance the visualization system with new components.

In Figure 7.2, we depict recent publications on ML methods in SciVis, classifying them according to the vis pipeline and the Type I/II dichotomy. As we can see, most of the publications focus on quantitative Type I applications of machine learning. The rendering methods are particularly common. In pale orange, we show approaches to volume super-resolution and interpolation, which can be considered as belonging to the data processing stage. However, they are mainly evaluated in the context of rendering the data directly, and so we lean towards classifying them as rendering applications.

This Type I prevalence is not particularly surprising when one considers the advantages of Type I applications. First, they offer a clearly defined and relevant problem, such as performance optimization. Rendering performance (as a goal) is well-understood and improvements in this space are always desired. The results are innately quantifiable and offer a reliable and convincing evaluation. Most importantly, since Type I applications aim to replace an existing algorithm or replicate existing data, they have access to supervised data, which makes the development of ML methods much more straightforward. For many of the applications, e.g., learning the rendering function, additional data can be easily generated given enough computational resources. Finally, Type I applications can often build on top of existing ML methods from computer vision and computer graphics, for instance, extending image/video models to handle volume data. Combined with supervised data, this provides a solid methodological foundation for new research.

Comparing these observations to the Type II applications, the problem of achieving qualitative improvements in the form of better features, a comprehensive overview or generally helping the user obtain additional insight is rather vague. The reviewers

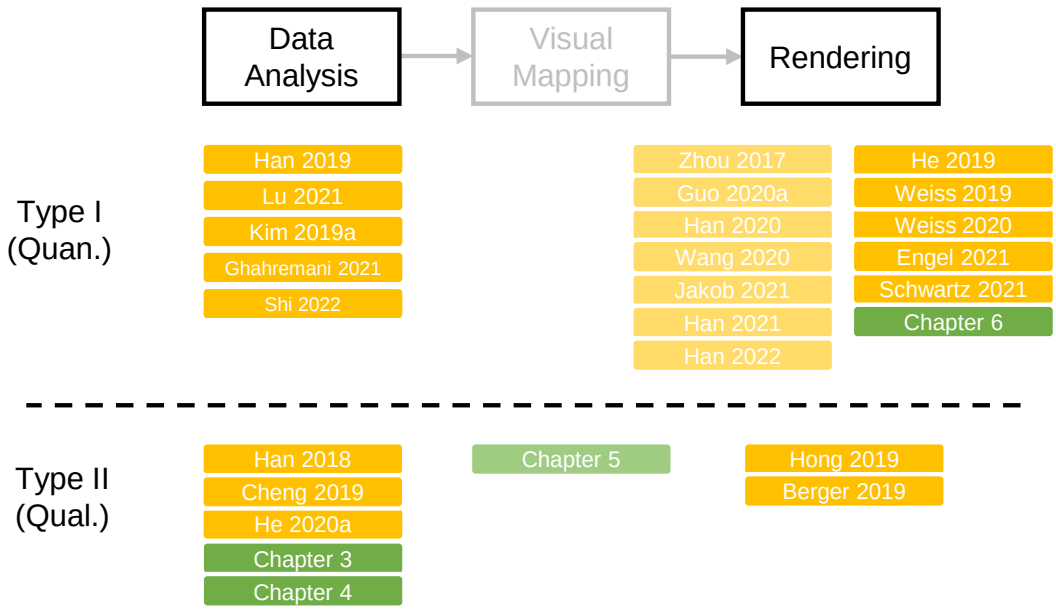


Figure 7.2 — A two-way classification of recent ML applications in scientific visualization. Type I methods focus on quantitative improvements and Type II on qualitative. Most publications are about Type I rendering applications. The pale-orange methods train super-resolution models and can be considered as data processing applications. In this work (in green), we focused primarily on the under-explored Type II methods.

typically need to be convinced that the problem (before even considering the solution) has practical merit. It is also difficult to judge whether the proposed solution actually solves the problem. The main obstacle here is that no benchmarks are available and qualitative evaluation is often the only option, leading to a less clear-cut evaluation. Furthermore, Type II applications have no supervised data available, as there are no labeled datasets of "good overviews" or "useful features", and conceptually, there cannot be any, due to heterogeneity of the problem space. Finally, since no supervised data is available, Type II approaches rely on unsupervised methods that are typically less effective and need significant adaptation to be useful.

With the above differences in mind, it is understandable why Type I applications are so much more common – they offer a more solid research path, from problem definition, to development and publication. But at the same time, we find Type II applications to be the more exciting because they have the potential of enhancing the visualization system with new analytical capabilities. We believe that this area can yield many promising methods but it remains under-explored due to the issues outlined above. In this thesis, we attempted to remedy the situation and proposed several of such Type II

applications, shown with green in Figure 7.2. They are situated in different locations along the visualization pipeline and concentrate on qualitative contributions. In the following sections, we summarize challenges that commonly occur in developing Type II applications and offer some advice on how to overcome them. We hope that these insights might be useful in continuing this line of research.

7.2 Self-supervised Learning

One of the main challenges for Type II ML applications is the lack of supervised data. While it is possible to collect, e.g., rendered images or downsampled volume data for the Type I scenarios, it is usually not possible for qualitative applications, for several reasons. The first reason is that the domain expert's time is too valuable to be spent labeling their data. And this task cannot be outsourced, since deep expertise is required to understand the original data. Second, it is difficult to even formalize some of the tasks addressed by the Type II methods. The expert could probably classify different types of events and behavior in the data, or perhaps even mark some anomalies or processes of particular interest. But it is difficult to imagine how to collect a dataset of "good overviews", since the answer is highly task-dependent and likely very subjective. The final and biggest issue is that the target domains of scientific visualization are highly heterogeneous, divided into domains with many sub-domains that in turn study many different phenomena. In fact, it is quite likely that a labeled visualization dataset created by a domain scientist might not be useful even to a colleague sharing their office.

This means that we have to contend with unsupervised data. And we believe that self-supervised learning offers the most promising solution for visualization applications. The motivation behind self-supervised learning is that labeled data is scarce, but there is often an abundance of unsupervised data that should be utilized in some way. The idea is to formulate an auxiliary supervised task from the unsupervised data (called the pretext task), and train a model to solve that task. While the model is training on the pretext task it is learning a feature space that can be useful not only for solving the pretext task, but for other tasks on the same data. For example, a vision model trained to fill-in missing parts of images is likely to learn low-level features that can be helpful for a wide range of vision tasks. See Section 2.2.5 for more detail.

A typical next step for a self-supervised application is to use a small labeled dataset to fine-tune the pretrained model on the downstream task. The fine-tuned model generally displays better performance than the model trained purely on the small supervised dataset. However, in visualization applications we typically have no supervised data at all, and so we cannot rely on supervised fine-tuning.

Our solution to this issue is prosaic, but can be surprisingly effective: we need to carefully design the pretext task to be as similar as possible to the downstream task.

We also should consider how the pretrained model will be used during inference on our downstream task and modify the model architecture to support our usage. By introducing appropriate inductive biases into the pretext task, the training procedure and the model architecture, we can obtain a model that has reasonable performance despite training on a different task. See Section 4.3 for an example of how we trained a similarity model without any supervised data. Another important factor is that visualization applications present a unique advantage compared to pure machine learning – they are typically interactive and have "access" to a domain expert at the time of inference. Even though we are unlikely to have any labeled data, we can compensate for it by relying on the user's live feedback.

7.3 User Interaction

User interaction is an important component of most visualization systems, especially when dealing with large and complex data. And it can be equally important for successful integration of ML models that are not trained on supervised data, like the self-supervised approaches discussed in the previous section.

Nevertheless, integrating the user with an ML model is generally considered to be difficult due to the black-box nature of the latter. A typical supervised approach would be to collect a supervised dataset of user inputs and correct responses and simply feed the user's input to the model during training. Of course, such datasets are unavailable in most visualization scenarios. In what we would call the "Vis4ML" approach to interaction, the trained model is evaluated on various data points, collecting and aggregating its predictions, gradients, weights and any domain-appropriate metrics. The user would then be presented with complex visualizations, enabling them to select the more appropriate model/prediction and better understand the causes of the model's output. This approach can perhaps be appropriate for an ML expert (the target audience in this case), but is unlikely to be useful for scientists in other domains.

Instead, we propose to continue building on the idea of self-supervised learning. The user input can compensate for the lack of supervised data during training because this live input is significantly more valuable than the static training data. Unlike the training data that should sufficiently cover the complete data domain, live feedback is concentrated on what the user is actually interested in, and is thus much more relevant. Furthermore, the user can adapt and correct the model's predictions, effectively sampling the most informative data points. Finally, a pretrained model provides a rich feature space, in which only a few user-provided data points may be sufficient for an accurate prediction. As an illustration, consider the projected latent space from our work on autoencoders for feature extraction (Gadirov et al. 2021), shown in Figure 7.3. Here, an autoencoder was trained on timesteps of a 2D+T simulation ensemble. The

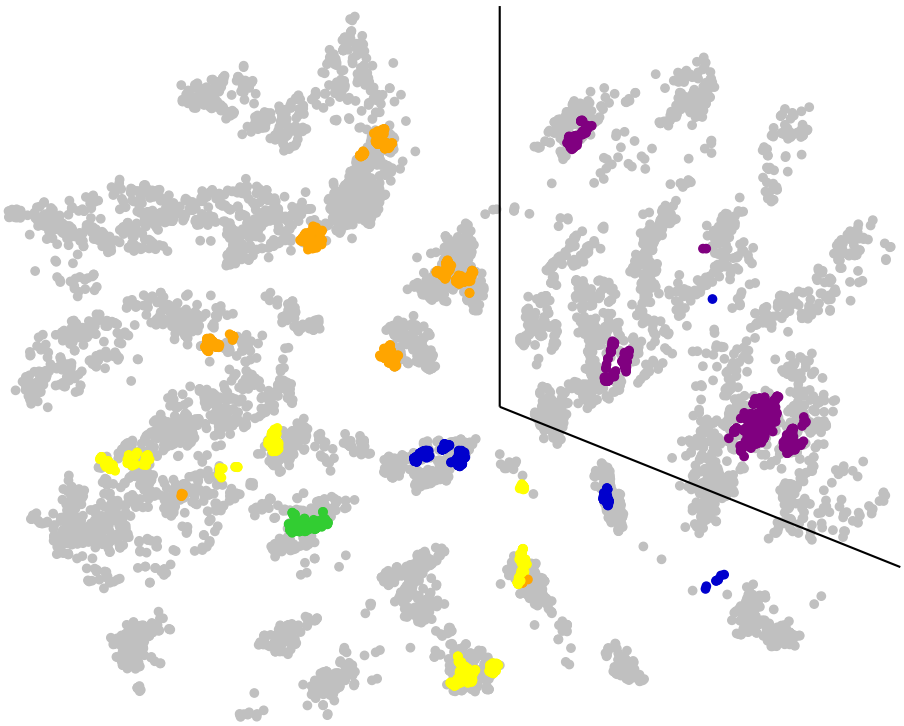


Figure 7.3 — A UMAP projection of the latent space of a Sliced-Wasserstein autoencoder trained on timesteps of a 2D+T ensemble dataset, e.g., the dark-blue class. So if the downstream task was to classify different classes, this could be achieved with only a few user examples, producing a simple decision boundary (the black line).

encoded timesteps were then projected with UMAP and are shown in the scatterplot. As part of our evaluation, we manually labeled some of the samples according to their behavior type. Notice that in this feature space the classes are relatively easy to separate, so only a few user samples would be needed to solve the downstream task, e.g., to locate timesteps with a certain behavior. This is an optimistic example, but it demonstrates the overall idea: user inputs provided interactively and considered in a learned higher-level feature space are significantly more data efficient than the supervised training data.

We can further improve the effectiveness of user interaction for self-supervised models if we also consider the model's architecture and training procedure in this context. Just as before, a core machine-learning principle applies – keeping the training regime as close to inference as possible. Therefore we should design our training regime with the future interaction in mind. For example, in Section 4.3 we enabled the user to perform similarity queries on spatiotemporal data. Since we computed the results based on the distances in the feature space of our model, i.e., nearby points in the feature space were

returned as similar, we also explicitly constrained the model to use a distance operator when training. Thereby, we encourage the training procedure to produce a space with meaningful distances, rather than hope that the distances would be representative of similarity. Furthermore, we allowed the user to provide multiple examples in their query, and accordingly, we also trained the model not on pairwise comparisons, but by comparing a point to a set of points. These measures bring the model and the pretext task closer to our target usage and make a noticeable difference for the model performance. In summary, the design of the entire pipeline should take into account how the users will interact with the system.

7.4 Evaluation

Likely the most difficult challenge faced by machine learning applications in visualization is evaluation. The problem of evaluation is prominent in both machine learning and visualization as standalone fields. Although benchmark datasets are available for many established ML problems, they do not always capture the target task perfectly and can be difficult to formalize, e.g., consider the BLEU and ROUGE metrics used in machine translation, which operate by comparing the model's outputs to a set of correct outputs produced by humans. While they correlate well with human judgement, it is clear that comparing to a set of reference translations can be insufficient to account for all possible and valid variations. Another issue is that over time the well-known benchmark datasets can get over-tuned by the competing methods, possibly neglecting some of the real-world usage scenarios.

In visualization, evaluation is an even more difficult subject. The spectrum of different methodologies ranges from purely qualitative interview-based evaluation to highly quantitative rendering performance metrics. In short, it is as broad as the research topics themselves and the discussion about which type of evaluation is appropriate for a given approach is still ongoing.

For ML applications in visualization, the evaluation approach largely follows the Type I-II divide. The Type I ML applications typically employ a quantitative evaluation that features established performance or quality metrics computed on common datasets. Overall, the approach is similar to pure machine learning, except that the datasets do not represent the final task, but rather the typical application contexts, e.g., common CFD datasets.

In comparison, Type II applications often cannot be evaluated quantitatively, and similarly to many other visualization approaches, rely on more qualitative evaluation methods, which can be difficult to implement. However, there is also a unique challenge present: the machine learning development workflow inherently relies on quantitative evaluation. Different models, parameters and training regimes need to be assessed

on daily basis in the process of developing an effective ML pipeline. In this context, qualitative evaluation would be infeasible and likely ineffective, as users start to form biases and expectations, affecting the outcome.

Self-supervised learning provides a part of the solution. While Type II applications do not have access to supervised data on the downstream task, the pretext task allows some form of quantification. The models can at least be judged on their pretext performance, which should correlate with the downstream task, given that the two tasks are chosen to be sufficiently similar. Furthermore, we can design the pretext task to have more useful performance metrics. For example, we recommend to define the task as a classification problem, as it provides metrics that are more easily interpreted. Compare an outcome of training a regression model (Chapter 3) or an autoencoder, which would produce an MSE value like "16.4", to a binary classification model with accuracy of 91% on the pretext task. The former metric is rather meaningless because it is difficult to judge whether the model is performing well. And any changes to the data or the sampling process would make the values incomparable. In contrast, the accuracy value is more easily interpreted, especially if we control the pretext task. Knowing that the data is generated to have a 50-50 distribution of positive and negative samples, and with some minor understanding of the application domain, we can be reasonably sure that the model is at least somewhat successful in solving the task. We can then further investigate the results, compute the F-score, study the mispredicted cases, and so on.

Of course, even a careful design of the pretext task does not guarantee the model's ultimate performance. The final evaluation must be performed on the downstream task, which is likely to be qualitative and lacking metrics for a Type II application. This leads to our main recommendation: when looking to do research on ML applications in SciVis, it is a good idea to choose problems that are amenable to quantification. This may appear to be running against our main point: did we not argue for qualitative Type II contributions? There are two nuances to consider here.

First, the binary Type I/II classification is convenient for our discussion of ML applications, but it is a simplification. Rather, one could consider it to be a continuum, with extremes corresponding to applications fulfilling all the criteria outlined before, and many intermediate methods that may display the characteristics of both. For instance, considering the two key publications of this thesis, the irregularity detection method (**Tkachev** et al. 2021a) is a more pure example of a Type II application. Our later work on similarity learning (**Tkachev** et al. 2021c) is a Type II application that departs from the extreme of this spectrum.

The second and the more important point is that a Type II qualitative application does not imply a qualitative evaluation. In fact, this is precisely why our similarity method departs from the pure image of a Type II application – it employs both quantitative and qualitative evaluation methods. The former allowed us to develop the ML pipeline

and provide a thorough comparison to alternative methods, while the latter offers a domain-situated test of the method's utility. The quantitative aspect connects with the ML requirements of the approach, which in term helps to address the qualitative visualization problem. Generalizing from this experience, we would argue that this combination of a Type II qualitative contribution and a possibility of at least partial quantification is the most desirable quality when searching for a potential ML applications in visualization.

Closing the subject of evaluation, we would like to bring up the question of benchmark tasks and datasets for ML applications in visualization. It would be easy to say that there is an unfulfilled need of better quantification and thus a need for a shared set of benchmark datasets for the more qualitative applications of ML. However, this is difficult in visualization, which is very diverse in both its application domains and user tasks. By seeking to formalize the visualization problems, we are perhaps trying to turn them into machine learning, losing sight of what is important in visualization. In this light, we think it would be impractical to try and collate evaluation datasets for Type II applications. The sub-area of ML in SciVis is still new and unexplored, and we would rather argue for a careful choice of applications, guided by the suggestions in this chapter. When the most promising directions are discovered, then perhaps more comprehensive evaluation data could be collected. But until then, it is likely that the Type II applications will still rely on a significant degree of qualitative evaluation.

7.5 Future Directions

Overall, we see great promise in applying machine learning to scientific visualization. We have generally argued for self-supervised learning approaches, since scientific data is usually unlabeled, but often large and with enough metadata (coordinates, simulation parameters, multiple fields, etc.) to set up a pretext task for training. We proposed one such task, but many other possibilities should be compared in the context of visualization. For instance, from our experience, we suspect that a reconstruction task, like in an auto-encoder, is less useful than a contrastive classification problem (Section 4.4) when it comes to learning a features space that distinguishes different behavior types. A more thorough and complete analysis may reveal many similar insights. One could experiment with not only the many existing image-centric tasks (Doersch and Zisserman 2017), but devise new ones specialized to scientific data, e.g., stability prediction, variable reconstruction, etc. And it would be particularly interesting to incorporate the simulation parameters into the model, for example, distinguishing between simulations under different physical conditions.

One future research area related to self-supervised learning is sharing pretrained models for scientific data. Pretrained models are commonly used as feature extractors in

computer vision and are a prominent trend in natural language processing. Training extremely large models on field data is infeasible and likely impractical, but a moderately-sized pretrained convolutional stack could be useful for many applications. Unsupervised model pre-training started to crop up in recent work (e.g., Han et al. 2021), it is still used for a single downstream task or within the same dataset. This trend is likely to continue to develop in the future, and so it would be of value to provide a comprehensive investigation across different datasets, models and tasks. An established and reliable pretrained volume/vector-field feature extractor can help accelerate other research in the area by reducing the training costs and the data requirements.

A potential application of self-supervised models would be to bridge the gap between data-rich and data-poor contexts. We are especially interested in transferring models trained on simulation data to help with the analysis of experimental data. Experiments are complicated and expensive, usually generating less or incomplete data, compared to a simulation. Newly acquired experimental data may not allow ML models to be trained to help with the analysis, however, a model pretrained on simulation could be fine-tuned to solve the issue. Furthermore, simulation-trained models can help fill in the gaps in the measurements, e.g., reconstructing missing variables from the available observations.

Another interesting direction for future work is interpretable ML. Providing additional insight into the learned feature spaces can help significantly with user interaction. A common approach is to perform feature visualization, computing relevance scores (Lapuschkin et al. 2015) or visualizing input patches that activate specific latent dimensions the most (Olah et al. 2017). This is often done when applying visualization to machine learning models, but can also support the reverse direction. However, when it comes to interpretability, we believe that the most promising approach is to apply additional constraints to the model, enouraging its features to be interpretable. For instance, sparsity (Makhzani and Frey 2014) and latent distribution constraints (Higgins et al. 2017) could be utilized to disentagle the latent dimensions (Gadirov et al. 2021). We can also apply spatial attention layers to attribute different parts of the data to latent dimensions of the model. Both visualization and constraints could even be combined, visualizing the model's features while constraining them to be more interpretable.

8

Conclusion

Computational methods are an essential component of the modern scientific process. The development of computer hardware and simulation technologies produce ever larger amounts of data that needs to be analyzed by the experts. Current works in scientific visualization propose many potent solutions, but they inevitably become specialized towards certain domains and applications. This can lead to young research domains lacking the tools they need to benefit from the newly acquired data.

In this thesis, we aimed to address this challenge by developing domain-agnostic visualization methods with the help of machine learning. First, we proposed an approach based on local prediction in spatiotemporal volumes, detecting regions of irregular behavior (Chapter 3). We showed that the method can be used to find anomalous local behavior, construct an overview of the dataset and automatically select important timesteps across different application domains. Next, we developed two techniques for learning spatiotemporal similarity metrics directly from the raw data (Chapter 4). Here we focused on the analysis of ensemble data – a particularly challenging scenario in scientific visualization. We demonstrated that it is possible to extract meaningful similarity for both simulation and experimental ensembles without any supervised data. The learned similarity matched the results obtained manually by the experts or with the help of specialized features. In Chapter 5, we presented a unique method for building data metaphors, which can connect data across any two domains. The method is highly flexible, personalizable and can be of particular use in communicating science to wider audiences. Then, we discussed our method for performance prediction of parallel volume rendering (Chapter 6). Once again, we developed a training approach that side-steps the need for expensive supervised data and can make predictions given

measurements from only a single cluster node.

We summarized our findings on integrating machine learning into visualization applications in Chapter 7. We presented the current research trends in this area and outlined the two major types of ML applications, their goals, advantages and current problems. The applications of the qualitative type are of a particular interest to visualization research, but they face many unique challenges, which we ourselves experienced. We discussed these issues of missing supervised data, user integration and evaluation, presenting our views on how they can be alleviated. We see self-supervised methods as being particularly potent in SciVis, and showed how careful and holistic design of the training pipeline can help adapt them to visualization.

Of course, this thesis only scratches the surface of how machine learning can be integrated into scientific visualization, and many possibilities remain unexplored. The more method-specific improvements were discussed in their respective chapters, while the higher-level directions were presented in Chapter 7. For example, we see the need for further analysis of self-supervised tasks and their impact on the extracted features, both adapting existing tasks from computer vision and developing specialized methods for scientific data. Another potential direction are pretrained feature extractors that already helped to accelerate research in other domains and could also be constructed for simulation data. Finally, techniques that improve interpretability of the learned models can prove to be useful in visualization, but should be carefully incorporated into the model and its training.

Author's Work

Agarwal, Shivam, **Gleb Tkachev**, Michel Wermelinger, and Fabian Beck (2020). "Visualizing Sets and Changes in Membership Using Layered Set Intersection Graphs." In: *VMV: Vision, Modeling, and Visualization*. Vol. VMV2020. The Eurographics Association, pp. 69–78.

Gadirov, Hamid, **Gleb Tkachev**, Thomas Ertl, and Steffen Frey (2021). "Evaluation and Selection of Autoencoders for Expressive Dimensionality Reduction of Spatial Ensembles." In: *International Symposium on Visual Computing*. Springer, pp. 222–234.

Heinemann, Moritz, Steffen Frey, **Gleb Tkachev**, Alexander Straub, Filip Sadlo, and Thomas Ertl (2021). "Visual Analysis of Droplet Dynamics in Large-Scale Multiphase Spray Simulations." In: *Journal of Visualization* 24.5, pp. 943–961.

Tabiai, Ilyass, **Gleb Tkachev**, Patrick Diehl, Steffen Frey, Thomas Ertl, Daniel Therriault, and Martin Lévesque (2019). "Hybrid Image Processing Approach for Autonomous Crack Area Detection and Tracking Using Local Digital Image Correlation Results Applied to Single-Fiber Interfacial Debonding." In: *Engineering Fracture Mechanics* 216, p. 106485.

Tkachev, **Gleb** (2022). *PyPlant: A Python Framework for Cached Function Pipelines*. URL: `https://doi.org/10.18419/darus-2249`.

Tkachev, **Gleb**, Rene Cutura, Michael Sedlmair, Steffen Frey, and Thomas Ertl (2022). "Metaphorical Visualization: Mapping Data to Familiar Concepts." In: *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI EA '22. Association for Computing Machinery, DOI: 10.1145/3491101.3516393.

Tkachev, **Gleb**, Steffen Frey, and Thomas Ertl (2021a). "Local Prediction Models for Spatiotemporal Volume Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 27.7, pp. 3091–3108.

Tkachev, **Gleb**, Steffen Frey, and Thomas Ertl (2021b). "S4: Self-supervised Learning of Spatiotemporal Similarity." In: *IEEE Transactions on Visualization and Computer Graphics* DOI: 10.1109/TVCG.2021.3101418.

Tkachev, **Gleb**, Steffen Frey, Christoph Müller, Valentin Bruder, and Thomas Ertl (2017). "Prediction of Distributed Volume Visualization Performance to Support Render Hardware Acquisition." In: *Proceedings of the 17th Eurographics Symposium on Parallel Graphics and Visualization*. EGPGV '17. Eurographics Association, pp. 11–20.

References

Agarwal, Shivam, **Gleb Tkachev**, Michel Wermelinger, and Fabian Beck (2020). "Visualizing Sets and Changes in Membership Using Layered Set Intersection Graphs." In: *VMV: Vision, Modeling, and Visualization*. Vol. VMV2020. The Eurographics Association, pp. 69–78 (cit. on p. 7).

Ahmed, Ejaz, Michael Jones, and Tim K. Marks (2015). "An Improved Deep Learning Architecture for Person Re-Identification." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3908–3916 (cit. on p. 73).

Amodei, Dario and Danny Hernandez (n.d.). *AI and Compute*. OpenAI. URL: `https://openai.com/blog/ai-and-compute/` (cit. on p. 26).

Andrienko, G., N. Andrienko, S. Bremm, T. Schreck, T. Von Landesberger, P. Bak, and D. Keim (2010). "Space-in-Time and Time-in-Space Self-Organizing Maps for Exploring Spatiotemporal Patterns." In: *Computer Graphics Forum* 29.3 (3), pp. 913–922 (cit. on p. 37).

Antonov, Alexander (2019). *Steam Games Complete Dataset*. URL: `https://kaggle.com/trolukovich/steam-games-complete-dataset` (cit. on p. 123).

Artetxe, Mikel, Gorka Labaka, and Eneko Agirre (2018). "A Robust Self-Learning Method for Fully Unsupervised Cross-Lingual Mappings of Word Embeddings." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 789–798 (cit. on p. 112).

Bach, Benjamin, Pierre Dragicevic, Daniel Archambault, Christophe Hurter, and Sheelagh Carpendale (2016). "A Descriptive Framework for Temporal Data Visualizations Based on Generalized Space-Time Cubes: Generalized Space-Time Cube." In: *Computer Graphics Forum* 36.6, pp. 36–61 (cit. on p. 36).

Baghsorkhi, Sara S, Matthieu Delahaye, Sanjay J Patel, William D Gropp, and Wen-mei W Hwu (2010). "An Adaptive Performance Modeling Tool for GPU Architectures." In: *ACM Sigplan Notices*. Vol. 45. 5, pp. 105–114 (cit. on p. 144).

Balabanian, Jean-Paul, Ivan Viola, Torsten Möller, and Eduard Gröller (2008). "Temporal Styles for Time-Varying Volume Data." In: *Proceedings of 3DPVT'08 - the Fourth International Symposium on 3D Data Processing, Visualization and Transmission*, pp. 81–89 (cit. on pp. 35, 36).

Baldi, Pierre and Yves Chauvin (1993). "Neural Networks for Fingerprint Recognition." In: *Neural Computation* 5.3, pp. 402–418 (cit. on p. 32).

Bar-Joseph, Ziv, David K. Gifford, and Tommi S. Jaakkola (2001). "Fast Optimal Leaf Ordering for Hierarchical Clustering." In: *Intelligent Systems in Molecular Biology*. Vol. 17, pp. 22–29 (cit. on p. 75).

Barnes, Bradley J., Barry Rountree, David K. Lowenthal, Jaxk Reeves, Bronis de Supinski, and Martin Schulz (2008). "A Regression-Based Approach to Scalability Prediction." In: *Proceedings of the ACM/IEEE Conference on Supercomputing*, pp. 368–377 (cit. on p. 143).

Baur, Dominikus, Frederik Seiffert, Michael Sedlmair, and Sebastian Boring (2010). "The Streams of Our Lives: Visualizing Listening Histories in Context." In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (6), pp. 1119–1128 (cit. on p. 110).

Bell, Sean and Kavita Bala (2015). "Learning Visual Similarity for Product Design with Convolutional Neural Networks." In: *ACM Transactions on Graphics* 34.4 (4), 98:1–98:10 (cit. on p. 112).

Benko, Hrvoje and Steven Feiner (2007). "Balloon Selection: A Multi-Finger Technique for Accurate Low-Fatigue 3D Selection." In: *2007 IEEE Symposium on 3D User Interfaces* (cit. on p. 111).

Berger, M., J. Li, and J. A. Levine (2019). "A Generative Model for Volume Rendering." In: *IEEE Transactions on Visualization and Computer Graphics* 25.4 (4), pp. 1636–1650 (cit. on p. 37).

Bertasius, Gedas, Lorenzo Torresani, and Jianbo Shi (2018). "Object Detection in Video with Spatiotemporal Sampling Networks." In: *2018 European Conference on Computer Vision*, pp. 342–357 (cit. on p. 73).

Beyer, Johanna, Markus Hadwiger, and Hanspeter Pfister (2015). "State-of-the-Art in GPU-Based Large-Scale Volume Visualization." In: *Computer Graphics Forum* 34.8, pp. 13–37 (cit. on p. 143).

Biswas, Ayan, Guang Lin, Xiaotong Liu, and Han-Wei Shen (2017). "Visualization of Time-Varying Weather Ensembles across Multiple Resolutions." In: *IEEE Transactions on Visualization and Computer Graphics* 23.1, pp. 841–850 (cit. on p. 10).

Bonneau, Georges-Pierre, Hans-Christian Hege, Chris R. Johnson, Manuel M. Oliveira, Kristin Potter, Penny Rheingans, and Thomas Schultz (2014). "Overview and State-of-the-Art of Uncertainty Visualization." In: *Scientific Visualization: Uncertainty, Multifield, Biomedical, and Scalable Visualization*. Mathematics and Visualization. Springer, pp. 3–27 (cit. on p. 10).

Bordoloi, U.D. and H.-W Shen (2005). "View Selection for Volume Rendering." In: *Proceedings of the IEEE Visualization Conference*, pp. 487–494 (cit. on p. 37).

Borkiewicz, Kalina, A J Christensen, Ryan Wyatt, and Ernest T. Wright (2020). "Introduction to Cinematic Scientific Visualization." In: *ACM SIGGRAPH 2020 Courses*. SIGGRAPH '20. Association for Computing Machinery, pp. 1–267 (cit. on p. 111).

Borkiewicz, Kalina, AJ Christensen, Helen-Nicole Kostis, Greg Shirah, and Ryan Wyatt (2019). "Cinematic Scientific Visualization: The Art of Communicating Science." In: *ACM SIGGRAPH 2019 Courses*. SIGGRAPH '19. Association for Computing Machinery, pp. 1–273 (cit. on p. 110).

Borkin, Michelle A., Azalea A. Vo, Zoya Bylinskii, Phillip Isola, Shashank Sunkavalli, Aude Oliva, and Hanspeter Pfister (2013). "What Makes a Visualization Memorable?" In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (12), pp. 2306–2315 (cit. on p. 111).

Bossavit, Benoît, Asier Marzo, Oscar Ardaiz, Luis Diaz De Cerio, and Alfredo Pina (2014). "Design Choices and Their Implications for 3D Mid-Air Manipulation Techniques." In: *Presence: Teleoperators and Virtual Environments* 23.4 (4), pp. 377–392 (cit. on p. 111).

Bromley, Jane, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah (1993). "Signature Verification Using a "Siamese" Time Delay Neural Network." In: *Proceedings of the 6th International Conference on Neural Information Processing Systems*. NIPS'93. Morgan Kaufmann Publishers Inc., pp. 737–744 (cit. on pp. 32, 78).

Brown, Peter N, Robert D Falgout, and Jim E Jones (2000). "Semicoarsening Multigrid on Distributed Memory Machines." In: *SIAM Journal on Scientific Computing* 21.5, pp. 1823–1834 (cit. on p. 143).

Bruckner, S. and T. Moeller (2010). "Result-Driven Exploration of Simulation Parameter Spaces for Visual Effects Design." In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (6), pp. 1468–1476 (cit. on p. 72).

Bruder, Valentin (2022). "Performance Quantification of Visualization Systems." In: (cit. on p. 143).

Bruder, Valentin, Steffen Frey, and Thomas Ertl (2016). "Real-Time Performance Prediction and Tuning for Interactive Volume Raycasting." In: *Proceedings SIGGRAPH ASIA 2016 Symposium on Visualization*, 7:1–7:8 (cit. on p. 158).

Bruder, Valentin, Steffen Frey, and Thomas Ertl (2017). "Prediction-Based Load Balancing and Resolution Tuning for Interactive Volume Raycasting." In: *Visual Informatics* 1.2, pp. 106–117 (cit. on p. 144).

Bruder, Valentin, Christoph Müller, Steffen Frey, and Thomas Ertl (2020). "On Evaluating Runtime Performance of Interactive Visualizations." In: *IEEE Transactions on Visualization and Computer Graphics* 26.9, pp. 2848–2862 (cit. on p. 143).

Bylinskii, Zoya et al. (2017). "Learning Visual Importance for Graphic Designs and Data Visualizations." In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST '17. Association for Computing Machinery, pp. 57–69 (cit. on p. 161).

Chapelle, Olivier, Bernhard Schölkopf, and Alexander Zien, eds. (2006). *Semi-Supervised Learning*. Red. by Francis Bach. Adaptive Computation and Machine Learning Series. MIT Press. 528 pp. (cit. on p. 29).

Chapelle, Olivier, Jason Weston, and Bernhard Schölkopf (2003). "Cluster Kernels for Semi-Supervised Learning." In: *Advances in neural information processing systems*, pp. 601–608 (cit. on p. 29).

Chen, M. and H. Jaenicke (2010). "An Information-theoretic Framework for Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (6), pp. 1206–1215 (cit. on p. 37).

Chen, Ting, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton (2020). *A Simple Framework for Contrastive Learning of Visual Representations*. URL: http://arxiv.org/abs/2002.05709 (cit. on pp. 33, 112, 117).

Cheng, Hsueh-Chien, Antonio Cardone, Somay Jain, Eric Krokos, Kedar Narayan, Sriram Subramaniam, and Amitabh Varshney (2019). "Deep-Learning-Assisted Volume Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 25.2, pp. 1378–1391.

Chopra, S., R. Hadsell, and Y. LeCun (2005). "Learning a Similarity Metric Discriminatively, with Application to Face Verification." In: *2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1, pp. 539–546 (cit. on p. 78).

Cui, Weiwei et al. (2020). "Text-to-Viz: Automatic Generation of Infographics from Proportion-Related Natural Language Statements." In: *IEEE Transactions on Visualization and Computer Graphics* 26.1, pp. 906–916 (cit. on p. 161).

Cuturi, Marco (2013). "Sinkhorn Distances: Lightspeed Computation of Optimal Transport." In: *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., pp. 2292–2300 (cit. on p. 96).

Daiber, Florian, Eric Falk, and Antonio Krüger (2012). "Balloon Selection Revisited: Multi-Touch Selection Techniques for Stereoscopic Data." In: *Proceedings of the International Working Conference on Advanced Visual Interfaces*. AVI '12. Association for Computing Machinery, pp. 441–444 (cit. on p. 111).

Dasgupta, Soumik Ranjan (2019). *Goodreads-Books*. 2021. URL: https://kaggle.com/jealousleopard/goodreadsbooks (cit. on p. 123).

Deng, Jiajun, Yingwei Pan, Ting Yao, Wengang Zhou, Houqiang Li, and Tao Mei (2019). "Relation Distillation Networks for Video Object Detection." In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7022–7031 (cit. on p. 73).

Dibia, Victor and Çağatay Demiralp (2019). "Data2Vis: Automatic Generation of Data Visualizations Using Sequence-to-Sequence Recurrent Neural Networks." In: *IEEE Computer Graphics and Applications* 39.5, pp. 33–46 (cit. on pp. 109, 160).

Doersch, Carl, Abhinav Gupta, and Alexei A. Efros (2015). "Unsupervised Visual Representation Learning by Context Prediction." In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1422–1430 (cit. on pp. 31, 73, 112).

Doersch, Carl and Andrew Zisserman (2017). *Multi-Task Self-Supervised Visual Learning*. URL: http://arxiv.org/abs/1708.07860 (cit. on pp. 31, 73, 78, 112, 169).

Dosovitskiy, Alexey, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox (2014). "Discriminative Unsupervised Feature Learning with Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., pp. 766–774 (cit. on pp. 73, 112).

Dutta, Soumya and Han-Wei Shen (2016). "Distribution Driven Extraction and Tracking of Features for Time-varying Data Analysis." In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (1), pp. 837–846 (cit. on p. 36).

Endert, A., W. Ribarsky, C. Turkay, B.L. William Wong, I. Nabney, I. Díaz Blanco, and F. Rossi (2017). "The State of the Art in Integrating Machine Learning into Visual Analytics." In: *Computer Graphics Forum* 36.8 (cit. on p. 37).

Engel, Dominik and Timo Ropinski (2021). "Deep Volumetric Ambient Occlusion." In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2), pp. 1268–1278 (cit. on p. 161).

Escobar, R. and R. V. Boppana (2016). "Performance Prediction of Parallel Applications Based on Small-Scale Executions." In: *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, pp. 362–371 (cit. on p. 143).

Fan, Chaoran and Helwig Hauser (2018). "Fast and Accurate CNN-based Brushing in Scatterplots." In: *Computer Graphics Forum* 37.3, pp. 111–120 (cit. on p. 161).

Fang, Zhe, Torsten Möller, Ghassan Hamarneh, and Anna Celler (2007). "Visualization and Exploration of Time-varying Medical Image Data Sets." In: *Proceedings of Graphics Interface 2007*. GI '07. ACM, pp. 281–288 (cit. on p. 36).

Ferstl, Florian, Mathias Kanzler, Marc Rautenhaus, and Rüdiger Westermann (2016). "Visual Analysis of Spatial Variability and Global Correlations in Ensembles of Iso-Contours." In: *Computer graphics forum* 35.3 (3), pp. 221–230 (cit. on p. 12).

Flamary, R'emi and Nicolas Courty (2017). *POT Python Optimal Transport Library*. URL: `https://github.com/rflamary/POT` (cit. on p. 96).

Fofonov, A. and L. Linsen (2019). "Projected Field Similarity for Comparative Visualization of Multi-Run Multi-Field Time-Varying Spatial Data." In: *Computer Graphics Forum* 38.1 (1), pp. 286–299 (cit. on p. 73).

Fogal, Thomas, Hank Childs, Siddharth Shankar, Jens Krüger, R Daniel Bergeron, and Philip Hatcher (2010). "Large Data Visualization on Distributed Memory Multi-GPU Clusters." In: *Proceedings of the Conference on High Performance Graphics*. Eurographics Association, pp. 57–66 (cit. on p. 143).

Frey, Steffen (2017). "Sampling and Estimation of Pairwise Similarity in Spatio-Temporal Data Based on Neural Networks." In: *Informatics* 4.3 (3), p. 27 (cit. on p. 37).

Frey, Steffen and Thomas Ertl (2017a). "Flow-Based Temporal Selection for Interactive Volume Visualization." In: *Computer Graphics Forum* 36.8 (8), pp. 153–165 (cit. on pp. 36, 57, 72).

Frey, Steffen and Thomas Ertl (2017b). "Progressive Direct Volume-to-Volume Transformation." In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (1), pp. 921–930 (cit. on pp. 36, 72).

Frey, Steffen, Filip Sadlo, and Thomas Ertl (2012). "Visualization of Temporal Similarity in Field Data." In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (12), pp. 2023–2032 (cit. on p. 36).

Frohn, Olaf (2017). *D3 Celestial*. URL: https://github.com/ofrohn/d3-celestial (cit. on p. 129).

Fuchs, R., J. Waser, and M. E. Groller (2009). "Visual Human+Machine Learning." In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (6), pp. 1327–1334 (cit. on p. 37).

Gadirov, Hamid (2020). "Autoencoder-Based Feature Extraction for Ensemble Visualization." Master's thesis. University of Stuttgart (cit. on p. 105).

Gadirov, Hamid, **Gleb Tkachev**, Thomas Ertl, and Steffen Frey (2021). "Evaluation and Selection of Autoencoders for Expressive Dimensionality Reduction of Spatial Ensembles." In: *International Symposium on Visual Computing*. Springer, pp. 222–234 (cit. on pp. 6, 71, 105, 165, 170).

Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge (2015). *A Neural Algorithm of Artistic Style*. URL: http://arxiv.org/abs/1508.06576 (cit. on p. 120).

Geppert, A., A. Terzis, G. Lamanna, M. Marengo, and B. Weigand (2017). "A Benchmark Study for the Crown-Type Splashing Dynamics of One- and Two-Component Droplet Wall–Film Interactions." In: *Experiments in Fluids* 58.12 (12), p. 172 (cit. on pp. 88, 89).

Ghahremani, Parmida et al. (2021). "NeuroConstruct: 3D Reconstruction and Visualization of Neurites in Optical Microscopy Brain Images." In: *IEEE Transactions on Visualization and Computer Graphics* (Early Access).

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep Sparse Rectifier Neural Networks." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315–323 (cit. on p. 150).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press (cit. on pp. 29, 31, 46, 49).

Guo, Li et al. (2020a). "SSR-VFD: Spatial Super-Resolution for Vector Field Data Analysis and Visualization." In: *2020 IEEE Pacific Visualization Symposium*, pp. 71–80 (cit. on p. 161).

Guo, Rongchen, Takanori Fujiwara, Yiran Li, Kelly M. Lima, Soman Sen, Nam K. Tran, and Kwan-Liu Ma (2020b). "Comparative Visual Analytics for Assessing Medical Records with Sequence Embedding." In: *Visual Informatics* 4.2, pp. 72–85 (cit. on p. 38).

Gygli, Michael (2017). *Ridiculously Fast Shot Boundary Detection with Fully Convolutional Neural Networks*. URL: http://arxiv.org/abs/1705.08214 (cit. on p. 38).

Han, Jun, Jun Tao, and Chaoli Wang (2018). "FlowNet: A Deep Learning Framework for Clustering and Selection of Streamlines and Stream Surfaces." In: *IEEE Transactions on Visualization and Computer Graphics* 26.4, pp. 1732–1744 (cit. on p. 38).

Han, Jun and Chaoli Wang (2020). "TSR-TVD: Temporal Super-Resolution for Time-Varying Data Analysis and Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (1), pp. 205–215 (cit. on pp. 37, 161).

Han, Jun and Chaoli Wang (2022). "VCNet: A Generative Model for Volume Completion." In: *Visual Informatics* (In Ppess).

Han, Jun, Hao Zheng, Danny Z. Chen, and Chaoli Wang (2021). "STNet: An End-to-End Generative Framework for Synthesizing Spatiotemporal Super-Resolution Volumes." In: *IEEE Transactions on Visualization and Computer Graphics* 28.1, pp. 270–280 (cit. on pp. 37, 170).

Han, Wei et al. (2016). *Seq-NMS for Video Object Detection*. URL: http://arxiv.org/abs/1602.08465 (cit. on p. 73).

Hao, Lihua, Christopher G. Healey, and Steffen A. Bass (2016). "Effective Visualization of Temporal Ensembles." In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (1), pp. 787–796 (cit. on p. 72).

Harrison, Lane, Katharina Reinecke, and Remco Chang (2015). "Infographic Aesthetics: Designing for the First Impression." In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI '15. Association for Computing Machinery, pp. 1187–1190 (cit. on p. 111).

Hartson, Rex and Pardha Pyla (2019). "Chapter 15 - Mental Models and Conceptual Design." In: *The UX Book (Second Edition)*. Ed. by Rex Hartson and Pardha Pyla. Morgan Kaufmann, pp. 327–340 (cit. on p. 111).

Hassanien, Ahmed, Mohamed Elgharib, Ahmed Selim, Sung-Ho Bae, Mohamed Hefeeda, and Wojciech Matusik (2017). *Large-Scale, Fast and Accurate Shot Boundary Detection through Spatio-temporal Convolutional Neural Networks*. URL: http://arxiv.org/abs/1705.03281 (cit. on p. 38).

Havre, S., E. Hetzler, P. Whitney, and L. Nowell (2002). "ThemeRiver: Visualizing Thematic Changes in Large Document Collections." In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (1), pp. 9–20 (cit. on p. 111).

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep Residual Learning for Image Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (cit. on p. 117).

He, Wenbin, Hanqi Guo, Han-Wei Shen, and Tom Peterka (2020a). "eFESTA: Ensemble Feature Exploration with Surface Density Estimates." In: *IEEE Transactions on Visualization and Computer Graphics* 26.4 (4), pp. 1716–1731 (cit. on p. 72).

He, Wenbin, Junpeng Wang, Hanqi Guo, Han-Wei Shen, and Tom Peterka (2020b). "CECAV-DNN: Collective Ensemble Comparison and Visualization Using Deep Neural Networks." In: *Vis. Informatics* 4.2 (2), pp. 109–121 (cit. on pp. 38, 159).

He, Wenbin et al. (2019). "InSituNet: Deep Image Synthesis for Parameter Space Exploration of Ensemble Simulations." In: *IEEE Transactions on Visualization and Computer Graphics* 26.1, pp. 23–33 (cit. on p. 38).

Heinemann, Moritz, Steffen Frey, **Gleb Tkachev**, Alexander Straub, Filip Sadlo, and Thomas Ertl (2021). "Visual Analysis of Droplet Dynamics in Large-Scale Multiphase Spray Simulations." In: *Journal of Visualization* 24.5, pp. 943–961 (cit. on p. 7).

Higgins, Irina et al. (2017). "Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 106, 170).

Hoefler, Torsten, William Gropp, William Kramer, and Marc Snir (2011). "Performance Modeling for Systematic Performance Tuning." In: *State of the Practice Reports*. SC '11, 6:1–6:12 (cit. on p. 144).

Hong, Fan, Can Liu, and Xiaoru Yuan (2019). "DNN-VolVis: Interactive Volume Visualization Supported by Deep Neural Network." In: *2019 IEEE Pacific Visualization Symposium*, pp. 282–291 (cit. on p. 37).

Honnibal, Matthew and Ines Montani (2017). *spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing*. URL: `https://spacy.io/` (cit. on p. 116).

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). "Multilayer Feedforward Networks Are Universal Approximators." In: *Neural Networks* 2.5, pp. 359–366 (cit. on p. 24).

Howison, M., E. W. Bethel, and H. Childs (2012). "Hybrid Parallelism for Volume Rendering on Large-, Multi-, and Many-Core Systems." In: *IEEE Transactions on Visualization and Computer Graphics* 18.1, pp. 17–29 (cit. on p. 143).

Hu, Kevin Zeng, Michiel A. Bakker, Stephen Li, Tim Kraska, and César A. Hidalgo (2018). *VizML: A Machine Learning Approach to Visualization Recommendation.* (Cit. on p. 109).

Huang, Dandan et al. (2015). "Personal Visualization and Personal Visual Analytics." In: *IEEE Transactions on Visualization and Computer Graphics* 21.3 (3), pp. 420–433 (cit. on p. 110).

Huang, Qingqiu, Wentao Liu, and Dahua Lin (2018). "Person Search in Videos with One Portrait Through Visual and Temporal Links." In: *2018 European Conference on Computer Vision*, pp. 437–454 (cit. on p. 73).

Hummel, M., H. Obermaier, C. Garth, and K. I. Joy (2013). "Comparative Visual Analysis of Lagrangian Transport in CFD Ensembles." In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (12), pp. 2743–2752 (cit. on p. 72).

Ipek, Engin, Bronis R. De Supinski, Martin Schulz, and Sally A. McKee (2005). "An Approach to Performance Prediction for Parallel Applications." In: *European Conference on Parallel Processing*. Springer, pp. 196–205 (cit. on p. 143).

Isenberg, Petra et al. (2017). "Vispubdata.Org: A Metadata Collection About IEEE Visualization (VIS) Publications." In: *IEEE Transactions on Visualization and Computer Graphics* 23.9 (9), pp. 2199–2206 (cit. on p. 115).

Jakob, Jakob, Markus Gross, and Tobias Günther (2021). "A Fluid Flow Data Set for Machine Learning and Its Application to Neural Flow Map Interpolation." In: *IEEE Transactions on Visualization and Computer Graphics* 27.2, pp. 1279–1289 (cit. on p. 37).

Jarema, M., I. Demir, J. Kehrer, and R. Westermann (2015). "Comparative Visual Analysis of Vector Field Ensembles." In: *2015 IEEE Conference on Visual Analytics Science and Technology (VAST)*. 2015 IEEE Conference on Visual Analytics Science and Technology (VAST), pp. 81–88 (cit. on pp. 12, 72).

Jeffery, Clinton L. (2019). "The City Metaphor in Software Visualization." In: *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision WSCG 2019*, pp. 153–163 (cit. on p. 111).

Jerald, Jason, Joseph J. LaViola, and Richard Marks (2017). "VR Interactions." In: *ACM SIGGRAPH 2017 Courses*. SIGGRAPH '17. Association for Computing Machinery (cit. on p. 111).

Jiao, Licheng, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu (2019). "A Survey of Deep Learning-Based Object Detection." In: *IEEE Access* 7, pp. 128837–128868 (cit. on p. 73).

Jing, Longlong and Yingli Tian (2020). "Self-Supervised Visual Feature Learning with Deep Neural Networks: A Survey." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.11, pp. 4037–4058 (cit. on p. 31).

Jing, Yongcheng, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song (2020). "Neural Style Transfer: A Review." In: *IEEE Transactions on Visualization and Computer Graphics* 26.11 (11), pp. 3365–3385 (cit. on p. 112).

Jänicke, H., A. Wiebel, G. Scheuermann, and W. Kollmann (2007). "Multifield Visualization Using Local Statistical Complexity." In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (6), pp. 1384–1391 (cit. on p. 37).

Johnson, Christopher and Charles Hansen (2004). *Visualization Handbook*. Academic Press, Inc. (cit. on p. 4).

Joshi, A. and P. Rheingans (2005). "Illustration-Inspired Techniques for Visualizing Time-Varying Data." In: *IEEE Visualization 2005*, pp. 679–686 (cit. on p. 35).

Kaggle (2014). *Dogs vs Cats*. URL: `https://www.kaggle.com/c/dogs-vs-cats` (cit. on p. 117).

Karras, Tero, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila (2020). "Analyzing and Improving the Image Quality of StyleGAN." In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8107–8116 (cit. on p. 120).

Kehrer, Johannes and Helwig Hauser (2013). "Visualization and Visual Analysis of Multifaceted Scientific Data: A Survey." In: *IEEE Transactions on Visualization and Computer Graphics* 19.3 (3), pp. 495–513 (cit. on pp. 10, 13).

Kim, Byungsoo and Tobias Günther (2019). "Robust Reference Frame Extraction from Unsteady 2D Vector Fields with Convolutional Neural Networks." In: *Computer Graphics Forum* 38.3, pp. 285–295.

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization.* URL: http://arxiv.org/abs/1412.6980 (cit. on pp. 49, 83).

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). "Optimization by Simulated Annealing." In: *Science* 220.4598 (4598), pp. 671–680 (cit. on p. 115).

Klemm, Paul, Sylvia Saalfeld, Kai Lawonn, Marko Rak, Henry Völzke, Katrin Hegenscheid, and Bernhard Preim (2015). "Interactive Visual Analysis of Lumbar Back Pain What the Lumbar Spine Tells About Your Life." In: *IVAPP 2015 - 6th International Conference on Information Visualization Theory and Applications; VISIGRAPP, Proceedings*, pp. 85–92 (cit. on p. 37).

Koch, Gregory R. (2015). "Siamese Neural Networks for One-Shot Image Recognition." In: 2015 ICML Deep Learning Workshop (cit. on pp. 32, 80).

Kohonen, T. (1990). "The Self-Organizing Map." In: *Proceedings of the IEEE* 78.9 (9), pp. 1464–1480 (cit. on p. 37).

Kolouri, Soheil, Phillip E. Pope, Charles E. Martin, and Gustavo K. Rohde (2019). "Sliced Wasserstein Auto-Encoders." In: *International Conference on Learning Representations* (cit. on p. 106).

Koopmans, Tjalling C. and Martin Beckmann (1957). "Assignment Problems and the Location of Economic Activities." In: *Econometrica* 25.1 (1), pp. 53–76 (cit. on p. 115).

Kulkarni, Sanjeev R. and Gilbert Harman (2011). "Statistical Learning Theory: A Tutorial." In: *Wiley Interdisciplinary Reviews: Computational Statistics* 3.6 (6), pp. 543–556 (cit. on p. 42).

Kumpf, Alexander, Marc Rautenhaus, Michael Riemer, and Rüdiger Westermann (2019). "Visual Analysis of the Temporal Evolution of Ensemble Forecast Sensitivities." In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (1), pp. 98–108 (cit. on p. 72).

Lake, Brenden M., Ruslan Salakhutdinov, and Joshua B. Tenenbaum (2015). "Human-Level Concept Learning through Probabilistic Program Induction." In: *Science* 350.6266 (6266), pp. 1332–1338 (cit. on p. 33).

Lakoff, George (1980). *Metaphors We Live By.* University of Chicago Press (cit. on p. 110).

Lample, Guillaume, Alexis Conneau, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou (2018). "Word Translation without Parallel Data." In: International Conference on Learning Representations (cit. on p. 112).

Lapuschkin, Sebastian, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek (2015). "On Pixel-Wise Explanations for

Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation." In: *PLoS ONE* 10.7 (cit. on p. 170).

Larsen, Matthew, Cyrus Harrison, James Kress, David Pugmire, Jeremy S. Meredith, and Hank Childs (2016). "Performance Modeling of in Situ Rendering." In: *Proc. High Performance Computing, Networking, Storage and Analysis*. SC '16, 24:1–24:12 (cit. on p. 144).

Lasserre, Julia A, Christopher M Bishop, and Thomas P Minka (2006). "Principled Hybrids of Generative and Discriminative Models." In: *2006 IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 1. IEEE, pp. 87–94 (cit. on p. 29).

Lawler, Eugene L (1963). "The Quadratic Assignment Problem." In: *Management Science* 9.4 (4), pp. 586–599 (cit. on p. 115).

Lee, Benjamin C., David M. Brooks, Bronis R. de Supinski, Martin Schulz, Karan Singh, and Sally A. McKee (2007). "Methods of Inference and Learning for Performance Modeling of Parallel Applications." In: *Proc. SIGPLAN Principles and Practice of Parallel Programming*, pp. 249–258 (cit. on p. 143).

Lee, Teng-Yok and Han-Wei Shen (2009). "Visualizing Time-varying Features with TAC-based Distance Fields." In: *2009 IEEE Pacific Visualization Symposium*. IEEE Computer Society, pp. 1–8 (cit. on p. 36).

Leone, Stefano (2019). *IMDb Movies Extensive Dataset | Kaggle*. URL: `https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset/version/2` (cit. on p. 123).

Li, Wei, Rui Zhao, Tong Xiao, and Xiaogang Wang (2014). "DeepReID: Deep Filter Pairing Neural Network for Person Re-identification." In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 152–159 (cit. on p. 73).

Liu, L., D. Silver, K. Bemis, D. Kang, and E. Curchitser (2017). "Illustrative Visualization of Mesoscale Ocean Eddies." In: *Computer Graphics Forum* 36.3 (3), pp. 447–458 (cit. on p. 35).

Liu, Xiao, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang (2021). "Self-Supervised Learning: Generative or Contrastive." In: *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1 (cit. on p. 31).

Lowe, David G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints." In: *International Journal of Computer Vision* 60.2 (2), pp. 91–110 (cit. on p. 91).

Lu, A. and H. W. Shen (2008). "Interactive Storyboard for Overall Time-Varying Data Visualization." In: *2008 IEEE Pacific Visualization Symposium*. 2008 IEEE Pacific Visualization Symposium, pp. 143–150 (cit. on pp. 35, 57).

Lu, Y., K. Jiang, J. A. Levine, and M. Berger (2021). "Compressive Neural Representations of Volumetric Scalar Fields." In: *Computer Graphics Forum* 40.3, pp. 135–146 (cit. on pp. 37, 160).

Lun, Zhaoliang, Evangelos Kalogerakis, and Alla Sheffer (2015). "Elements of Style: Learning Perceptual Shape Style Similarity." In: *ACM Transactions on Graphics* 34.4 (4), 84:1–84:14 (cit. on p. 112).

Ma, K., J. Painter, C. Hansen, and M. Krogh (1994). "Parallel Volume Rendering Using Binary-Swap Compositing." In: *IEEE Computer Graphics and Applications* 14.4, pp. 59–68 (cit. on p. 16).

Ma, Kwan-Liu (2007). "Machine Learning to Boost the Next Generation of Visualization Technology." In: *IEEE Computer Graphics and Applications* 27.5 (5), pp. 6–9 (cit. on p. 37).

Mairal, Julien, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman (2008). *Supervised Dictionary Learning*. URL: https://arxiv.org/abs/0809.3083 (cit. on p. 29).

Makhzani, Alireza and Brendan J. Frey (2014). "K-Sparse Autoencoders." In: *2014 International Conference on Learning Representations*. Ed. by Yoshua Bengio and Yann LeCun (cit. on p. 170).

McInnes, Leland, John Healy, and James Melville (2018). *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. URL: http://arxiv.org/abs/1802.03426 (cit. on pp. 106, 116).

Mead, A. (1992). "Review of the Development of Multidimensional Scaling Methods." In: *Journal of the Royal Statistical Society* 41.1 (1), pp. 27–39 (cit. on p. 115).

Mendes, D., F. M. Caputo, A. Giachetti, A. Ferreira, and J. Jorge (2019). "A Survey on 3D Virtual Object Manipulation: From the Desktop to Immersive Virtual Environments." In: *Computer Graphics Forum* 38.1 (1), pp. 21–45 (cit. on p. 111).

Mikolov, Tomas, Quoc V. Le, and Ilya Sutskever (2013a). *Exploiting Similarities among Languages for Machine Translation*. URL: https://arxiv.org/abs/1309.4168 (cit. on p. 112).

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean (2013b). "Distributed Representations of Words and Phrases and Their Compositionality." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 3111–3119 (cit. on pp. 31, 115).

Misra, Ishan, C. Lawrence Zitnick, and Martial Hebert (2016). "Shuffle and Learn: Unsupervised Learning Using Temporal Order Verification." In: *2016 European Conference on Computer Vision*, pp. 527–544 (cit. on p. 73).

Mitchell, Tom M. (1997). *Machine Learning*. McGraw-Hill (cit. on p. 17).

Müller, C., M. Strengert, and T. Ertl (2006). "Optimized Volume Raycasting for Graphics-Hardware-Based Cluster Systems." In: *Proc. EGPGV*, pp. 59–66 (cit. on p. 143).

Molnar, Steven, Michael Cox, David Ellsworth, and Henry Fuchs (1994). "A Sorting Classification of Parallel Rendering." In: *IEEE Computer Graphics and Applications* 14.4, pp. 23–32 (cit. on pp. 14, 143).

Montúfar, Guido, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio (2014). *On the Number of Linear Regions of Deep Neural Networks*. URL: http://arxiv.org/abs/1402.1869 (cit. on pp. 25, 29).

Muelder, C. and K. L. Ma (2009). "Interactive Feature Extraction and Tracking by Utilizing Region Coherency." In: *2009 IEEE Pacific Visualization Symposium*. 2009 IEEE Pacific Visualization Symposium, pp. 17–24 (cit. on p. 36).

Mutlu, Belgin, Eduardo E. Veas, and Christoph Trattner (2016). "VizRec: Recommending Personalized Visualizations." In: *Ksii Transactions on Internet and Information Systems* 6.4 (4), p. 31 (cit. on p. 109).

Noroozi, Mehdi and Paolo Favaro (2016). "Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles." In: *European Conference on Computer Vision*. Springer, pp. 69–84 (cit. on p. 31).

Obermaier, H. and K. I. Joy (2014). "Future Challenges for Ensemble Visualization." In: *IEEE Computer Graphics and Applications* 34.3 (3), pp. 8–11 (cit. on p. 72).

Olah, Chris, Alexander Mordvintsev, and Ludwig Schubert (2017). "Feature Visualization." In: *Distill* 2.11 (11), e7 (cit. on p. 170).

Pan, Sinno Jialin and Qiang Yang (2010). "A Survey on Transfer Learning." In: *IEEE Transactions on Knowledge and Data Engineering* 22.10, pp. 1345–1359 (cit. on p. 29).

Pathak, Deepak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros (2016). "Context Encoders: Feature Learning by Inpainting." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2536–2544 (cit. on p. 31).

Peterka, Tom, Hongfeng Yu, Robert B Ross, Kwan-Liu Ma, et al. (2008). "Parallel Volume Rendering on the IBM Blue Gene/P." In: *Proc. EGPGV*, pp. 73–80 (cit. on p. 143).

Petitet, Antoine (2004). *HPL – A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. URL: http://www.netlib.org/benchmark/hpl/ (cit. on p. 144).

Potter, Kristin, Andrew Wilson, Peer-Timo Bremer, Dean Williams, Charles Doutriaux, Valerio Pascucci, and Chris R. Johnson (2009). "Ensemble-Vis: A Framework for the Statistical Visualization of Ensemble Data." In: *In Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, pp. 233–240 (cit. on p. 72).

Pousman, Zachary, John Stasko, and Michael Mateas (2007). "Casual Information Visualization: Depictions of Data in Everyday Life." In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (6), pp. 1145–1152 (cit. on p. 109).

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. (2019). "Language Models Are Unsupervised Multitask Learners." In: *OpenAI blog* 1.8, p. 9 (cit. on p. 31).

Richards, Stephen, Philip Barker, Ashok Banerji, Charles Lamont, and Karim Manji (2009). "The Use of Metaphors in Iconic Interface Design." In: *Digital Creativity (Intelligent Tutoring Media)* 5.2, pp. 73–80 (cit. on p. 111).

Rizzi, Silvio, Mark Hereld, Joseph A. Insley, Michael E. Papka, Thomas D. Uram, and Venkatram Vishwanath (2014). "Performance Modeling of Vl3 Volume Rendering on GPU-Based Clusters." In: *Proc. EGPGV*, pp. 65–72 (cit. on p. 144).

Ruder, Sebastian, Ivan Vulić, and Anders Søgaard (2019). "A Survey of Cross-lingual Word Embedding Models." In: *Journal of Artificial Intelligence Research* 65, pp. 569–631 (cit. on p. 112).

Sacha, Dominik, Matthias Kraus, Jürgen Bernard, Michael Behrisch, Tobias Schreck, Yuki Asano, and Daniel A. Keim (2018). "SOMFlow: Guided Exploratory Cluster Analysis with Self-Organizing Maps and Analytic Provenance." In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (1), pp. 120–130 (cit. on p. 37).

Sanyal, J., S. Zhang, J. Dyer, A. Mercer, P. Amburn, and R. Moorhead (2010). "Noodles: A Tool for Visualization of Numerical Weather Model Ensemble Uncertainty." In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (6), pp. 1421–1430 (cit. on p. 72).

Schroff, Florian, Dmitry Kalenichenko, and James Philbin (2015). "FaceNet: A Unified Embedding for Face Recognition and Clustering." In: 2015 Conference on Computer Vision and Pattern Recognition, pp. 815–823 (cit. on p. 32).

Sedlmair, M., C. Heinzl, S. Bruckner, H. Piringer, and T. Möller (2014). "Visual Parameter Space Analysis: A Conceptual Framework." In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (12), pp. 2161–2170 (cit. on pp. 13, 72).

Selim, Ahmed, Mohamed Elgharib, and Linda Doyle (2016). "Painting Style Transfer for Head Portraits Using Convolutional Neural Networks." In: *ACM Transactions on Graphics* 35.4 (4), 129:1–129:18 (cit. on p. 139).

Shi, Neng, Jiayi Xu, Skylar W. Wurster, Hanqi Guo, Jonathan Woodring, Luke P. Van Roekel, and Han-Wei Shen (2022). "GNN-Surrogate: A Hierarchical and Adaptive Graph Neural Network for Parameter Space Exploration of Unstructured-Mesh Ocean Simulations." In: *IEEE Transactions on Visualization and Computer Graphics (Early Access)* (Early Access) (cit. on pp. 37, 160).

Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 97, 120).

Singh, Karan, Engin Ipek, Sally A. McKee, Bronis R. de Supinski, Martin Schulz, and Rich Caruana (2007). "Predicting Parallel Application Performance via Machine Learning Approaches." In: *Concurrency and Computation* 19.17, pp. 2219–2235 (cit. on p. 143).

Smith, Samuel L., David H. P. Turban, Steven Hamblin, and Nils Y. Hammerla (2016). "Offline Bilingual Word Vectors, Orthogonal Transformations and the Inverted Softmax." In: 2017 International Conference on Learning Representations (cit. on p. 112).

Sodhi, Sukhdeep, Jaspal Subhlok, and Qiang Xu (2008). "Performance Prediction with Skeletons." In: *Cluster Computing* 11.2, pp. 151–165 (cit. on p. 143).

Song, Peng, Wooi Boon Goh, William Hutama, Chi-Wing Fu, and Xiaopei Liu (2012). "A Handle Bar Metaphor for Virtual Object Manipulation with Mid-Air Interaction." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. Association for Computing Machinery, pp. 1297–1306 (cit. on p. 111).

Srujan, Suddala (2020). *Art for Generative Modelling*. URL: https://www.kaggle.com/suddalasrujan/art-for-generative-modelling-images/version/3 (cit. on p. 120).

Sutherland, Ivan E., Robert F. Sproull, and Robert A. Schumacker (1974). "A Characterization of Ten Hidden-Surface Algorithms." In: *ACM Computing Surveys* 6.1, pp. 1–55 (cit. on p. 14).

Sutton, Richard (2019). *The Bitter Lesson*. URL: http://www.incompleteideas.net/IncIdeas/BitterLesson.html (cit. on pp. 2, 26).

Tabiai, Ilyass, **Gleb Tkachev**, Patrick Diehl, Steffen Frey, Thomas Ertl, Daniel Therriault, and Martin Lévesque (2019b). *Hybrid Image Processing Approach for Autonomous Crack Area Detection and Tracking Using Local Digital Image Correlation Results Applied to Single-Fiber Interfacial Debonding*. Version 1.0.0. URL: https://doi.org/10.5281/zenodo.2566394 (cit. on p. 7).

Tabiai, Ilyass, **Gleb Tkachev**, Patrick Diehl, Steffen Frey, Thomas Ertl, Daniel Therriault, and Martin Lévesque (2019a). "Hybrid Image Processing Approach for Autonomous Crack Area Detection and Tracking Using Local Digital Image Correlation Results Applied to Single-Fiber Interfacial Debonding." In: *Engineering Fracture Mechanics* 216, p. 106485 (cit. on p. 7).

Thompson, Neil C., Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso (2020). *The Computational Limits of Deep Learning*. URL: http://arxiv.org/abs/2007.05558 (cit. on p. 26).

Tkachev, **Gleb** (2017). "Investigation and Prediction of Distributed Volume Rendering Performance." Master's thesis. University of Stuttgart (cit. on pp. 142, 143).

Tkachev, **Gleb** (2022). *PyPlant: A Python Framework for Cached Function Pipelines*. URL: https://doi.org/10.18419/darus-2249 (cit. on p. 7).

Tkachev, **Gleb**, Rene Cutura, Michael Sedlmair, Steffen Frey, and Thomas Ertl (2022). "Metaphorical Visualization: Mapping Data to Familiar Concepts." In: *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI EA '22. Association for Computing Machinery, DOI: 10.1145/3491101.3516393 (cit. on pp. 6, 109).

Tkachev, **Gleb**, Steffen Frey, and Thomas Ertl (2021a). "Local Prediction Models for Spatiotemporal Volume Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 27.7, pp. 3091–3108 (cit. on pp. 5, 35, 71, 168).

Tkachev, **Gleb**, Steffen Frey, and Thomas Ertl (2021b). *Replication Data for: "S4: Self-supervised Learning of Spatiotemporal Similarity"*. URL: https://doi.org/10.18419/darus-2174 (cit. on p. 7).

Tkachev, **Gleb**, Steffen Frey, and Thomas Ertl (2021c). "S4: Self-supervised Learning of Spatiotemporal Similarity." In: *IEEE Transactions on Visualization and Computer Graphics* DOI: 10.1109/TVCG.2021.3101418 (cit. on pp. 5, 7, 71, 168).

Tkachev, **Gleb**, Steffen Frey, Christoph Müller, Valentin Bruder, and Thomas Ertl (2017). "Prediction of Distributed Volume Visualization Performance to Support Render Hardware Acquisition." In: *Proceedings of the 17th Eurographics Symposium on Parallel Graphics and Visualization*. EGPGV '17. Eurographics Association, pp. 11–20 (cit. on pp. 6, 141).

Tong, Xin, Teng-Yok Lee, and Han-Wei Shen (2012). "Salient Time Steps Selection from Large Scale Time-Varying Data Sets with Dynamic Time Warping." In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE Symposium on Large Data Analysis and Visualization (LDAV), pp. 49–56 (cit. on pp. 35, 36, 57).

Tzeng, Fan-Yin and Kwan-Liu Ma (2005). "Intelligent Feature Extraction and Tracking for Visualizing Large-Scale 4D Flow Simulations." In: *Proceedings of the ACM/IEEE SC 2005 Conference*. Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference, pp. 6–6 (cit. on p. 36).

Vapnik, V. N. and A. Ya. Chervonenkis (1969). "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities." In: *Theory of Probability & Its Applications* 16.2 (2), pp. 264–280 (cit. on p. 46).

Velten, Andreas et al. (2013). "Femto-Photography: Capturing and Visualizing the Propagation of Light." In: *ACM Transactions on Graphics* 32.4 (4), 44:1–44:8 (cit. on pp. 52, 54).

Viegas, Fernanda B., Martin Wattenberg, and Jonathan Feinberg (2009). "Participatory Visualization with Wordle." In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (6), pp. 1137–1144 (cit. on p. 110).

Vig, Jesse, Shilad Sen, and John Riedl (2012). "The Tag Genome: Encoding Community Knowledge to Support Novel Interaction." In: *ACM Transactions on Interactive Intelligent Systems* 2.3 (3), 13:1–13:44 (cit. on p. 129).

Vincent, Pascal, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol (2008). "Extracting and Composing Robust Features with Denoising Autoencoders." In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Association for Computing Machinery, pp. 1096–1103 (cit. on p. 31).

Viola, I., M. Feixas, M. Sbert, and M. E. Groller (2006). "Importance-Driven Focus of Attention." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (5), pp. 933–940 (cit. on p. 37).

Wang, C., H. Yu, and K. L. Ma (2008). "Importance-Driven Time-Varying Data Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (6), pp. 1547–1554 (cit. on p. 36).

Wang, J., S. Hazarika, C. Li, and H. Shen (2019). "Visualization and Visual Analysis of Ensemble Data: A Survey." In: *IEEE Transactions on Visualization and Computer Graphics* 25.9 (9), pp. 2853–2872 (cit. on pp. 10, 11, 13, 72, 91).

Wang, Jiang et al. (2014). "Learning Fine-Grained Image Similarity with Deep Ranking." In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1386–1393 (cit. on p. 72).

Wang, Yifan, Zichun Zhong, and Jing Hua (2020). "DeepOrganNet: On-the-Fly Reconstruction and Visualization of 3D / 4D Lung Models from Single-View Projections by Deep Deformation Network." In: *IEEE Transactions on Visualization and Computer Graphics* 26.1, pp. 960–970.

Wang, Yunhai, Wei Chen, Jian Zhang, Tingxing Dong, Guihua Shan, and Xuebin Chi (2011). "Efficient Volume Exploration Using the Gaussian Mixture Model." In: *IEEE Transactions on Visualization and Computer Graphics* 17.11 (11), pp. 1560–1573 (cit. on p. 37).

Wang, Z., H. P. Seidel, and T. Weinkauf (2016). "Multi-Field Pattern Matching Based on Sparse Feature Sampling." In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (1), pp. 807–816 (cit. on pp. 72, 93, 94).

Waser, J., R. Fuchs, H. Ribicic, B. Schindler, G. Bloschl, and E. Groller (2010). "World Lines." In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (6), pp. 1458–1467 (cit. on p. 72).

Wei, Tzu-Hsuan, Chun-Ming Chen, Jonathan Woodring, HuiJie Zhang, and Han-Wei Shen (2017). "Efficient Distribution-Based Feature Search in Multi-Field Datasets." In: *2009 IEEE Pacific Visualization Symposium*, pp. 121–130 (cit. on p. 72).

Wolpert, D.H. and W.G. Macready (1997). "No Free Lunch Theorems for Optimization." In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82 (cit. on p. 21).

Woodring, Jonathan and Han-Wei Shen (2003). *Chronovolumes: A Direct Rendering Technique for Visualizing Time-Varying Data*. The Eurographics Association (cit. on p. 36).

Wu, G., J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou (2015). "GPGPU Performance and Power Estimation Using Machine Learning." In: *Proc. HPCA*, pp. 564–576 (cit. on p. 144).

Ynnerman, Anders, Jonas Löwgren, and Lena Tibell (2018). "Exploranation: A New Science Communication Paradigm." In: *IEEE Computer Graphics and Applications* 38.3 (3), pp. 13–20 (cit. on p. 110).

Yu, Hongfeng, Chaoli Wang, and Kwan-Liu Ma (2008). "Massively Parallel Volume Rendering Using 2–3 Swap Image Compositing." In: *Proc. High Performance Computing, Networking, Storage and Analysis*, pp. 1–11 (cit. on pp. 15, 16, 145, 151).

Zhang, Richard, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang (2018). "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric." In: *2018*

IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 586–595 (cit. on pp. 97, 139).

Zhang, Yao and John D Owens (2011). "A Quantitative Performance Analysis Model for GPU Architectures." In: *Proc. HPCA*, pp. 382–393 (cit. on p. 144).

Zheng, Hao, Jun Han, Hongxiao Wang, Lin Yang, Zhuo Zhao, Chaoli Wang, and Danny Z. Chen (2021). "Hierarchical Self-supervised Learning for Medical Image Segmentation Based on Multi-domain Data Aggregation." In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021: 24th International Conference, Strasbourg, France, September 27–October 1, 2021, Proceedings, Part I*. Springer-Verlag, pp. 622–632 (cit. on p. 38).

Zhou, Zhenglei, Yule Hou, Qirui Wang, Guangxiang Chen, Jiawei Lu, Yubo Tao, and Hai Lin (2017). "Volume Upscaling with Convolutional Neural Networks." In: *Proceedings of the Computer Graphics International Conference* (Yokohama, Japan). CGI '17. ACM, 38:1–38:6 (cit. on p. 37).

Zhuang, Fuzhen et al. (2021). "A Comprehensive Survey on Transfer Learning." In: *Proceedings of the IEEE* 109.1, pp. 43–76 (cit. on p. 29).

Ziemkiewicz, Caroline and Robert Kosara (2008). "The Shaping of Information by Visual Metaphors." In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (6), pp. 1269–1276 (cit. on p. 111).

Zou, Zhengxia, Zhenwei Shi, Yuhong Guo, and Jieping Ye (2019). *Object Detection in 20 Years: A Survey*. URL: http://arxiv.org/abs/1905.05055 (cit. on p. 73).