

Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelor's Thesis

EvalQuiz: Self-Assessment Generated through Language Transformer Models

Julian Kieslinger

Course of Study: Medieninformatik

Examiner: Prof. Dr.-Ing. Steffen Becker

Supervisor: Niklas Meißner, M.Sc.,
Sandro Speth, M.Sc.,

Commenced: April 20, 2023

Completed: October 25, 2023

Abstract

This thesis explores the constraints of self-assessment creation in higher education, focusing on the lack of tools, standardization, and time. We conduct didactic field expert interviews to investigate how teaching evolves towards learning management systems (LMS) with new possibilities for self-assessment generation. We propose EvalQuiz, a novel system that enables language model-assisted self-assessment creation. The system generates questions of different types, improving upon related work while addressing language model restrictions through filtering. EvalQuiz aims to bridge the gap between lecturers and students, enabling lecturers to work with educational objectives and lecture materials to provide student-centered self-assessment. The thesis aims to answer questions about the reliability and quality of self-assessment generation. EvalQuiz proposes a message composer scheme to reliably generate output according to a specification. The thesis defines a standardized self-assessment specification used with EvalQuiz. We conduct a survey evaluating EvalQuiz's real-world performance.

Kurzfassung

Die Bachelorarbeit untersucht Einschränkungen bei der Erstellung von Lernstandsquizze im universitären Umfeld, mit Fokus auf fehlenden Tools, Standardisierung und Zeit. Die Ausarbeitung umfasst Interviews mit Didaktikexperten um die Generierung von Lernstandsquizen und dessen Integration in Lernplattformen zu erörtern. Wir präsentieren EvalQuiz, ein System zur Erstellung von Lernstandsquizen mit Sprachmodellen. EvalQuiz kann Fragen verschiedener Typen generieren und verbessert somit vorherige Arbeiten. EvalQuiz filtert Materialien für die Nutzung mit Sprachmodellen. EvalQuiz strebt beiderseits die Unterstützung von Dozenten und Studenten an. Es ermöglicht Dozenten ihre Lernziele und Vorlesungsmaterialien mit EvalQuiz zu nutzen, um Lernquize zu erstellen, die relevant für die Studierenden sind. Die Arbeit beantwortet Fragen zur Zuverlässigkeit und Qualität der Erstellung von Lernstandsquizen mit Sprachmodellen. Die Arbeit beschreibt eine Methode um zuverlässig Daten bestimmter Datentypen mit Sprachmodellen zu erzeugen. Die Ausarbeitung standardisiert Datentypen von Fragetypen. Eine Studie liefert Einblicke in die Performance von EvalQuiz.

Acknowledgements

I want to thank my supervisors Niklas Meißner and Sandro Speth for guiding the thesis at every step. The project improved through their immediate and constructive feedback. Their experience in academic writing experience helped me shape the thesis structure and wording. Special thanks to Prof. Maria Wirzberger and Avni Qekaj for their time and interest. Their valuable insight and feedback influenced EvalQuiz concepts. Their perspective highlighted previously unconsidered aspects. I also want to thank Niklas Krieger for providing feedback on the UML architecture models and proofreading the thesis.

Contents

1	Introduction	1
2	Foundations	5
2.1	Self-Assessment	5
2.1.1	Bloom’s Taxonomy	5
2.1.2	Question Types	6
2.2	Transformer Architecture	9
2.3	Transformer Models	11
2.3.1	Transformer Models by OpenAI	11
2.3.2	BERT	13
2.3.3	T5	14
2.3.4	PaLM	14
2.3.5	Other Transformer Models	15
2.3.6	Transformer Model Steering	15
3	Related Work	17
3.1	Self-Assessment Generation	17
3.1.1	Methodology and Overview	17
3.1.2	Sentence Selection from Input Text	17
3.1.3	Multiple-Choice Generation	18
3.1.4	Question Generation with Transformer Models	19
3.2	Pipeline Reference Architecture	20
4	Requirements Engineering	21
4.1	Problems with Self-Assessment	21
4.2	Didactic Specialist Interviews	23
4.2.1	Procedure	23
4.2.2	Introduction of interaction prototype	23
4.2.3	Condensed Findings: Interviews	25
4.2.4	Collection of Further Project Ideas	27
4.3	Student Benefits Through Self-Assessment	27
4.4	Language Barriers	28
4.5	Visual Materials	28
4.6	EvalQuiz Self-Assessment System: Requirements Summary	28
4.6.1	Functional Requirements	28
4.6.2	Nonfunctional Requirements	30
5	Concept	31
5.1	Pipeline modules and data	31

5.2	Matching/Converting Model Output to Specification	33
5.2.1	Multi Prompt Self-Assessment Creation: Concept	34
5.2.2	Single Prompt Self-Assessment Creation: Prototype	35
5.3	Model Steering	35
5.3.1	Topic Model Enabled Keyword Augmentation	37
5.3.2	From Keywords to Capabilities	37
5.4	The Transformer Models Role	38
5.5	Specification of Self-Assessment Question Types	38
5.5.1	Monolithic Design of Question Specification	39
5.5.2	Question Specification Responsibilities and Usage	40
6	Architecture and Implementation	41
6.1	Architecture Modelling	42
6.1.1	Material Server Architecture	43
6.1.2	Pipeline Server Architecture	43
6.1.3	CLI Client Architecture	44
6.2	Development Tools	48
6.3	VSCode Development Containers	51
6.4	Server Interfaces	51
6.5	Material Server	51
6.5.1	Content-Addressable Storage	52
6.5.2	Hash function: BLAKE3 [Webf]	52
6.6	Pipeline Server	52
6.6.1	Material Filter Pipeline Module	53
6.6.2	Question Generation Pipeline Module	57
6.6.3	Question Evaluation Pipeline Module	59
6.6.4	Question Specification Changes	66
6.7	Frontend	66
6.7.1	gRPC Support	67
6.7.2	Custom JSONForms Components	67
6.7.3	Protobuf to JSONSchema Conversion	68
6.8	Deployment and Networking	68
6.8.1	Multistage Builds	69
6.8.2	Cloud Computing Platform: Microsoft Azure	69
6.8.3	Continuous Integration (CI)	69
6.8.4	Continuous Deployment (CD)	71
7	Evaluation	73
7.1	Study Design	73
7.1.1	Usage Example: Summarize As Self-Assessment	74
7.1.2	Usage Example: Find Connections Between A And B	74
7.1.3	Usage Example: Partial Regeneration	75
7.1.4	Questionnaire Structure	76
7.2	Results	77
7.3	Discussion	79
7.4	Threats to Validity	80

8	Conclusion and Future Work	81
8.1	Summary	81
8.2	Benefits	81
8.3	Limitations	82
8.4	Lessons Learned	82
8.5	Future Work	82
	Bibliography	85
A	Didactic Specialist Interviews	93
A.0.1	General Self-Assessment Interview	93
A.0.2	Interaction Prototype Interview	95
B	Question Specification	97
C	Implementation Libraries and Repositories	103

List of Figures

2.1	Transformer architecture [VSP+17].	10
2.2	Token calculation from a sentence with OpenAI's tiktoken tokenizer for GPT-3.5 & GPT-4 [Webai].	12
2.3	OpenAI: Reinforcement learning through human feedback [Webaj].	13
2.4	OpenAI: Language model alignment [Webaj].	13
4.1	Self-Assessment: Student Benefits.	21
4.2	Self-Assessment: Required Domain Experts.	22
4.3	Self-Assessment Creation: Bottlenecks.	22
4.4	Lecturer interaction prototype: Sequence diagram.	26
5.1	EvalQuiz Self-Assessment Concept Overview.	31
5.2	Processing pipeline.	32
5.3	Iterative pipeline interaction.	33
6.1	Architecture planning UML class diagram: Material server overview.	43
6.2	Architecture planning UML class diagram: Pipeline server overview.	44
6.3	Architecture planning UML class diagram: Pipeline server I	45
6.4	Architecture planning UML class diagram: Pipeline server II	46
6.5	Architecture planning UML class diagram: Pipeline server III	47
6.6	Architecture planning UML class diagram: CLI client overview.	48
6.7	Architecture planning UML class diagram: CLI client IV	49
6.8	Architecture planning UML class diagram: CLI client V	49
6.9	Architecture planning UML class diagram: CLI client VI	50
6.10	BLAKE3 benchmark against state-of-the-art cryptographic hashes: Source: O'Connor et al. [Webf].	53
6.11	Mathpix conversion text: Source PDF: James et al. [JWHT+13].	55
6.12	Mathpix conversion text: Compiled \LaTeX from converted PDF, source: James et al. [JWHT+13].	55
6.13	Comparison of original PPTX and converted markdown.	56
6.14	EvalQuiz frontend: Configuration question specification subset.	67
6.15	EvalQuiz frontend: Material suggestion component.	68
6.16	EvalQuiz full stack deployment: Docker compose configuration (YAML) visualized with PlantUML [Webao].	70
7.1	Survey results: Self-assessment that the participants have experience working with.	77
7.2	Survey results: Self-assessment constraints.	78
7.3	Survey results: Beneficial features for question generation.	79

List of Tables

2.1	Question types and their answer space sizes.	9
2.2	GPT-3.5 variants (directly cited) [Webs].	12
6.1	Structure of messages created by Message Composer.	58
6.2	Evaluation prototype test: One-shot example.	63
6.3	Evaluation prototype test: Zero-shot example.	65

List of Algorithms

2.1	Automated evaluation of an unambiguous question.	8
5.1	Material Filter.	36

1 Introduction

Self-assessment creation is challenging and can be laborious. It is not easy to come up with quizzes that are relevant, interesting and of the right difficulty level. University teaching focuses on student self-assessment too little, despite its advantages for students [And19; BB95; CB17; SSS21]. Researchers are often responsible for both research and teaching, while research alone occupies a large portion of time [Kar12; MM05]. Publications predominantly measure academic success [BPH+20], which incentivizes to neglect teaching. Researchers use the available time for incrementally improving lectures and exercises. Creating and offering self-assessment requires additional time. Tools and standardization have unexplored potential in guiding researchers towards self-assessment, conveying which forms of self-assessment exist and assisting in self-assessment creation.

The educational landscape evolves towards remote teaching, opening new possibilities for self-assessment. The demand for learning management systems (LMS), as Ilias [Webv], and their respective features skyrocketed due to the pandemic. As the trend continues, institutes extend their support for online formats, as they experience efficiency benefits. Online lectures can be reused easily. A mistake in a traditional in-person lecture can remain unseen, while videos allow us to identify and fix mistakes easily. Students benefit from additional tools, for example, quizzes, to self-assess. The possibility to re-watch and pause sections is valuable for students.

Learning management systems allow to manually insert self-assessment in the form of quizzes [Webv]. The inserted data is LMS or plugin-specific. Sharing quizzes with other LMS and other universities requires laborious data conversion and in-depth communication with the other party. There is no standardized question specification for self-assessment to our knowledge. Related work of "question generation" presents approaches that are fixed to one specific question type [ARS+16; KM+06; KN19; MS15; NBM+22; TCY]. Most approaches focus on multiple-choice [ARS+16; KM+06; KN19; MS15; NBM+22; TCY], while there lies huge potential in more flexible self-assessment systems. Only a few newer approaches utilize large language models [KN19; NBM+22; TCY]. To our knowledge, none of the current approaches account for the know-how of the didactic field: Bloom's taxonomy [BK20], question construction guidelines [Unia] and motivational factors [Unic]. Publications describe technical challenges and regard implications for students to little [ARS+16; KM+06; KN19; MS15; NBM+22] except for Tsai et al. [TCY]. A common optimization criterion of question generation is correctness [MS15]. We argue that student benefit is a better factor for self-assessment creation to optimize.

Open questions for self-assessment creation are: How to incorporate accomplishments from didactic sciences in self-assessment generation? Can LMSs benefit from the capabilities that language models offer? Can language models reliably generate self-assessment according to a specification? How to teach a language model a specification? The thesis aims to answer the following research questions: **RQ1** Is the automatic generation of self-assessment quizzes including correct answers using GPT-4 reliable? **RQ2** Is the quality of GPT-generated self-assessment quizzes suited for higher

education courses in computer science, software engineering, and languages? **RQ3 (Optional)** Is GPT-4 capable of generating a larger variety of self-assessment questions when using (text + image) input instead of text-only input? **RQ4** Which approach withholds better results for specification matching: (GPT-4) few-shot learning or fine-tuning using a less-capable open source model? **RQ5** Are state-of-the-art language models capable of reliably producing output according to an API specification? **RQ6** Is keyword recommendation effectively assisting lecturers in the selection of subtopics for self-assessment questions?

We propose EvalQuiz: A system that tackles current self-assessment creation challenges. EvalQuiz generates self-assessment of a variety of question types. EvalQuiz is designed for LMS compatibility through a specification. EvalQuiz solves language model input restrictions through filtering. Novelties are the message composer scheme, question specification and educational objective-based question generation. The message composer scheme teaches the model to answer with a specific data type. EvalQuiz uses a novel way to connect educational objectives with self-assessment. This gives lecturers control over question generation depth. We propose the concept of Capabilities. Capabilities contain educational objectives that connect to questions. This association benefits LMSs in categorizing, recommending and sharing self-assessment. A contribution is the implementation of content-addressable lecture material management through the material server. The thesis explores how LMS can benefit from content addressable storage. The thesis also discusses EvalQuiz's role in student-centered self-assessment recommendations. Contributions of the thesis are conceptualizing, implementing, deploying and evaluating EvalQuiz. The system implementation consists of 7 individual services.

We evaluate the system in a survey targeted to lecturers of different domains. The remainder of the thesis is structured as follows:

Thesis Structure

Foundations gives an overview of key concepts such as self-assessment, question types and transformer models. We highlight aspects necessary for understanding later sections.

Related Work describes our search methodology for related work and the lack of current self-assessment creation solutions. This chapter highlights the lack of connection between the didactic field and question generation.

Requirements Engineering states current problems with self-assessment and explores solutions through didactic specialist interviews. We discuss an initial interaction prototype with experts in the didactic field and distill requirements from our findings.

Concept discusses technical challenges and possible solutions. We embed the requirements created in the RE process in our concept. We justify and explore data flow and data types.

Architecture and Implementation encompasses architecture modelling and design decisions. We explain which technologies we chose. We discuss implementation details of components and their function, as pipeline modules.

Evaluation summarizes the real-world evaluation of our system. The evaluation consists of (1) familiarizing the participant with the system and its concepts, (2) enabling the participant to test the system with their lecture materials and (3) a short questionnaire collecting experiences and remarks.

Conclusion and Future Work sums up the project and its findings and states open questions for future work. We suggest directions for future work to explore.

2 Foundations

This chapter explains the necessary foundations of self-assessment and transformer models. We discuss the transformer architecture and several proprietary and open-source implementations. We explain important terms used in this thesis.

2.1 Self-Assessment

Boud et al. define self-assessment as:

Definition 2.1.1 (Self-assessment)

The involvement of students in identifying standards and/or criteria to apply to their work and making judgments about the extent to which they have met these criteria and standards. [BB95]

A student can identify standards/criteria through the lecture itself. Either in a direct or an indirect way:

- A standard or criteria is explicitly stated by the lecturer as required
- A student assumes through former knowledge and experience. For example previous lectures

Lecturers should be encouraged to state standards/criteria. Students are unlikely to match the lecturer's expectations by assumption. If told, students can grasp the lecturer's expectations. Expected standards/criteria can be specified through Bloom's taxonomy [BK20]. Stated standard/s/criteria are then interpreted by students. Students have to judge to which extent they match those standards/criteria. This happens in two steps. (1) A student internalizes the standard/criteria and (2) estimates to what extent they meet the standard/criteria.

Remark

We use the word self-assessment for two concepts. (1) Self-assessment helps a student to self-assess, Boud et al. [BB95]. Self-assessment can be a self-assessment question, for example. And (2) Definition 2.2.1.

2.1.1 Bloom's Taxonomy

Bloom categorizes educational objectives into six hierarchical categories [BK20] [Unib]. The cognitive model, published as early as 1956 by Bloom:

1. **Knowledge:** Concepts can be recalled.

2. **Comprehension:** Concepts can be compared, discussed, explained, distinguished or summarized.
3. **Application:** Concepts can be applied to exercises: Calculation, classification and interpretation in known or unknown scenarios.
4. **Synthesis:** Concepts can be combined, developed, constructed or derived.
5. **Evaluation:** Concepts can be evaluated, criticized, explained or predicted.

The taxonomy orders educational objectives from simple to complex. Bloom's taxonomy helps to structure learning. Lecturers can provide exercises of the next level when the previous level is sufficiently assessed. The taxonomy allows one to check if exercises match the specified level. Bloom's taxonomy helps to specify to which depth students should internalize a concept. Students then can expect a specific set of exercises.

Each category includes all concepts of the categories below. We do not consider the **Knowledge** category in this thesis, as **Knowledge** alone can not be considered a valuable educational objective for higher education. Simply recalling concepts is not sufficient. The first category considered is **Comprehension**. **Comprehension** includes **Knowledge** due to the taxonomies hierarchical nature.

Bloom's taxonomy consists of three models for different domains [BK20]. The cognitive, affective and psychomotor domains. The affective and psychomotor domains are not considered in the thesis. We focus on the cognitive domain with its categories: **Comprehension, Application,**

2.1.2 Question Types

University lectures use a variety of question types in exercises and exams. Question types have different characteristics. Some are better to automatically evaluate. Exams designed with automatic evaluation in mind prefer unambiguous questions.

Definition 2.1.2 (Unambiguous question)

The question is correctly constructed according to the question type. A lecturer declares every answer option as either true or false.

Definition 2.1.3 (Answer option)

User input that conforms with the question type.

Definition 2.1.4 (Answer)

True answer option.

We need an example to clarify what an unambiguous question means:

We can define a question type: **True-or-false-question**, with a question text and two answer options: "true" and "false". The question text must be a closed question. The question type defines the answer options "true" and "false". A lecturer then labels answer option "true" → (as) "true" and option "false" → (as) "false". A question of question type **True-or-false-question** is always unambiguous. Each answer option is inherently declared as true or false.

Two of the most prominent question types enable unambiguous questions. Multiple-choice and multiple-response questions. We argue that multiple-choice and multiple-response questions are unambiguous.

Multiple-choice questions have $0..n$ fields and answer options. A lecturer declares one answer option as true and all other answer options as false. This is a solution to the multiple-choice question.

Multiple-response questions with $0..n$ fields have 2^n answer options. Each subset of n elements is an answer option. Each subset can be the answer. Only one subset is the answer. The order in which elements are chosen does not play a role. Fields should be independent of each other. A field changes its validity depending on surrounding fields, if dependent. A lecturer can correct a field in isolation, if independent.

We suggest reformulating multiple-response questions with dependent fields. For example:

Which of the following statements are true?	
- <input type="checkbox"/>	Statement A)
- <input type="checkbox"/>	Statement A) and B)
- <input type="checkbox"/>	Statement B)
- <input type="checkbox"/>	Statement C, but only if A)

Two refactoring options help solve answer option dependencies:

1. Change the question type from multiple-response to multiple-choice
2. Rewrite each field as pairwise independent:

Which of the following statements are true?	
- <input type="checkbox"/>	Statement A)
- <input type="checkbox"/>	Statement B)
- <input type="checkbox"/>	Statement C)

The original example is more complicated to evaluate than the refactored examples: A linear evaluation algorithm checks each answer option in order. The algorithm must have already checked the third field: Statement C), when it wants to check the second field: Statement A) and B). The second field references the third field and therefore creates a dependency. This causes linear evaluation to break for the given example. Removing references (refactoring option 2) fixes the problem. The algorithm does not need to follow any references. Evaluation is possible in n steps. No state needs to be carried over between field evaluations. Refactoring option 1 changes the question type to multiple-choice, reducing the answer options to n . This simplifies evaluation. Evaluating the original example (multiple-response) requires 2^n steps initially, as all of 2^n subsets have to be evaluated.

Question types with text input answers can be unambiguous. For example the **Diagram-cloze-question** type.

Definition 2.1.5 (Diagram-cloze-question)

Describes a graphic, for example from a lecture, with missing text fields. The task is to restore the missing text.

The original graphics' exact words can be an answer. However, only declaring the exact words as valid can render some answer options as false negatives. The student can write down a synonym. A lecturer has to consider synonyms as possible answers (valid answer options). The same holds for correct, but misspelled words. Similarly, diagrams with symmetries can have different valid answer option configurations.

This is also true for the **Free-text-question** type.

Definition 2.1.6 (Free-text-question)

Input to a single text field composes the answer. Students have to answer in their own words.

Exercises and exams contain **Free-text-questions**. Automatic scoring is more difficult for ambiguous question types than unambiguous ones. All text input field combinations are hard to label. Labeling all answers as either true or false in advance is impractical. The more answers, the harder it is to argue against the ambiguity of a question. Table 2.1 gives an overview of answer space sizes for different question types.

Self-assessment questions benefit from automatic evaluation. Manual correction by academic personnel is unfeasible. Academics can spend time on research instead. Automatic evaluation can give students immediate feedback.

Unambiguous questions allow a simple and automated evaluation:

Algorithm 2.1 Automated evaluation of an unambiguous question.

Question is unambiguous.

if User input in answer options **then**

 Return label "true" or "false" according to answer option.

else

 Invalid user input. Not conform with question type. Return "false".

end if

An ambiguous question requires a mechanism to handle unlabeled answer options. Large language models (LLM) are capable of automatic evaluation. GPT-4 performs respectably on academic exams [Ope23]. Cross-contamination between exams and pre-training data is low [Ope23]. Yet, performance does not live up to the standard needed for sample solutions.

Converting ambiguous to unambiguous questions can simplify automatic evaluation. Large language models are capable of converting one question type into another. Next, one applies Algorithm 2.1 on the resulting unambiguous question. Future work needs to assess the robustness of this idea.

Multiple-response and multiple-choice questions are the first question types implemented in EvalQuiz. We assess more question types and their ambiguity in the following. Implementation candidates for EvalQuiz:

Name	Description	Answer space size
Multiple-choice questions	Similar to multiple-response questions $0..n$ answer options exist of which 1 answer is true. This question type can be confused with Multiple-response questions.	n
Multiple-response questions	Consist of a sentence formulating a question, $0..n$ answer options of which $0..n$ answers can be true. Wrong answers are described as distractors [KM+06].	2^n , but manageable through pairwise independence.
List-ordering questions	A total or relative order exists on $2..n$ items that need to be established as the self-evaluation task.	$n!$
Concept matching questions	Concepts that are related can be matched to each other. $2..n$ concepts can be clustered into $1..n$ clusters.	$n!$
Cloze deletion questions	Also known as fill-in-the-blanks question and explained by the Anki [Webb] documentation [Webc]. First proposed by Taylor et al. [Tay53]. Very easy to generate as existing text must not be changed in its grammatical structure. A blank simply needs to be filled. Dealing with typos or synonyms is a challenge in the evaluation of this question type. Mathematical answers also can be given with different precision or as different equivalent mathematical expressions. A way to mitigate this problem is to provide fixed answers, for example in a multiple-choice format.	$ \Sigma ^{* \prod_{\text{TEXT_FIELD}} \text{MAX_FIELD_LENGTH}}$, where Σ is the input alphabet and MAX_FIELD_LENGTH the maximum text that fits in the input field: TEXT_FIELD .
Free-text questions	Definition 2.1.6	$ \Sigma ^{* \text{MAX_TEXT_LENGTH}}$, where Σ is the input alphabet and MAX_TEXT_LENGTH the maximum text length that fits in the input field.

Table 2.1: Question types and their answer space sizes.

2.2 Transformer Architecture

The original transformer paper introduces self-attention in combination with a feed-forward network. The network uses positional input encodings for self-attention [VSP+17]. Fig. 2.1 illustrates the model's architecture. The approach maps text, as a sequence of symbols, (x_1, \dots, x_n) to a sequence

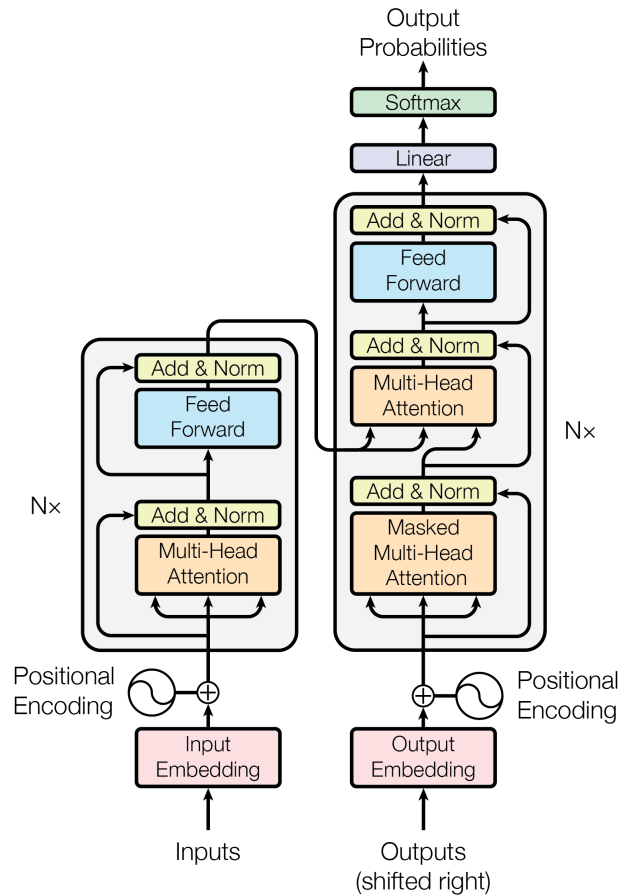


Figure 2.1: Transformer architecture [VSP+17].

of words (z_1, \dots, z_m) . The model's encoder then accepts those words as input [VSP+17]. The model processes all input at once in comparison to the previous approaches. Former state-of-the-art, recurrent neural networks (RNN), process input word-by-word. Sequential input renders parallelization difficult [VSP+17]. Back-connections store copies of the previous value. RNNs keep state in this way.

In comparison: a layer of the transformer encoder consists of a self-attention module. A feed-forward network follows and features skip-connections: $\text{LayerNorm}(x + \text{Sublayer}(x))$. Skip connections pass information to layers ahead. Layer normalization prevents skip-connections from amplifying the signal, Fig. 2.1 [VSP+17].

Both encoder and decoder layers stack $N \times 6$ times. Decoder layers process the output of encoder layers as input [VSP+17]. This information consists of previous predictions, excluding the current word to predict.

2.3 Transformer Models

Using transformer models can withhold certain restrictions. There are two possibilities to access a model:

1. **Model access via API** gives the organization that developed/trained the model control over the model's usage and modification. Training requires a large amount of compute and memory. A paid API helps the organization to compensate for training costs. The organization is able to use API requests to improve the model's performance and curate its contents. OpenAI's ChatGPT interface and API is an example of this usage [Webr; Webs].
2. **Pre-trained publicly available models** enables to use, modification and benchmark of the model according to given license restrictions. The larger the model, the more memory and compute it requires. Researchers need to take the model's resource footprint into account when running and/or fine-tuning the model. The Hugging Face library offers a wide range of openly pre-trained models and datasets [Tra]. Models are often published by researchers for transparency reasons [DCLT18; RSR+20].

In the following we will explore transformer models approaches by OpenAI and Google and compare their advancements in recent years:

2.3.1 Transformer Models by OpenAI

Training larger models on larger datasets makes it possible to achieve better performance in downstream tasks [RSR+20] [CND+22]. OpenAI's GPT-3 model trained with 175B parameters on 96 layers is considered state-of-the-art among NLP models, delivers coherent text-output [KRS21] and is able to learn on a few-shot basis [BMR+20]. GPT-3 features a decoder-only architecture and was trained on 499B tokens of training data, consisting mostly of filtered Common Crawl data [BMR+20].

OpenAI improved its GPT-3 model releasing GPT-3.5 models optimized for chat in different varieties:

GPT-3.5 variants (directly cited) [Webs]:

Latest model	Description	Max to-kens	Training data
gpt-3.5-turbo	Most capable GPT-3.5 model and optimized for chat at 1/10th the cost of text-davinci-003. Will be updated with our latest model iteration.	4,096 to-kens	Up to Sep 2021
gpt-3.5-turbo-0301	Snapshot of gpt-3.5-turbo from March 1st 2023. Unlike gpt-3.5-turbo, this model will not receive updates, and will only be supported for three months ending on June 1st 2023.	4,096 to-kens	Up to Sep 2021

text-davinci-003	Can do any language task with better quality, longer output, and consistent instruction-following than the curie, babbage, or ada models. Also supports inserting completions within text.	4,097 tokens	Up to Jun 2021
text-davinci-002	Similar capabilities to text-davinci-003 but trained with supervised fine-tuning instead of reinforcement learning	4,097 tokens	Up to Jun 2021
code-davinci-002	Optimized for code-completion tasks	8,001 tokens	Up to Jun 2021

Table 2.2: GPT-3.5 variants (directly cited) [Webs].

The table describes each GPT-3.5 model with the maximum number of tokens that are able to be processed in one input prompt. One token refers to around 4 characters according to OpenAI [Webai] and longer words can be split up into multiple tokens:

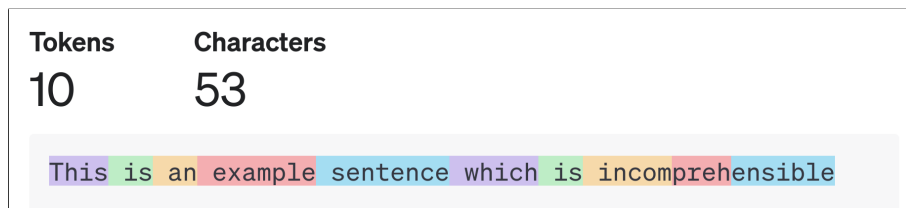


Figure 2.2: Token calculation from a sentence with OpenAI's tiktoken tokenizer for GPT-3.5 & GPT-4 [Webai].

The last column of Figure 2.2 describes up until which date input data is sampled.

OpenAI has released GPT-4 on the 15th March of 2023 [Webr]. GPT-4 outperforms GPT-3.5 in a variety of academic benchmarks, including MMLU (multiple-choice questions of 57 subjects), HellaSwag (Everyday commonsense reasoning), HumanEval (python coding tasks), etc. [Ope23]. Additional to previous models it allows visual input upon text input and improves upon safety measurements. ChatGPT's premium tier was upgraded to GPT-4 [Webr]. OpenAI has not released details about the architecture of GPT-4 due to "safety implications" and "the competitive landscape" [Ope23] that its product is placed in.

OpenAI offers a variety of state-of-the-art language models, however, those models are not openly available and can be accessed from OpenAI's API [Dal21] [Webah], where OpenAI limits access via the token system.

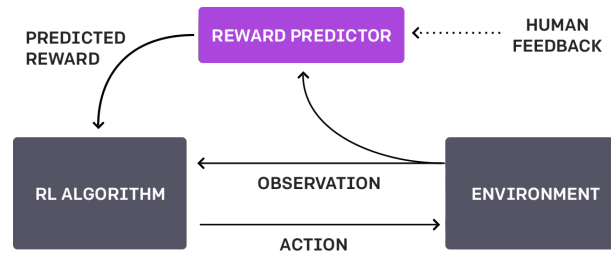


Figure 2.3: OpenAI: Reinforcement learning through human feedback [Webaj].

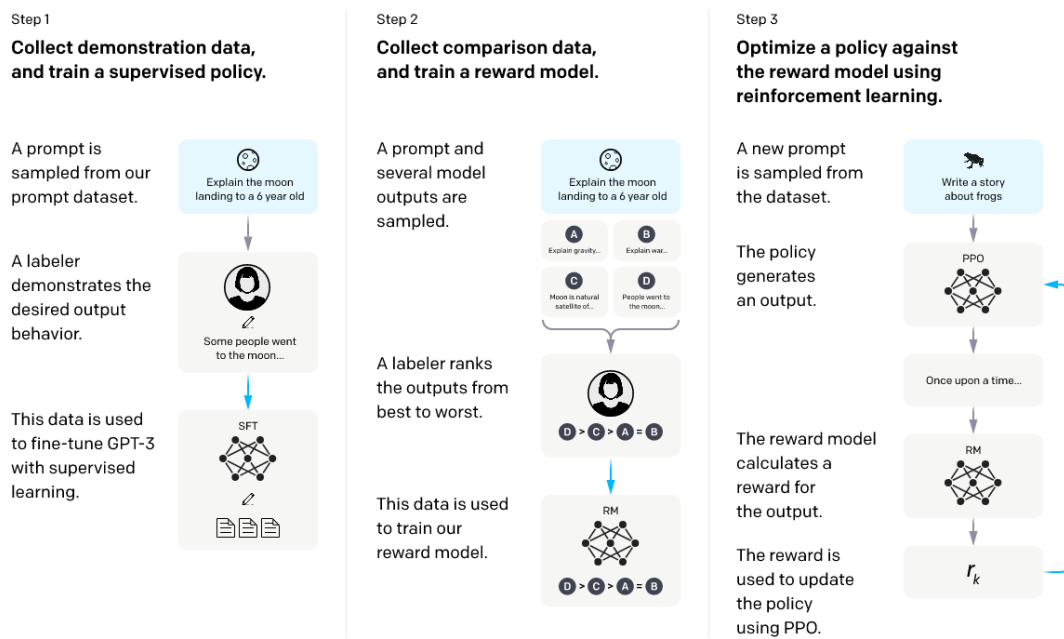


Figure 2.4: OpenAI: Language model alignment [Webaj].

Reinforcement Learning through Human Feedback (RLHF)

OpenAI applies RLHF for tuning its GPT-3.5 and GPT-4 models [Webs]. The goal of RLHF is to align models with human preferences [Webak]. RLHF is used in the three-step approach of model alignment that OpenAI presents on its website [Webaj]. Figure 2.4 depicts Step 2 of OpenAI's alignment method using RLHF. The language model outputs different versions that a labeler ranks for preferences. This trains a reward predictor similar to Figure 2.3. Step 3 describes the reward predictor to rank numerous outputs of the language model (Figure 2.4). The human curates the model's behavior regarding sensitive and dangerous topics. The model is incentivized to follow the labeler's preferences.

2.3.2 BERT

BERT is a language model released 2018 by researchers at Google [DCLT18]. BERT is available open-source under the Apache-2.0 license. The novelty of BERT is that it is trained on bidirectional representations. Bidirectional self-attention is used in comparison to constrained self-attention, as

implemented by the authors of the original transformer paper [VSP+17], where each token only references tokens to its left [DCLT18]. BERT is trained using a masking task (cloze [Tay53]), and a next sentence prediction task [DCLT18]. BERT-Large has fewer parameters (340M) than comparable language models of OpenAI [KRS21], excluding the original GPT model. Training happens in two phases. Pre-training on the Toronto Book Corpus and English Wikipedia and fine-tuned with a downstream task. For example, question-answering using a version of the Stanford Question Answering Dataset (SQuAD). Fine-tuning is "relatively inexpensive" in comparison to pre-training, as BERT delivers comparable performance with a limited amount of fine-tuning data [DCLT18]. BERT is an encoder network [DCLT18] and therefore better on prediction tasks than generative tasks, as translation and summarization highlighted by Raffel et al. [RSR+20].

2.3.3 T5

T5 (Text-to-Text Transfer Transformer) regards a variety of NLP tasks as text-to-text transformations [RSR+20]. Input and output are text representations. This design decision is made to improve comparability among transfer learning tasks. The model is trained with up to 11B parameters and uses a standard encoder-decoder transformer as proposed by Vaswani et al. [VSP+17] [RSR+20]. An extensive evaluation is conducted on how architecture, training and data affect the model's performance. Code, models and training data are released under the Apache-2.0, similar to BERT, and therefore benefits further research and usage [RSR+20]. T5 is built according to the model proposed by Vaswani et al. [VSP+17], differing in a few details: Layer normalization only features a scalar, without an additive bias. Position embeddings are relative instead of absolute. Positions offset up until 128 tokens can be attended to [RSR+20]. Training is conducted through a masking task, similar to BERT [DCLT18]. The model is fine-tuned and evaluated on GLUE, SuperGLUE tasks and SQuAD, as well as classification, general text understanding and translation [RSR+20]. Tasks are formulated in a text-to-text format and follow a specific format: Basic structure of a task: Task-specific information.

2.3.4 PaLM

PaLM is a 560B parameter transformer model announced by Google in 2022 [CND+22]. It uses a transformer architecture in a decoder-only setup, using SwiGLU activation functions with improved transformer blocks of a more parallel nature. The model incorporates RoPE (rotational position encoding) [CND+22]. Novelty is the training approach using Pathways, a system enabling more efficient training of deep neural networks [CND+22]. The non-fine-tuned model is able to outperform fine-tuned state-of-the-art models on a wide variety of tasks, using chain-of-thought prompting [CND+22]. The 780B token dataset contains Wikipedia pages, web pages, social media conversations, news and source code. $\approx 22\%$ of the training data consists of languages other than English. Multi-lingual evaluation of translation, summarization and question answering is conducted, with promising results [CND+22]. The model archives state-of-the-art performance on 28/29 English NLP benchmarks and performs well in zero- to few-shot scenarios [CND+22].

2.3.5 Other Transformer Models

Noteworthy language transformer models that archive state-of-the-art performance, comparable to GPT-3, are based on the architecture proposed by Vaswani et al. [VSP+17] and include: GLaM, Chinchilla, Megatron–Turing NLG and LaMDA as stated by Chowdhery et al. [CND+22].

2.3.6 Transformer Model Steering

Pre-existing models are able to produce the desired output by:

1. **Prompt-design (Prompt engineering)** enables the user to get the model to output the desired answer by precisely designing the text prompt and giving the model feedback iteratively with follow-up prompts (few-shot prompting). This describes using the model as-is and finding out which prompt gives the desired behavior by iterative testing. Few-shot learning, as well as Chain-of-thought (CoT) prompting [WWS+22] are techniques used in prompt design.
2. **Fine-tuning** enables feeding the models a series of prompts and completions to optimize the model for a specific behavior. The model itself is fine-tuned, by changing its parameters. Fine-tuning is more prompt to over-fitting than few-shot learning [BMR+20].
3. **Feature-based** approaches use "task-specific architectures that include the pre-trained representations as additional features" [DCLT18].

3 Related Work

In this section, we explore related work of self-assessment/question generation and describe the pipeline reference architecture.

3.1 Self-Assessment Generation

We describe our methodology to collect related work, categorize sentence selection from text input and compare multiple-choice generation approaches. Our findings reveal state-of-the-art weaknesses.

3.1.1 Methodology and Overview

We used Google Scholar [Webq] to search for existing question-generation approaches. The search term 'self-assessment generation' did not provide any related results, based on the titles and abstracts. We chose more general terms such as 'question generation' and 'multiple-choice generation' in response. We applied forward and backward scrolling to gather more resources about question generation. We ordered the results for recency, as we are particularly interested in the potential that transformer models hold for self-assessment generation. The transformer architecture is first introduced 2017 by Vasvani et al. [VSP+17]. Related work assesses transformer model question-answering capabilities, with promising results [Ope23]. Related work about self-assessment generation is sparse and has its limitations, as Section 3.1.4 identifies. We opted to take a look at less recent methods, to gain insight into general challenges and ideas about question generation. To our knowledge, multiple-choice is the most researched question type for question generation. However, none of the found papers connect question generation with a student-centered approach, with the key goal of benefiting the students learning. Such an approach would consider the findings and strategies of the didactic field. Research about self-assessment, and assisting a student-centered is out there [BB95] [Unib] [Unia] [Unic]. Existing work focuses on the technical aspect of mostly non-transformer approaches. We argue that both the technical and didactic side of question generation leaves room for improvement.

3.1.2 Sentence Selection from Input Text

Previous work focuses on multiple-choice questions, as defined in Table 2.1, in roughly three steps. 1. Sentence selection from an input text, 2. Question creation, 3. Distractor creation.

Input consists of a text corpus. A sentence selector chooses a subset of the input. This subset can even be a single sentence [MS15].

Majumder et al. explore a variety of different features for sentence extraction [MS15]:

- **Keywords:** A sentence selector can filter text through keywords. A set of keywords can be a topic. And topic modeling can find topics as keywords from the text.
- **Grammatical structure:** Grammar-based filters find sentences with specific grammar structures. This enables sentence-to-question conversion. Questions are sentences in interrogative form.
- **Position:** The position of a sentence in a document can provide information about the sentence's content. Abstract and conclusion sections are at the beginning of scientific papers and contain a condensed overview of the contents. A paper generally includes a problem statement, contributions and key findings. A reader expects specific bits of information at specific positions. The probability that a sentence belongs to the introduction can be statistically analyzed, based on its position in the document. Similar in the conclusion.
- **The length of a sentence speaks about its usability for question creation.** The longer a sentence, the more information it can carry. But, the more likely it is to include jargon: Redundant words, adverbs and empty phases. Co-reference resolution enables to separation of long sentences into shorter sentences [MS15]. This results in more sentences usable for question creation.
- **Pronouns**
- **Completeness of context**
- **Similarity to definitions**

3.1.3 Multiple-Choice Generation

Majumder et al. [MS15] present a system for multiple-choice generation using the topic modeling tool-set (TMT) [Webbh]. Their approach filters relevant sentences based on keywords and applies a process called co-reference resolution to separate long sentences into shorter sentences, isolating information related to keywords. First, a reference set with multiple-choice questions from a domain is collected. The collected reference questions are converted from interrogative to assertive form [MS15]. The reference sentences parse tree structure is analyzed and matched with the input text. Matching the input text allows the authors to find more "multiple-choice" sentence candidates. The approach adapts the test sentence's tenses before comparing the test sentence's structure with a sentence's structure from the input text. Conversion from assertive to interrogative takes the respective answer keywords into account. The approach selects answer keywords by named entity recognition, removes the answer keywords and converts the sentences from assertive to interrogative form [MS15]. The approach generates distractors by scraping the web. The authors evaluate their sentence selection approach in a survey. The participants categorize 93% of the sentences as usable. The participant evaluates 91% of the generated distractors as usable [MS15]. The grammatical structure of sentences can be used to identify multiple-choice question candidates. The authors state that their approach selects sentence candidates, which results in too general questions. Too general questions are hard to answer, as they omit information required for a definite answer. The answer to a question asking about "last year is dependent on when it is asked [MS15]. Question generation systems should generate robust, context-independent questions instead. Another limitation is that Majumder et al.'s approach generates questions and distractors about single-named entities. This

limits the possibility of asking about the relations of multiple named entities. A problem with the approach is that generated distractors can be valid answers. For example, the question: "Which person won the Olympics in [year]?", can have multiple answers. There is no mechanism to exclude scraped true distractors.

Araki et al. (2016) [ARS+16] generate question sentences from multiple input sentences. Their approach uses semantic text analysis. The authors argue that questions generated from a single sentence purely test low-level knowledge, unsuitable to "properly measure understanding of the overall contents" [ARS+16]. Their reliability checking system validates distractors as false and the answer as correct. The approach is able to create phrase-level distractors, a limitation of previous work [KM+06; MS15]. Their approach uses semantically annotated text, labeled by experts. The authors realize distractor generation by searching for annotations similar to the answer. A limitation of their approach is that questions and distractors can be grammatically incorrect [ARS+16]. Another limitation is that a part of the distractors can be easily eliminated. The answer is logically connected with the distractors in a few cases. The authors evaluate their approach to lecture materials from the biology domain. The biology domain features sequential processes. Questions about sequential processes often examine temporal relationships. The authors emphasize that semantic annotations used by their system reflect temporal relationships. The generalizability to other domains is unclear.

3.1.4 Question Generation with Transformer Models

Self-assessment question generation in a fixed specification format using transformer models has not been assessed by state-of-the-art, to our knowledge. Transformer-model-based question generation comes with limitations regarding question types, steerability and context awareness. Former approaches do not consider which users use their questions in which contexts.

Klein et al. (2019) [KN19] developed a system to generate questions and answers using transformer models, arguing that both tasks are closely connected and therefore can benefit from each other. The authors respectively use BERT and GPT-2, arguing that GPT-2 is suitable for question generation and BERT is suitable for question answering, due to the network's respective architecture [KN19]. Both networks work together end-to-end. The authors test the system on the SQuAD dataset and evaluate the quality of generation and answering. The authors compare their question generation approach using the BLEU [PRWZ02] and ROGUE [LO04] metrics. Their approach outperforms LM-init, referenced as a GPT-2 variant [RWC+19], but it is unclear if the improvement is significant.

Tsai et al. (2021) [TCY] present a method of retrieving sentences containing relevant keywords on a Python learning course using BERT, and generating self-assessment questions with GPT-2 [TCY]. However, the authors do not reason why they have chosen BERT and GPT-2 for the respective task and do not discuss possible other approaches for the respective sub-tasks. BERT is fine-tuned for domain-specific keyword recognition, improving keyword recognition performance from 94% to 98% [TCY]. We criticize that examples of falsely recognized keywords are not transparently highlighted. Domain-specific fine-tuning does not apply to lecture materials outside the domain of Python programming. The type of questions generated by Tsai et al. is stated as short answers, however, no definition of this question type is given. We consider the generated questions as not specific enough. For example: "What language does the body of the if statement come from?" [TCY] does not provide enough context to be answered with a definite answer. There is no indication

in which context the if statement is written: It could be written in another language than Python or even if written in Python include inline code from another language. We expect a question about Python to state: "written in Python" or "written in Python syntax" more specifically.

Nguyen et al. (2022) [NBM+22] present and evaluate a multiple-choice question generation approach on a data-science course using MOOCCubeX as a topic modeling tool. Their approach uses the T5 [RSR+20] model, fine-tuned on the SQuAD dataset. The dataset and source code are publicly available [NBM+22]. The authors do not offer a fixed format for generated questions. However, it is mentioned as a possible direction for future work.

3.2 Pipeline Reference Architecture

The pipeline reference architecture provides benefits regarding separation of concerns, parallelism and re-usability [RL77]. A pipeline can be defined as a directed graph, where each node is a module and each edge an interface. Data flows unidirectional, therefore the graph is directed. The graph must include a start and end node. The start node has no incoming connection and the end node has no outgoing connection. We focus on acyclic and non-splitting/merging graphs, which represent linear pipelines. A stateless pipeline module guarantees that output is only computed on the input. Stateless and deterministic pipeline modules guarantee the same output for the same input. This is important for pipeline scaling and parallelization. A parallel configuration of stateless pipeline modules allows horizontal scaling while mathematically behaving similarly to a single node. This is not the only way that pipelines provide parallelism. Each pipeline module can process its own data packet in isolation, data is passed through interfacing. Dependency-based pipeline modules, waiting for a specific amount of input data packets can be modeled through Petri Nets [CM90]. Such pipeline modules are stateful. A pipeline architecture-compatible process allows the implementation of each pipeline module in isolation. Independent implementation requires pipeline module interfaces to be well-defined and stable.

4 Requirements Engineering

The following sections describe the requirements engineering process. Requirements, highlighted in blue create the foundation for EvalQuiz's concept, presented in Chapter 5.

4.1 Problems with Self-Assessment

A student builds a mental model through the course and former knowledge. Figure 4.1 depicts this process. Lecturers teach a course through different modalities. Often, a lecture with lecture slides, exercise sessions with exercises or a book. Misconceptions in the student's mental model can occur, represented with a lightning symbol in Figure 4.1. Misconceptions can lead to unsatisfied course standards/criteria (Section 2.1). Self-assessment helps the student to resolve misconceptions. However, not all courses feature self-assessment. Limitations in creating, distributing and using self-assessment exist:

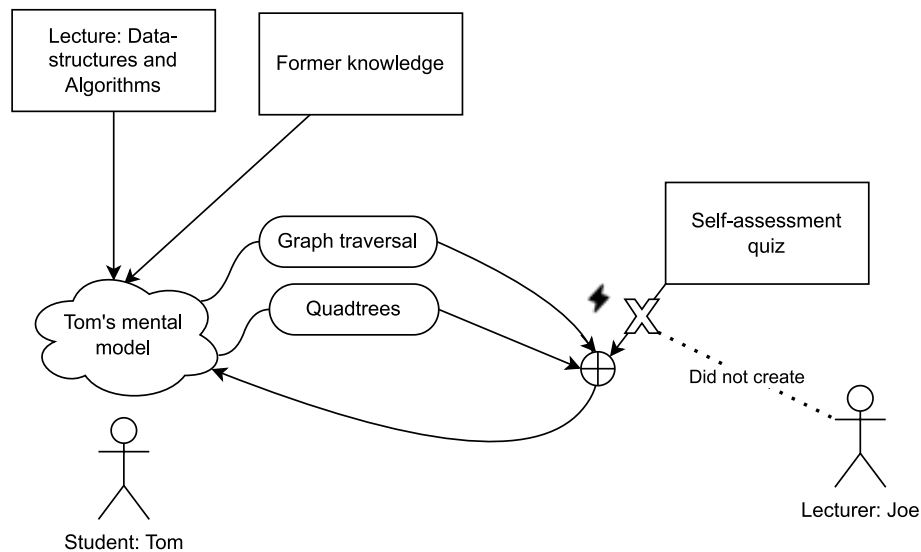


Figure 4.1: Self-Assessment: Student Benefits.

Lecture course creation and forum management occupy lecturers sufficiently. The maintenance of mandatory lecture slides and exercises does not leave room for non-mandatory self-assessment. Institutes are responsible for lectures in a university setting. Responsibilities are distributed hierarchically. Teaching is a part of the institute's responsibilities which include: Academic publishing, financing, sponsoring and supervision. Self-assessment creation is not easily outsourceable for multiple reasons: Domain experts are required to create high-quality and correct self-assessment (Figure 4.2). **The question's scope and depth need to be communicated and matched F-REQ-3.** Financing

of outsourcing is unclear as well. **Self-assessment should be cheap NF-REQ-1**. The institute additionally needs to evaluate if in-house creation is cheaper. Lecture materials and **self-assessment needs to be correct F-REQ-4**. Ideally **at least two domain experts approve self-assessment F-REQ-5**. That can be the lecturer who created the self-assessment and a coworker familiar with the domain, as the example in Figure 4.2 shows.

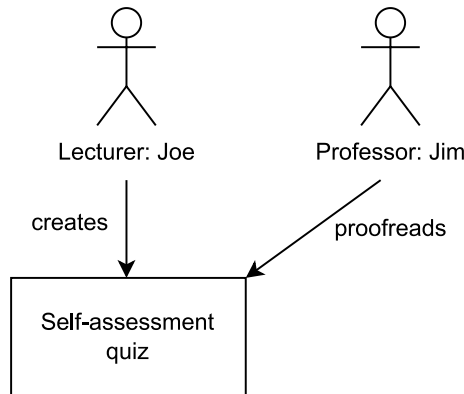


Figure 4.2: Self-Assessment: Required Domain Experts.

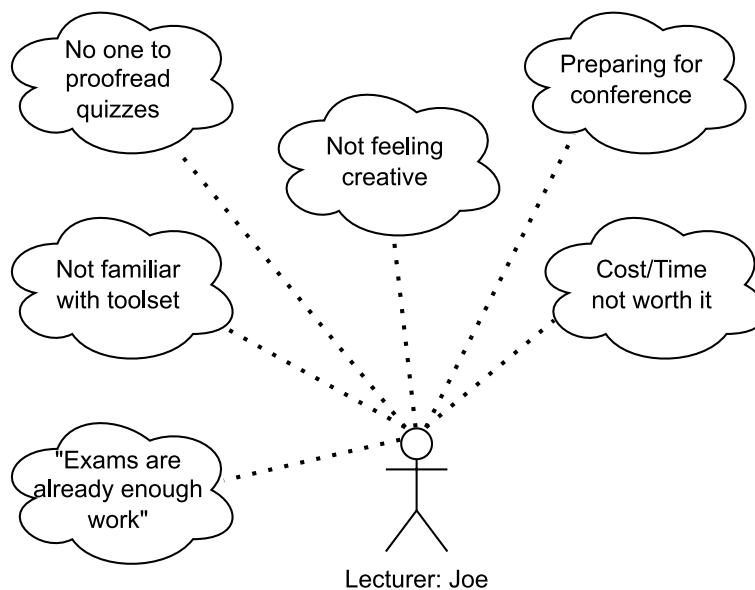


Figure 4.3: Self-Assessment Creation: Bottlenecks.

Self-assessment creation can be tricky. **Difficulty must match the lectures standards/criteria F-REQ-6** (Section 2.1). A low-level question asking for a single scientific term for example, is only quizzing the students **Knowledge** (Bloom's taxonomy, Section 2.1.1). The terms then need to be of high relevance. An isolated term lacks connection and context, making it hard to remember. High-level questions can lack depth. For example, building a question around: *Information hiding is the principle of segregation of the design decisions* [MO03] leaves out why information hiding is desirable **Comprehension** and how it can be used for a software program **Application** (Bloom's taxonomy, Section 2.1.1).

There is no [standardized question specification F-REQ-7](#) available, to our knowledge. Question integration in learning management systems depends on platform-specific data types or a markup language. This is a hurdle for cross-integration and re-usability. Learning management system-specific data-structure requires manual intervention for migrating questions to other services. Questions formulated in markup languages lack flexibility in terms of presentation and interaction. This begins with the learning management systems theme and ends with hints, rewards, plugins and gamification. An interactive puzzle is difficult to write in a markup language. Markup languages can integrate programming languages for interactivity. This comes with drawbacks in terms of platform specificity, learning management system compatibility and possibly performance. Integrating content written in different programming languages in one learning management system can create memory and computational overhead. Maintenance and testing can be more difficult, negatively affecting reliability.

Existing self-assessment generation approaches focus on multiple-choice questions, according to Chapter 3. The potential of other question types remains largely unexplored. Self-assessment can be versatile. An approach [extendable with different question types F-REQ-8](#) allows us to assess the full potential of self-assessment. Testing questions of different types does not require designing and building a new system.

A general system also should assist with selecting which lecture parts to choose a generation input. The system ideally selects by itself which lecture material parts are relevant to course standards/criteria. The selection process takes in multiple [lecture materials of different formats F-REQ-10 NF-REQ-2](#). The system allows [one or more lecture materials as input F-REQ-9](#). [The system should perform similarly on different material formats with equivalent content NF-REQ-3](#).

4.2 Didactic Specialist Interviews

We have conducted specialist interviews with experts in the didactic field to understand how a self-assessment generation system can assist lecturers and students. Notes taken during the interview are reconstructed as text answers. Valuable input regarding self-assessment and the interaction prototype helped to shape EvalQuiz in later stages. We started with a short presentation about the ideas, backgrounds and goals of the project, as a conversation primer to spark interest and communicate context.

4.2.1 Procedure

We first give a (1) slide-based project introduction presentation. Then conduct (2) an interview about self-assessment in general. (3) Introduce our interaction prototype and (4) collect feedback about our prototype in the form of an interview. Then (5) collect further ideas regarding the project.

4.2.2 Introduction of interaction prototype

The interaction prototype is a scenario of a lecturer using a system to:

- [Generate self-assessment materials F-REQ-1](#)

- Use an [iterative approach to improve the self-assessment generation outcome](#) **F-REQ-2**

We construct the interaction prototype according to the requirements highlighted in [blue](#). The interaction prototype is part of requirements engineering. We present the interaction prototype to didactic experts in order to collect more requirements and validate base requirements. For example, we want to assess if didactic experts consider [iterative human-model collaboration](#) **F-REQ-2** as valuable for self-assessment generation. General non-functional requirements are [ease of use](#) **NF-REQ-7**, [reliability](#) **NF-REQ-6**, [resourcefulness with memory and compute](#) **NF-REQ-5** and security.

We changed our first command line prototype sketch to a sequence diagram, depicted in Figure 4.4. We do not require didactic experts to be familiar with command-line interfaces. The prototype defines interaction in multiple steps: (1) connection, (2) material upload, (3) creation of question-generation specification, (4) audit question generation specification, (5) generation/iteration and (6) log/history.

A server-client architecture is chosen for several reasons: Setup on the lecturer client is not required. A web interface [removes the need for additional dependencies on the client side](#) **NF-REQ-4**, speaking of dedicated client software. The user does not need to install additional software, which improves [ease of use](#) **NF-REQ-7**. The server can manage multiple clients. Each connection serves a unique context. Materials and caches on the other hand can be shared between clients, [lowering the computational and memory footprint](#) **NF-REQ-5** that every isolated client would have. A server could run a large language model (LLM), which can be problematic locally, due to the large amounts of random-access memory (RAM) usually required for larger models. A client-only setup can decrease [reliability](#) **NF-REQ-6** due to the variety of environments that need to be supported. [Different operating systems \(OS\)](#) **NF-REQ-8**, dependency version conflicts etc. Supporting a single OS is also not an option, as ruling out potential use cases on different systems by design is not a good choice. Another benefit is server-side secret management, mitigating the problem of client-side secret sharing and [reducing the risk for secrets to be compromised](#) **NF-REQ-9**. Secrets can be required, if the LLM is not run locally but provided as a third-party service [Webr].

The user configures a [question-generation-specification to steer and configure self-assessment generation](#) **F-REQ-17**. This document references lecture materials. Those need to be present on the server and can be uploaded through the interface. Integrating lecture materials directly into the config poses a repetitive workload on the network if a generation with similar lecture materials is repeated. This scales up with the number of users, requiring potentially more resources from the hosting provider. Separating material upload and generation only [requires upload bandwidth once](#) **F-REQ-18**.

The prototype realizes interaction with an [iterative approach](#) **F-REQ-2**. It is [possible for the user to configure a question-generation-specification, iterate/generate, judge the outcome, adjust parameters accordingly and rerun the iteration/generation. Reiteration is also possible on the same configuration](#) **F-REQ-19**, as the output of LLMs is nondeterministic. A user therefore has the incentive to rerun the same question-generation-specification multiple times. The conceptualized lecturer client features a [version control to save and compare different config versions](#) **F-REQ-20**.

Audit mechanisms are added to [visualize the difference between the previously iterated question-generation-specification and the current question-generation-specification](#) **F-REQ-21**, in order to improve the visibility of the systems status. One of Nielsen's ten interface design principles [Nie94].

History/Log builds on audit by presenting different visualizations of the whole history. History is deliberately managed on the client to decrease server state and leave the decision to which degree history is managed to the client implementation. History/Log is only used where needed and can be added in case it provides a benefit for the specific use case.

4.2.3 Condensed Findings: Interviews

This subsection condenses the interview findings. Appendix Chapter A states the full interviews.

It is important to consider each question in relation to its surrounding questions to arrange the sequence in which questions are asked carefully. Asking questions is an act of communication. Questions that are topically similar should be adjacent or at least close to each other [Unia] **F-REQ-11**. Topic jumps should be prevented as well as possible **F-REQ-12**. In order to connect sections of different topics, transition elements should be designed. Topic sections are marked by headlines. The inquiry purpose is best stated at the beginning of the document, followed by an "icebreaker" question, sparking interest and motivation to continue [Unia]. The most important questions should be placed in the first third of the inquiry. **F-REQ-13** Each topic should be assessed from general to specific questions [Unia]. Context from previous questions can mislead or bias to specific answers. More information about context effects can be found in the "Item and question construction guidelines" of the ZLW University of Stuttgart [Unia], as well as additional question-wording guidelines [Unia] **F-REQ-14**:

- Simple and clear language
- Simple and short questions
- Positive language
- Not suggestions or assumptions

Questions can be composed based on educational objectives [BK20] **F-REQ-15**, which focus on specific lecture parts. The idea is to measure to what degree an educational objective is satisfied **F-REQ-16**. Individualized teaching **F-REQ-24** is a strength of a self-assessment generation system. In contrast to traditional teaching, with fixed-size classes where each student follows the same content in the same time span, online systems can take a different approach. Individual advantages of new methods should be explored, instead of building around existing educational frameworks. Currently, a fraction of the new options discussed by educational sciences are incorporated into university curricula. Furthermore, the context in which questions are asked always depends on the individuals partaking in that class. A lecturer adapts their teaching to the student's unique circumstances. Not every year is similar in firstly the student capability levels and outside factors. **F-REQ-25**. A system needs to allow adjustment for student capability and outside factors **F-REQ-25**. For example, a pandemic requires to switch to online lectures. The lecturers need to readjust their curriculum for online teaching and accommodate their students needs **F-REQ-26**.

4 Requirements Engineering

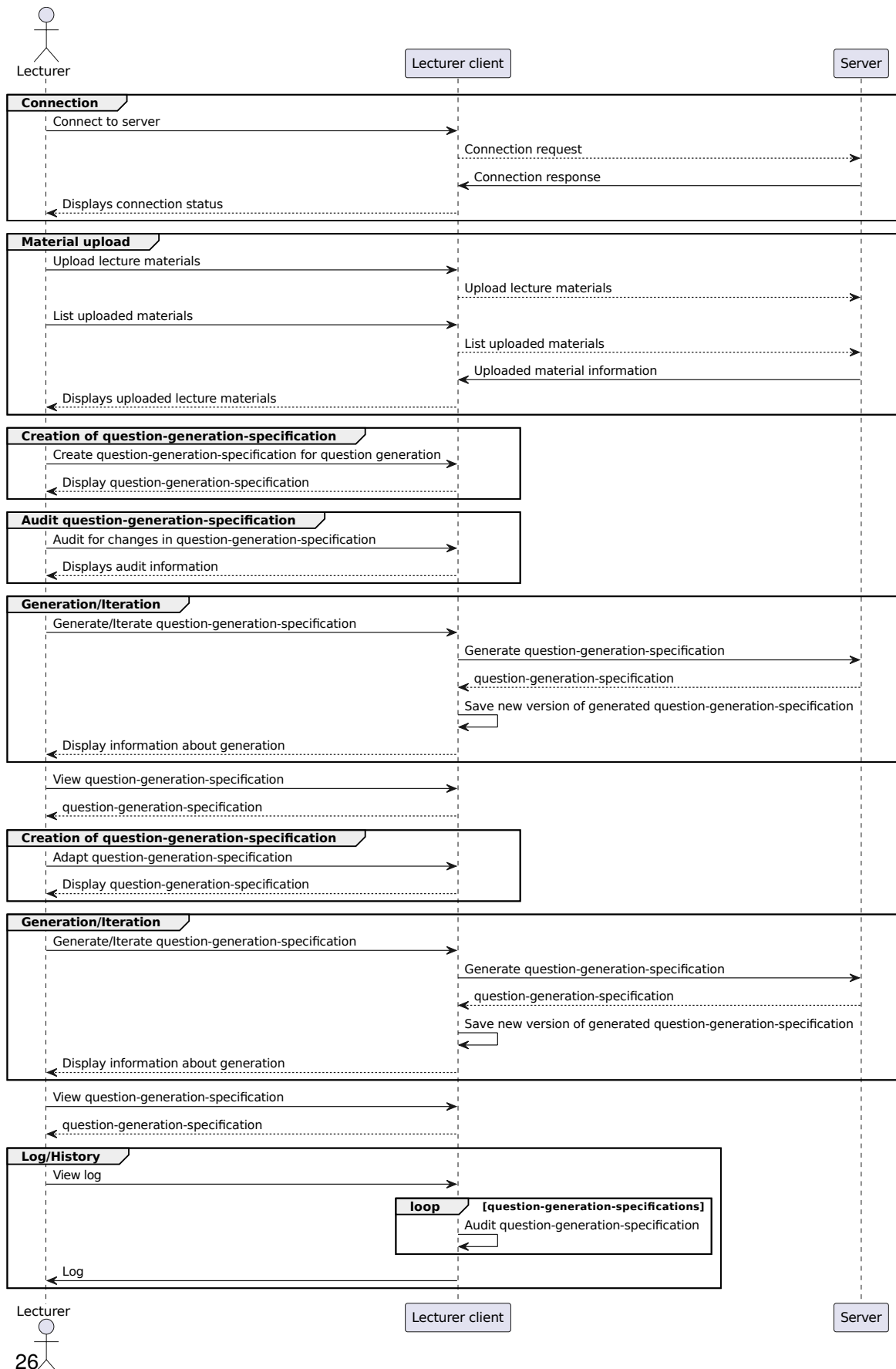


Figure 4.4: Lecturer interaction prototype: Sequence diagram.

4.2.4 Collection of Further Project Ideas

Individualized teaching F-REQ-24 can be a strength of self-assessment generation systems. New approaches, such as self-assessment generation systems, allow for higher flexibility in comparison to traditional approaches. Traditional teaching embraces fixed-size classes, uniform curricula and fixed schedules for students. The advantages of new methods should be explored. Currently, a fraction of the new options discussed by educational sciences are incorporated into university curricula. Furthermore, the context in which questions are asked always depends on the class of individuals. An in-presence class allows the lecturer to adapt teaching to the student's unique circumstances. Additionally, not every year is similar in the **student's capability level and the outside factors.** A system needs to allow and embrace **F-REQ-25** student capability and outside factors. For example, the pandemic required lectures to switch to learning management systems. Lecturers needed to readjust their curriculum to new formats while **accommodating for their students needs F-REQ-26.**

4.3 Student Benefits Through Self-Assessment

We propose an idea that tackles the difficulties of self-assessment as stated in the section 2.1. Students have to understand standards/criteria and estimate how well they have met those. Resulting in two steps. A simplification reduces both steps to a single step, easier for students to verify:

A standard/criteria [BK20] maps to a self-assessment question **F-REQ-6**. The self-assessment question triggers a training effect on the standards/criteria described. The ability to answer the question correctly indicates strength in the standard/criteria.

For example in a physics course:

Criteria on Bloom's taxonomy): **Know and understand** (syn. **Comprehension**) for which elements nuclear fusion releases energy.

- **Initial approach:** Find and read the lecture material section about nuclear fusion. Estimate the depth/detail required for satisfying "know and understand". Then estimate, if the given answer is correct (see self-assessment definition in Section 2.1) 2.1. Possible depths:
 1. List all elements that can release energy on nuclear fusion.
 2. State shared property of all elements releasing energy on nuclear fusion.
 3. Explain why the shared property of all elements releasing energy makes them release energy.
- **Combined approach:** Answer given multiple-choice question: What is the shared property of all elements releasing energy on nuclear fusion?
 1. Lower atomic mass than iron-56
 2. Higher atomic mass than iron-56
 3. Only carbon

The **combined approach** can reduce misconceptions about standards/criteria and simplify self-assessment. In the case above it is clear to which depth the criteria should be **”known and understood”**. Only the property of atomic mass is relevant for the lecturer. The question gives feedback about the correct answer: ”Lower atomic mass than iron-56”. This generalizes to other question types with direct feedback answers. Furthermore, it allows to [assess multiple standards/criteria at the same time](#) **F-REQ-15**. Questions then trigger a training effect on more than one standard/criteria

Questions of unambiguous question types 2.1.2 can reduce false answers. The lecturer specifies all answer options as either true or false in advance.

4.4 Language Barriers

It is expected that question generation using a general language model performs better on English course materials, as the largest part of training data to train state-of-the-art models is written in English. BERT is trained on the Toronto Book Corpus and English Wikipedia [DCLT18]. T5 proposes the C4 dataset containing ”English text scraped from the web” [RSR+20], etc. Testing and development are preferably done in text-rich English lecture materials [Webbj]. English computer science courses are widely available. Also in countries where English is not the first official language. Therefore, we focus on testing our method on [English lecture materials](#) **F-REQ-22**.

4.5 Visual Materials

Courses with few text-rich materials and teaching concepts with images instead of text explanations render it more difficult to generate self-assessment questions, as image-to-text transformation is necessary for text-only models to enable question generation on the given information. However, multi-modal models exist that are able to process visual information by itself, as OpenAIs GPT-4 [Webr]. We aim to explore [utilization of visual information](#) and leave it as an [additional feature](#) **F-REQ-23** to our approach.

4.6 EvalQuiz Self-Assessment System: Requirements Summary

This section summarizes the functional and nonfunctional requirements resulting from the previous considerations and explanations.

4.6.1 Functional Requirements

- F-REQ-1** The user can generate self-assessment with the system.
- F-REQ-2** The system uses an iterative approach to improve the self-assessment generation outcome.
- F-REQ-3** The user can communicate a question’s scope and depth to the system. The system matches the question’s scope and depth.

- F-REQ-4** Self-assessment created with the system is correct.
- F-REQ-5** The system follows the "four-eyes" principle. The system is the first actor and needs another actor to verify its output.
- F-REQ-6** The user can communicate a lecture's standards/requirements to the system. Self-assessment created with the system matches the lecture's standards/criteria (and level of difficulty).
- F-REQ-7** The system creates questions in a standardized format: Question specification.
- F-REQ-8** The system can be extended with different question types.
- F-REQ-9** The system generates self-assessment with at least one are more text-based lecture materials.
- F-REQ-10** The system supports common text-based lecture material formats.
- F-REQ-11** Topically similar questions are placed adjacent or close to each other.
- F-REQ-12** The system structures resulting questions as topically similar segments.
- F-REQ-13** The user has control to place the most important questions at the top of all resulting questions.
- F-REQ-14** The system allows to check questions according to the ZLW question-wording guidelines [Unia].
- F-REQ-15** The system allows questions creation upon multiple educational objectives [BK20].
- F-REQ-16** The system allows the user to assess to which degree an educational objective is satisfied.
- F-REQ-17** The user can steer self-assessment creation through a single question specification.
- F-REQ-18** The system requires the user to upload lecture materials only once. The system reuses the lecture material if uploaded.
- F-REQ-19** The user can configure a question specification, iterate/generate, judge the outcome, adjust parameters accordingly and rerun the iteration/generation.
- F-REQ-20** The user can compare different question specification versions.
- F-REQ-21** The user can visualize a question specifications history.
- F-REQ-22** The system supports the English language.
- F-REQ-23** The system can be extended to create self-assessment creation with visual materials.
- F-REQ-24** The system allows for individualized teaching with created questions.
- F-REQ-25** The user can input student capability levels and outside factors. The system takes student capability levels and outside factors into account.
- F-REQ-26** The user can configure the system to adapt to the student's needs.

4.6.2 Nonfunctional Requirements

- NF-REQ-1** The systems development, operation and deployment is economical.
- NF-REQ-2** The system scales with large amounts of lecture material data.
- NF-REQ-3** The system should perform similarly on different material formats with equivalent content.
- NF-REQ-4** The system requires minimal user client setup.
- NF-REQ-5** The user client's computational and memory footprint is low.
- NF-REQ-6** The system works reliably.
- NF-REQ-7** Lecturers with different backgrounds can use the system with ease.
- NF-REQ-8** The system supports common operating systems (OS).
- NF-REQ-9** The user cannot access API secrets.

5 Concept

EvalQuiz’s concept processes lecture materials with a ”generation pipeline”. A domain expert supervises self-assessment creation, satisfying **F-REQ-5**. The domain expert proofreads generated output. Question created according to the question specification enables their deployment on a learning management system **F-REQ-7**. Other usages are imaginable, for example, composing mock exams.

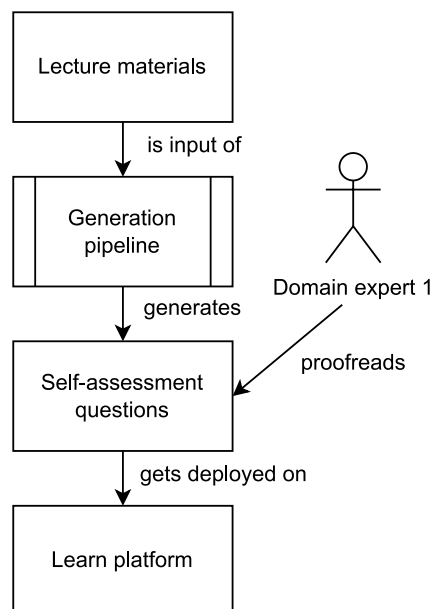


Figure 5.1: EvalQuiz Self-Assessment Concept Overview.

5.1 Pipeline modules and data

Data is processed in a pipeline-reference architecture approach. Figure 5.2 sketches EvalQuiz’s data processing:

The pipeline decomposes the problem of self-assessment generation into smaller subproblems. Execution of each module is independent from other modules. Only parts needed in the current execution have to be available through memory. Module-based caching benefits executions by providing calculated results. This speeds up execution and requires less RAM than without caching **NF-REQ-5**.

Decomposition has advantages for code maintenance, development, re-usability and extend-ability. It is possible to exchange pipeline modules or reuse them in other contexts **NF-REQ-6**.

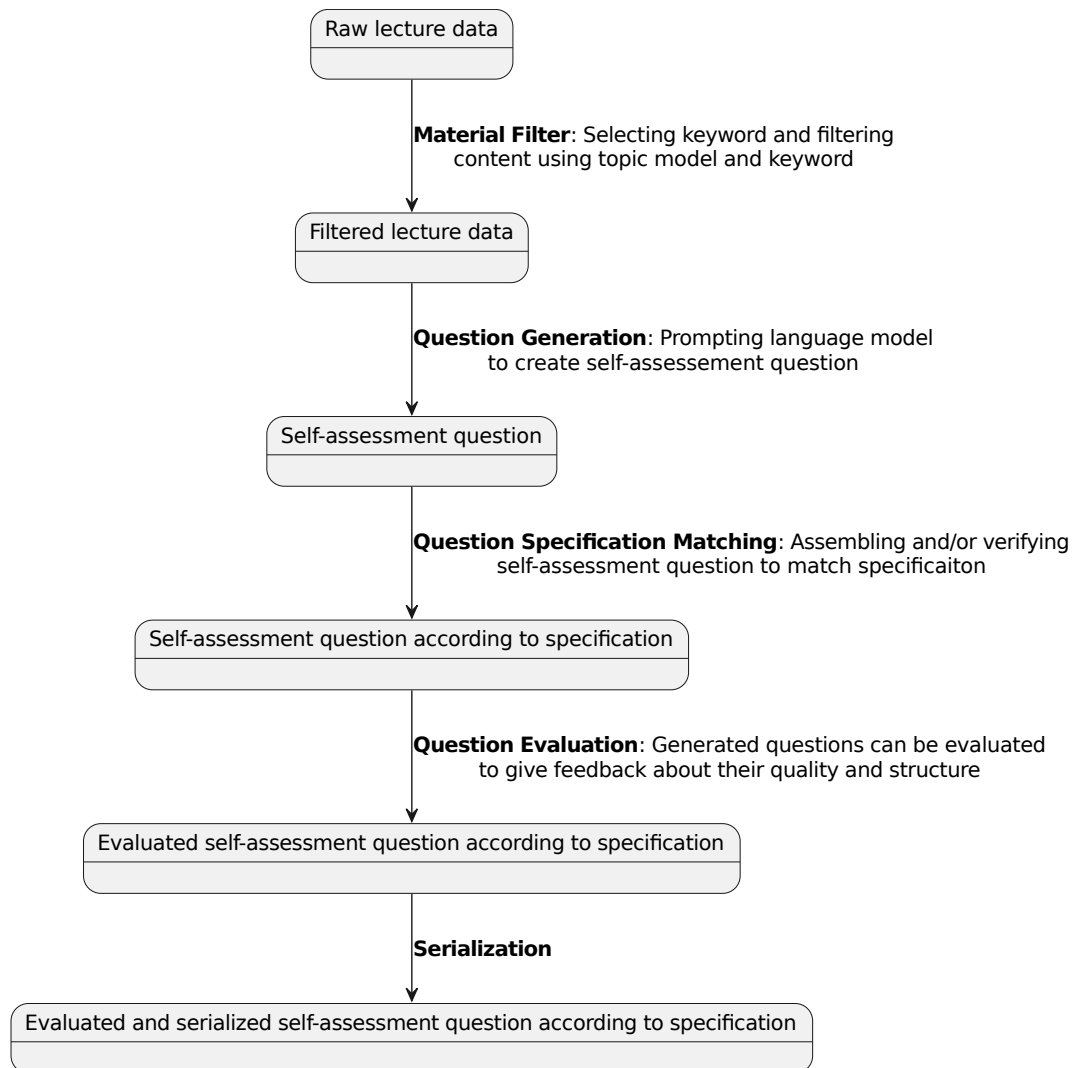


Figure 5.2: Processing pipeline.

We conceptualize pipeline modules to describe EvalQuiz’s functioning. A user inputs lecture materials (“Raw lecture data”) to EvalQuiz. The approach processes data in many intermediate pipeline module steps. The result is a “serialized self-assessment question according to specification” **F-REQ-7**.

The **Material Filter** filters input documents to select parts of specific topics. A reduction of quantity enables generation in later steps. A user can influence this filtering process. Manually selecting content on which questions are created can be tedious. Section 5.3 illustrates why the **Material Filter** is necessary.

Next, the **Question Generation** uses processed input to generate questions **F-REQ-1**. Followed by **Question Specification Matching**, ensuring that the generated content matches the expected datatype **F-REQ-7**. This module casts data if necessary.

The **Question Evaluation** step enables automatic evaluation of generated questions. This gives the user more insight into the quality of generation **F-REQ-16**. **Question Generation** enables automatic regeneration if a question is lacking in quality **F-REQ-14**.

Serialization ensures that the generated output is transmittable over the network.

Pipeline interaction follows the human-in-the-loop principle. Similar to the interaction described section 4.2.2 and visualized in Figure 5.3. A lecturer directly inspects the output of the pipeline, can adjust the question ordering **F-REQ-13**, judge the quality and correctness of the questions **F-REQ-4** and iterate upon the question configuration **F-REQ-2**.

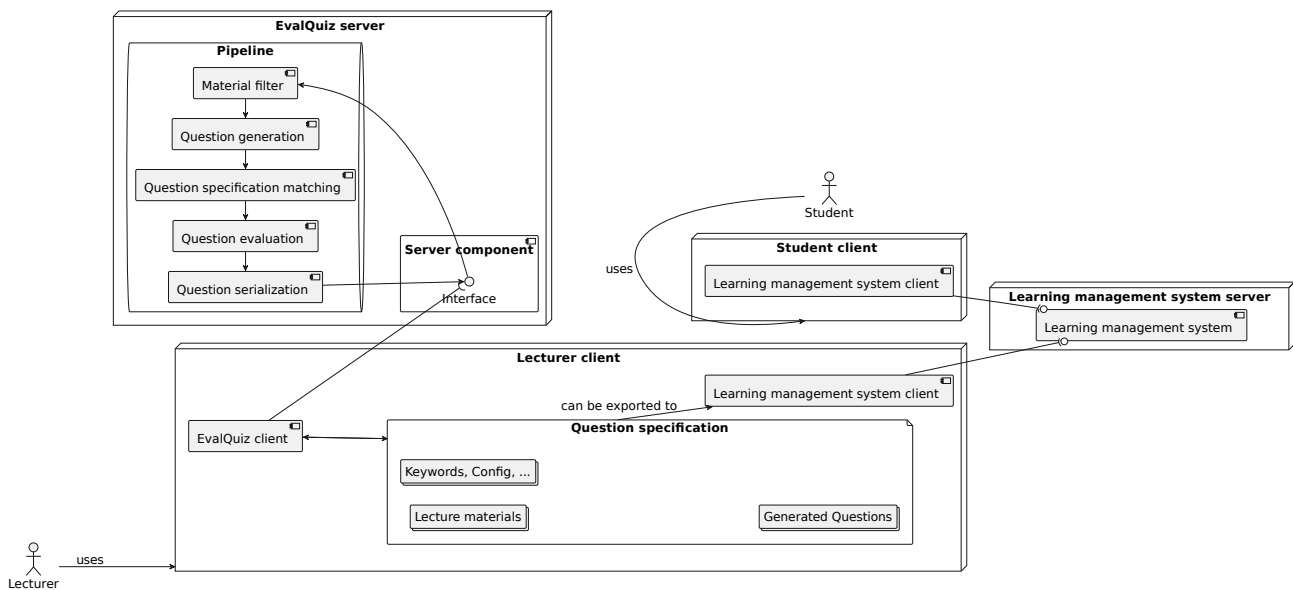


Figure 5.3: Iterative pipeline interaction.

5.2 Matching/Converting Model Output to Specification

Our approach synthesizes questions according to a question specification **F-REQ-7**. A transformer model output needs to be parsed to match this specification. Model output can take the following forms:

1. Output matches specification: Output is directly parseable.
2. Output roughly matches specification: Further modification might be required. Possible solution: Retry.
3. Output does not match specification: Direct parsing impossible. Possible solution: Iteratively argument data with further prompts until it matches the specification.

(Case 1) A transformer model needs to internalize the specification and be able to reproduce data according to it. Internal reflection about which outputs are valid and which are invalid is necessary.

EvalQuiz aims to solve the question, of whether output according to the specification (case 1) can be archived only with few-shot examples. Fine-tuning a model is an option, but can occupy additional resources for training and/or cause extra costs [Webr].

Part of the thesis is to research whether prompts can be designed to deliver good enough results using few shot techniques. We aim to explore other model training options, if the results are not good enough (case 2, case 3). This includes multi-model approaches, where intermediate output is processed by a second model optimized for datatype matching/conversion. Data error correction models are options for datatype matching/conversion.

Our approach needs to check the output subtypes in order to verify that the parsed data is correct. This is essential for the correctness of the following steps.

5.2.1 Multi Prompt Self-Assessment Creation: Concept

An idea is to create data according to the specification with multiple prompts if direct parsing is impossible (case 3). An algorithm can be designed to carry out this process. The following example explores this idea. Two actors: A user and an AI interact with each other:

Scenario: The AI can access lecture material that the user wants to create a multiple-response question with. The AI is instructed to create self-assessment about Model-Driven-Development:

User: Generate a question about Model-Driven-Development.

AI: What are the advantages of self-describing models?

User: Please give me a correct answer for your generated question. Double-check that your answer is correct by reading the lecture materials.

AI: The definition of the model does not rely on another model, but is self-containing.

User: Please give me different a correct answer for your generated question. Double-check that your answer is correct by reading the lecture materials.

AI: Self-describing models can be used as meta-models. Meta models can describe more specific models.

User: Please give me a distractor for your generated question.

AI: Self-describing models are simpler to define than non-self-describing models.

Next, an algorithm converts each model output to a specification field. In this case: Multiple-response: question_text, answer_texts and distractor_texts. A sequence of questions and their respective answers are combined into a question matching the specification. A drawback is that the model is queried more often: This can be a problem if the model forgets its message history.

5.2.2 Single Prompt Self-Assessment Creation: Prototype

We have developed a simple working prototype that uses the OpenAI API to generate multiple-choice questions. It queries gpt-3.5-turbo with the following prompt: Create 3 multiple choice questions with multiple possible answers about the following text:, followed by the filtered text. Multiple text files can be parsed as arguments and filtered by the `-k --keyword` parameter. Whole sentences containing the keyword are retained. Output is the direct response by the model, which Section 5.2 classifies as case 1. On the first attempts, multiple-choice generated by the approach looks promising, but a proper evaluation is needed before jumping to conclusions. First tests are conducted on English Wikipedia about undergraduate computer science concepts, the Scrum guide [Webbd] and a text generated by GPT-4 [Webr] using ChatGPT about Eclipse' QVTo [Webk]:

```
$ python evalquiz-prototype.py --help
```

```
usage: evalquiz-prototype.py [-h] [-k KEYWORD] [-w WORD_LIMITATION]
                             [-n NUMBER_OF_QUESTIONS]
                             [files ...]
```

positional arguments:

```
files                Text files that serve as the input for the self-
                    assessment question generation.
```

optional arguments:

```
-h, --help            show this help message and exit
-k KEYWORD, --keyword KEYWORD
                    Keywords that the text should be filtered for.
-w WORD_LIMITATION, --word_limitation WORD_LIMITATION
                    The maximum number of words that are input into the
                    model.
-n NUMBER_OF_QUESTIONS, --number_of_questions NUMBER_OF_QUESTIONS
                    The number of questions that should be created by the
                    model.
```

A simple form of interaction is given by the CLI, which offers the user the possibility to integrate the proposed tool with other applications, host it as a micro-service, etc. For our objective to make self-assessment generation available to university courses, further work needs to be conducted. For example: a lecturer of the biology domain, who is not as familiar with CLI interfaces, is likely to find it more time-consuming to set up a question-generation tool, than writing questions by hand. The solution can be a GUI querying a backend.

5.3 Model Steering

The input of the language model is text from lecture materials **F-REQ-9**. Language models only support prompts up to a certain size. This is due to the model architecture or API specification [Webai]. Lecture materials can exceed these boundaries easily. A single book chapter can already exceed the max. size of 4096 tokens that OpenAI's GPT-3.5 supports [Webs].

There are a variety of methods to handle large data as input. Splitting up chapters and processing each part individually is one option. But this leads to a major disadvantage. Cross-connections of different chapters of a lecture become difficult. References to text parts not in the model's memory become obsolete. The model views parts of the text in isolation. Longer materials become unusable for question generation.

Another way to mitigate this problem is to use hierarchical summaries. References can be embedded by summaries. An algorithm then resolves references. The language model used for question generation is also capable of creating summaries. This is an interesting idea to explore in future work.

Our solution takes a simple approach to solve two problems at once. Making input manageable for the language model and steering the self-assessment generation **NF-REQ-2**. We select a subset of the input text, which then contains content on which we generate questions **F-REQ-9**. The **Material Filter** filters a collection of lecture materials. Information on related topics is kept but length is reduced.

We propose a keyword-based algorithm described by Algorithm 5.1:

Algorithm 5.1 Material Filter.

```
procedure FILTERSENTENCESWITHKEYWORDS([sentence0, . . . , sentencen-1],  
[keyword0, . . . , keywordn-1], maxAllowedLength, extendCount)  
  sentences ← [sentence0, . . . , sentencen-1]  
  keywords ← [keyword0, . . . , keywordn-1]  
  extendedKeywords ← keywords concatenated with extendCount similar words through topic  
  model  
  filteredSentences ← []  
  for keyword in extendedKeywords do  
    for sentence in textInput do  
      if Accumulated token count of all filteredSentences < maxAllowedLength then  
        if keyword in sentence then  
          Insert sentence to filteredSentences in the original order  
        end if  
      end if  
    end for  
  end for  
return filteredSentences
```

The algorithm searches for and selects sentences with keyword synonyms. Sentences containing one of the keywords are sequentially added until exceeding the length threshold `maxAllowedLength`. The original order of sentences is kept. A topic model provides word embedding for this search. The goal is to extract a text of a certain length. The length should be compatible with the language model input. A drawback is the algorithm cuts off information if `maxAllowedLength` is too small.

This could result in fragmented output. Adjacent sentences could lack semantic connection. A mitigation option is increasing the `maxAllowedLength`. We expect future work to refine this algorithm with more sophisticated methods.

Language models with a `maxAllowedLength` greater than avg. lecture materials would be capable of processing input without filtering. But then need to be sufficiently capable of focusing on the parts that the user wants.

5.3.1 Topic Model Enabled Keyword Augmentation

Factors a researcher needs to deal with using a topic modeling tool given by Asmussen et al. [AM19]:

- Pre-processing of text
- Selection of model parameters and number of topics to be generated
- Evaluation of reliability
- Valuation of validity

The topic model selects the most similar keywords to the given keywords (keyword list). Those augment the original keyword list. Algorithm 5.1 only selects sentences containing keywords in the keyword list. The user specified all keywords in the original keyword list. A keyword that occurs only in the input lecture materials can be problematic. The output becomes a single sentence. We want the output to be descriptive in order to withhold lecture material information **F-REQ-9**. The goal of keyword augmentation is to find connected keywords to prevent Algorithm 5.1 outputs sparse results. Algorithms 5.1 output with keyword augmentation represents a topic. A topic can be described with synonyms and different forms of speech and still represent the same concepts. Keyword augmentation aims to provide topical in-variance for sentence selection. This is to improve the **Material Filters** consistency and therefore reliability **NF-REQ-6**.

5.3.2 From Keywords to Capabilities

The option to use multiple keywords gives the user further control over how questions should be composed is considered and needs to be further investigated. For example: two keywords "TCP" and "UDP" enable the generation of questions that highlight the relation between/roles of both technologies. We propose to extend keywords to capabilities to satisfy the requirement: **F-REQ-15**. Capabilities are more powerful than keywords. They specify the keywords educational objective according to Bloom's taxonomy (Section 2.1.1). For example: "TCP", "UDP" should be applied (**Application**, Section 2.1.1). We describe lecture standards/criteria with capabilities. The third part of a capability is the relationship. The relationship specifies the educational objectives scope. Example are: "differences", "similarities" and "unique features".

Capabilities represent EvalQuiz's standards/criteria. We use capabilities to define lecture standards/criteria as well **F-REQ-6**. Algorithm 5.1 similarly works with capabilities, as keywords are capabilities fields.

5.4 The Transformer Models Role

The transformer model is a key component of the processing pipeline. It creates the self-assessment material through its input and the selected question type **F-REQ-8**. Its input is a subset of the lecture materials, dependent on keywords selected by the user. The transformer model carries out main rephrasing, summarization and combination **F-REQ-1**. It converts input to a data representation matching the self-assessment specification **F-REQ-7**. The model should primarily generate questions/answer pairs covered by the lecture materials. However, the model is advised to insert arbitrary information into distractors. Distractors are used by multiple-response and multiple-choice questions, sec 2.1.2.

5.5 Specification of Self-Assessment Question Types

A specification is required **F-REQ-7**. Self-assessment questions can be formulated as YAML or JSON:

```
1 {
2   "version": "string",
3   "questionType": "QuestionType",
4   "content": "QuestionContent"
5 }
```

where QuestionType: MultipleChoice | SingleChoice | SingleChoiceCloze | MultipleChoiceCloze | ...

Example of multiple-choice datatype:

```
1 {
2   "version": "1.0",
3   "questionType": "SingleChoice",
4   "result": {
5     "question": "How tall is the Eiffel Tower?",
6     "answer": "324m",
7     "distractors": ["354m", "424m"]
8   }
9 }
```

Example of multiple-response datatype:

```
1 {
2   "version": "1.0",
3   "questionType": "MultipleChoice",
4   "result": {
5     "question": "Which atoms are contained in water?",
6     "answers": ["H (Hydrogen)", "O (Oxygen)"],
7     "distractors": ["He (Helium)", "Fe (Iron)"]
8   }
9 }
```

Example of a multiple-choice cloze datatype:


```

1 {
2   "version": "1.0",
3   "questionType": "MultipleChoice",
4   "result": {
5     "question": "Water is an inorganic compound with the chemical formula {{c1::H_2O}}",
6     "distractors": [{"CH_4", "C_8H_{10}N_4O_2", "CO"}]
7   }
8 }

```

Cloze question formatting is described in detail by the Anki documentation [Webc]. Cloze questions are formatted as `{{ID::Cloze::Hint}}`, where ID is formatted as `cn` with $n \in \{1, \dots, k\}$ starting from 1 and increasing in steps of 1, Cloze describes the hidden text. Hint describes a hint that can be shown, without revealing the answer.

We can construct an example with multiple cloze fields and a hint:

```

1 {
2   "version": "1.0",
3   "questionType": "MultipleChoice",
4   "result": {
5     "question": "The {{c1::registrar::enables to lease second-level domains}} communicates the
6     ↪ authoritative name servers to the domain registry for the {{c2::top-level}} zone.",
7     "distractors": [
8       ["internet service provider", "government", "signing agency"],
9       ["first-level", "second-level", "entire"]
10    ]
11 }

```

Distractors are ordered similarly to the cloze fields. The additional hint is described after the `:: Enable to lease second-level domains`.

EvalQuiz supports multiple-choice and multiple-response question types. Cloze versions of multiple-choice and multiple-response are possible additions to extend EvalQuiz's feature set in the future **F-REQ-8**.

The Appendix Section B contains EvalQuiz's full question specification that the implementation uses.

5.5.1 Monolithic Design of Question Specification

EvalQuiz processes and returns a single question specification, similar to Figure 5.3. The question specification contains all relevant information. Course settings **F-REQ-6**, generation settings **F-REQ-1** and evaluation settings **F-REQ-16**. We designed the question specification to provide defaults and optional settings. EvalQuiz accepts subsets of all specification parameters, skips steps not specified and falls back to default where necessary **NF-REQ-7**.

EvalQuiz writes its output in the configuration. An argument against a mutable configuration is, that the system should not change its own settings. This can introduce unexpected behavior. EvalQuiz writes output only in dedicated result sections for generation and evaluation. Those sections are

isolated and do not influence global settings or other sections. The advantage of the result sections is, that results are directly visible under their configuration scope. The direct mapping improves config readability and interpretability **NF-REQ-7**.

A monolithic config design benefits iteration **F-REQ-2**. The user can redirect output as input (Figure 5.3). The user has control over which parts to reiterate and which to keep. The global generation mode settings for example allow to fine-tune generation reiteration. Section B describes the generation mode in the question specification. The user cooperates with EvalQuiz and can partially reiterate the section of the question specification. Computational and memory overhead of unwanted reprocessing is prevented.

5.5.2 Question Specification Responsibilities and Usage

The question specification defines question types and generation/evaluation tasks **F-REQ-1 F-REQ-16**. It models the course context, standards and criteria **F-REQ-6**. The question specification is deliberately designed to not model individualized question creation, see requirement **F-REQ-25**. Individualized question creation delivers questions for each student (individual). A question is marked as suitable for specific students, see requirement **F-REQ-26**. We argue that questions generated through the question specification should match a wide audience already, speaking of the course's students. We assign the task of individualizing self-assessment to the learning management system, as this platform knows the student and their learning behavior best. This data must be made available for the self-assessment generation service, in order for EvalQuiz to use. Exporting learning management system data is undesirable, as data protection policies rightfully forbid sharing data with third parties without consent [Webm]. Even with consent, processing student behavior data through third-party APIs is questionable. Another challenge is embedding student data into the question specification. Encoding and presenting it to the language model is challenging. The language models max. size of tokens is a limitation, see. 5.3.

We see the responsibility for individualized teaching in the learning management system. The question specification is designed to delegate this responsibility to the learning management system. The question specification connects each question with capabilities, according to Section 5.3.2. Capabilities describe a course's standards/criteria (Section 2.1). We imagine an individualized approach to use standards/criteria that the course teaches as well as prerequisite standards/criteria. The approach models the student's mental model, depicted by Figure 4.1. The approach arranges standards/criteria in a hierarchical tree structure. The tree models standards/criteria as dependencies. Unknown dependencies in the student's mental model should be resolved first.

The evaluation module (Figure 5.2) allows augmenting a question's associated data. Additional attributes can contribute to individualizing self-assessment. Higher-quality questions should be preferred over lower-quality ones. Quality can be indicated by ZLW question-wording guidelines for example [Unia]. Diversification in question types paired with gamification has the potential to improve motivation through **Autonomy** and **Perception of Competence** [Unic].

6 Architecture and Implementation

We communicated and modeled the architecture according to the collected requirements, which Section 4 encapsulates. Frequent communication helps to gather feedback and create a clear picture before implementation. Mistakes can be identified early, reducing further complications that can lead to noticeable architectural changes during implementation. Section 6.1 explains the architecture model creation process. The architecture model describes pipeline modules, execution and data processing and EvalQuiz's domain concepts. The project implements the question specification as stated in the Appendix Section B. We highlight how the specification format changed throughout implementation in section 6.6.4. An approval meeting concluded the architecture planning phase.

The first part of the project implementation is the material server. Implementing the material server is a prerequisite for the pipeline server. We implemented each of the pipeline server modules in sequence. First the `MaterialFilter` (Section 6.6.1), then `QuestionGeneration` (Section 6.6.2) and `QuestionEvaluation` (Section 6.6.3). The question evaluation pipeline module is not to be confused with the evaluation of EvalQuiz itself which Chapter 7 discusses. Design decisions shape the implementation of each module. For example: the decision on which topic model to use (Section 6.6.1) and its integration with document filtering. The `MaterialFilter` converts lecture materials of different formats into a unified representation. Section 6.6.1 states supported document formats and the implementation process of material filtering. The `MaterialFilter` then filters documents of the unified representation (Section. 6.6.1).

The `QuestionGeneration` module formulates queries built upon the question specification, explained in Section 6.6.2. The queries are then processed through OpenAI's API [Webah]. Section 6.6.2 highlights message composition, API usage and converting the language models output to the question specification.

Implementation was more complex than expected. The project kickoff work packages and tasks do not include all necessary parts of the system. For example: API management and functionality are required for our evaluation survey (Chapter 7). The planning sketch does not account for the evaluation of the system through a client. A client is required for user interaction. The initial solution is a command-line interface as proposed by the interaction prototype, section 4.2.2. CLI is powerful for interacting with APIs for development or debugging but lacks friendliness for the average user. We reason in Section 6.7 about why we implemented a reactive frontend instead of a CLI. A reactive frontend is not the scope of the thesis originally, but usability benefits outweigh the additional effort.

The `QuestionEvaluation` pipeline module (Section 6.6.3) demonstrates advantages in automatic evaluation with our question specification 6.6.2. This module is also not part of the thesis' original scope.

6.1 Architecture Modelling

The architecture model was created in the architecture modeling phase. Changes made in implementation extend the original sketch.

The architecture model consists of three diagrams. The pipeline server (6.2), material server (Figure 6.1) and client (Figure 6.6). The pipeline server is responsible for the core functionality of EvalQuiz: Material filtering, question generation and question evaluation. A strong separation of technical and domain components is a key principle for the architecture models. The pipeline server can be reused for pipelines of other domains. Each server features a `ServerComponent` class exposing logic as a service. Classes representing datatype are shared among the architecture models. For example, the `InternalConfig`, which is the internal representation of the generation config. Pipeline modules in the pipeline server are modeled through the shared interface `PipelineModule`, as Figure 6.2 shows.

We separated the pipeline and material server to divide their respective responsibilities. The material server is responsible for saving, providing and indexing materials. Those materials are saved persistently and can be provided to multiple services. One of them is the pipeline server. This reduces redundant disk usage and increases versatility, adaptability and re-usability. Existing file storage solutions can use the interface defined by the material server and provide materials to the pipeline server. Other services, for example, a learning management system, can utilize the material servers interface as well.

A simple solution for resource management is a REST-compatible HTTP server. Endpoints can be defined with for example OpenAPI [Webal]. An advantage of REST is its wide support by existing web technologies. Path-based resource management comes with its availability drawbacks. Resource availability depends on the entity running the service. A path structure change can invalidate other paths.

There are multiple ways to provide redundancy for path-based web resources. Load balancing through reverse proxies or DNS service discovery enables redirecting requests to multiple servers [Gra]. However, DNS load balancing has major drawbacks as it delegates requests with round robin. There is no mechanism to filter out non-responding servers. Responding and non-responding servers are provided both through rotation [Gra]. DNS load balancing does not improve availability. Reverse proxies feature mechanisms to improve availability. Still, a reverse proxy configuration is a single point of failure, as all incoming requests have to go through the reverse proxy. A reverse proxy is also managed by a single entity. It is possible to create an organization managing the proxy. But then it is unclear how to divide costs and responsibilities.

We want to encourage cross-institutional teaching. Organizational, availability and redundancy concerns affect cross-institutional teaching. EvalQuiz's material server is a proof-of-concept sharing lecture materials through content-addressable storage, see section 6.5.1.

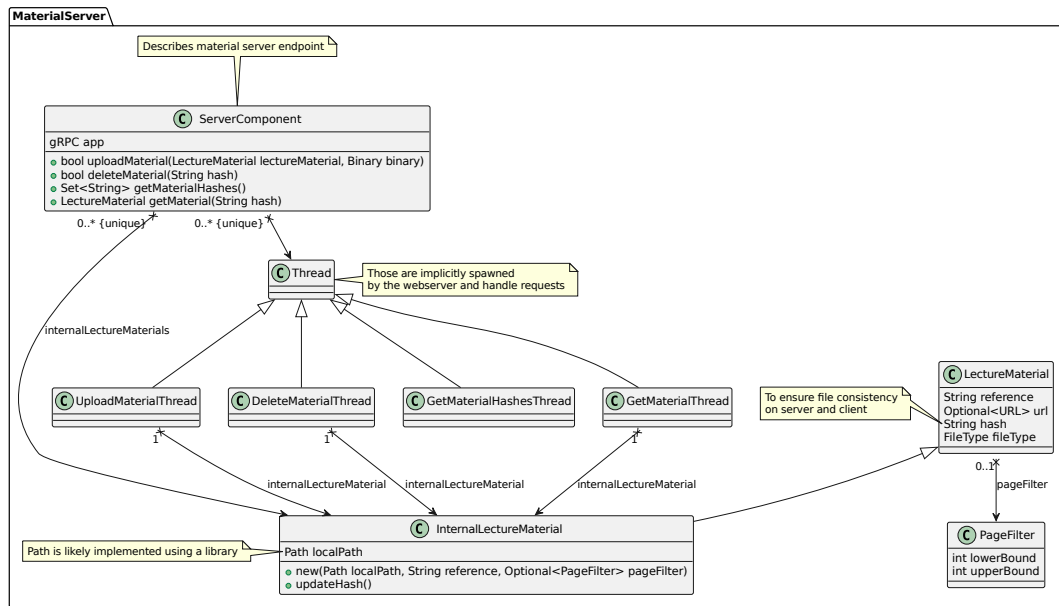


Figure 6.1: Architecture planning UML class diagram: Material server overview.

6.1.1 Material Server Architecture

The material servers `ServerComponent` handle incoming requests on endpoints to upload, delete and get materials (Figure 6.1). A new `Thread` handles each request and applies the operation on the `InternalLectureMaterial`. `InternalLectureMaterials` represent `LectureMaterials` saved under a `localPath`.

6.1.2 Pipeline Server Architecture

The pipeline server consists of domain-independent classes (Figure 6.3 part I). Domain-independent classes include request handling and pipeline execution. `EvalQuiz` domain-specific pipeline modules (Figure 6.4 part II) and the question specification in its class representation (`InternalConfig`) (Figure 6.5 part III).

The pipeline servers `ServerComponent` feature a single endpoint for question specification iteration (Figure 6.3 part I). The endpoint is `EvalQuiz` domain specific while the pipeline execution, modeled by the `PipelineExecutor` is domain independent. A new `PipelineThread` handles each incoming request as a `PipelineExecution`. A `PipelineExecution` is the execution of a `Pipeline`. Each `PipelineModule` is able to yield `PipelineStatus` instances, all streamed back to the client. `PipelineStatus` include error message for each `Batch` (Figure 6.5): `BatchStatus`. `BatchStatus` consist of one `ModuleStatus` and one `PipelineModule`. Each `Batch` runs on a `BatchThread`. `BatchThreads` run in parallel. Therefore each `Batch` needs its own `BatchStatus` to report about the `Batches` execution to `PipelineStatus`.

Figure 6.4 part II displays the question specification iterations `PipelineModules`. The diagram lists `PipelineModules` top to bottom in execution order. A subclass to represent the question specification iterations `Pipeline` is missing. The question specification iteration `Pipeline` matches

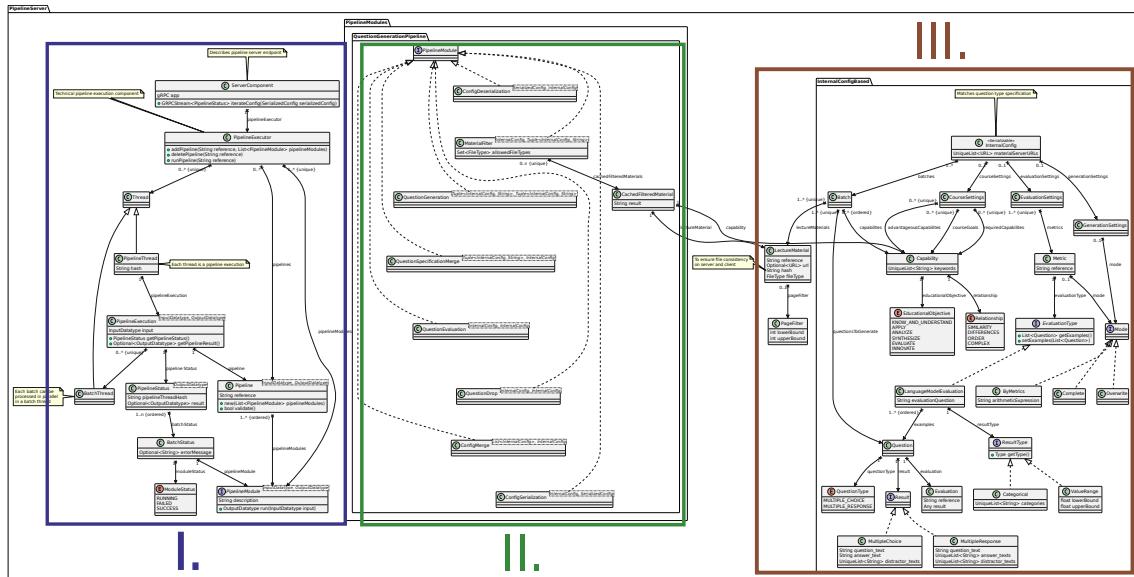


Figure 6.2: Architecture planning UML class diagram: Pipeline server overview.

the conceptualized pipeline in Chapter 5 Figure 5.1 except for a few additions. The QuestionDrop and ConfigMerge modules. The QuestionDrop module removes questions that fail QuestionEvaluation according to a Metric (Figure 6.5). ConfigMerge merges different InternalConfig (Figure 6.5) versions resulting of parallel Batch (Figure 6.5) execution. The MaterialFilter can cache LectureMaterials as CachedFilteredMaterials to save computational and memory resources.

Figure 6.4 represents an earlier version of the question specification as InternalConfig. Appendix Section B contains the latest version. The question specification iteration Pipeline processes the InternalConfig.

6.1.3 CLI Client Architecture

Remark

The CLI client is replaced by a reactive frontend. We describe the frontend implementation and its advantages over the CLI client in section 6.7.

The CLI client conceptualizes a command line interface (CLI) for interacting with the material and pipeline server. A user interacts with CLICommands methods through CLI, as Figure 6.9 part VI highlights. CLICommands uses the CLIClient to talk with both servers gRPC interfaces. Figure 6.9 part III is Figure 6.5 of the pipeline server as both components share the InternalConfig.

The CLIClient lists client stubs of the server endpoints (Figure 6.7 part IV). The InternalConfigController manages serialization and deserialization of InternalConfig. The InternalConfigController allows to update a InternalConfig with another The InternalConfig.

The InternalMaterialController references InternalLectureMaterials by their hashes (Figure 6.8 part V).

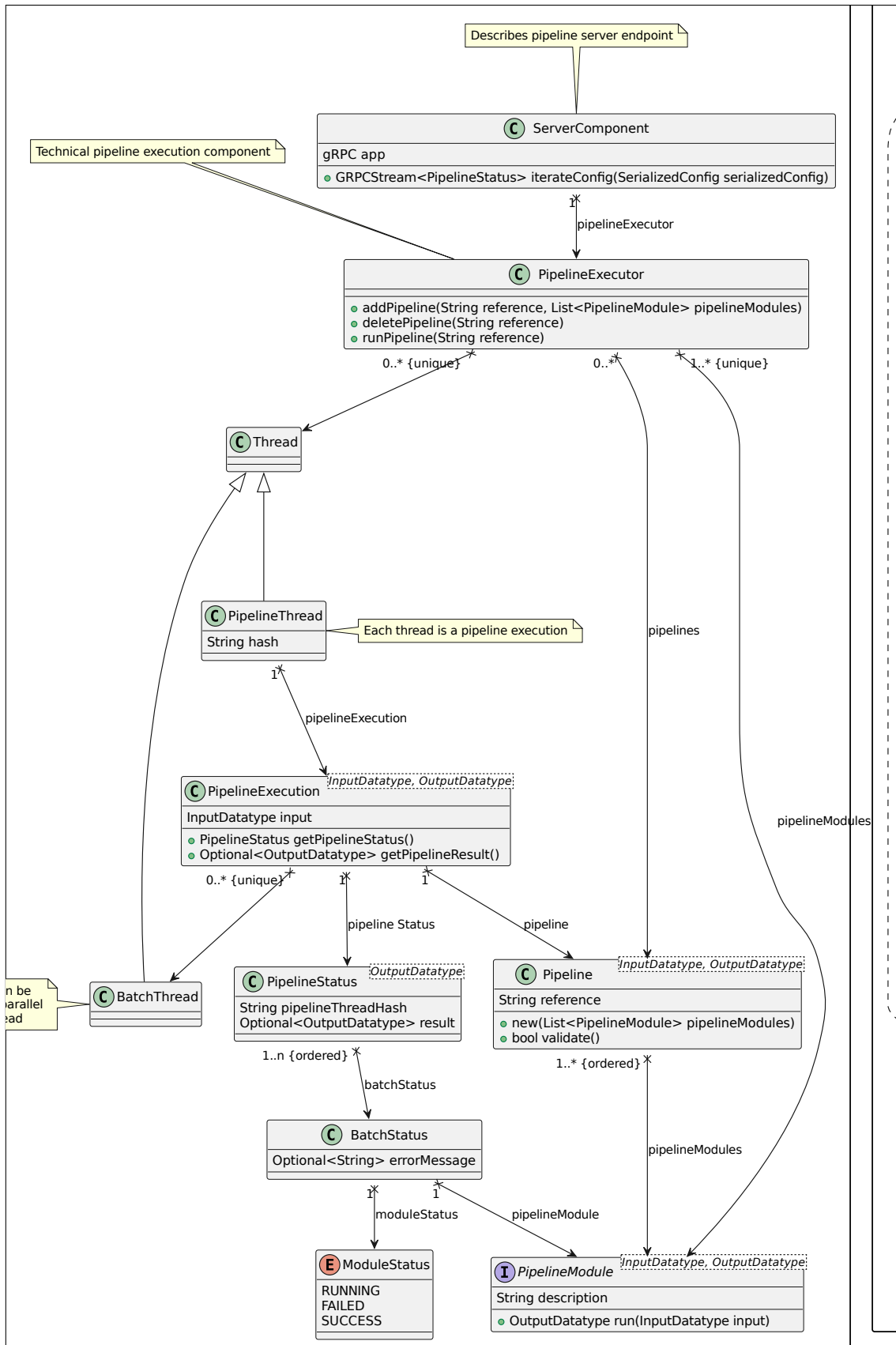


Figure 6.3: Architecture planning UML class diagram: Pipeline server I.

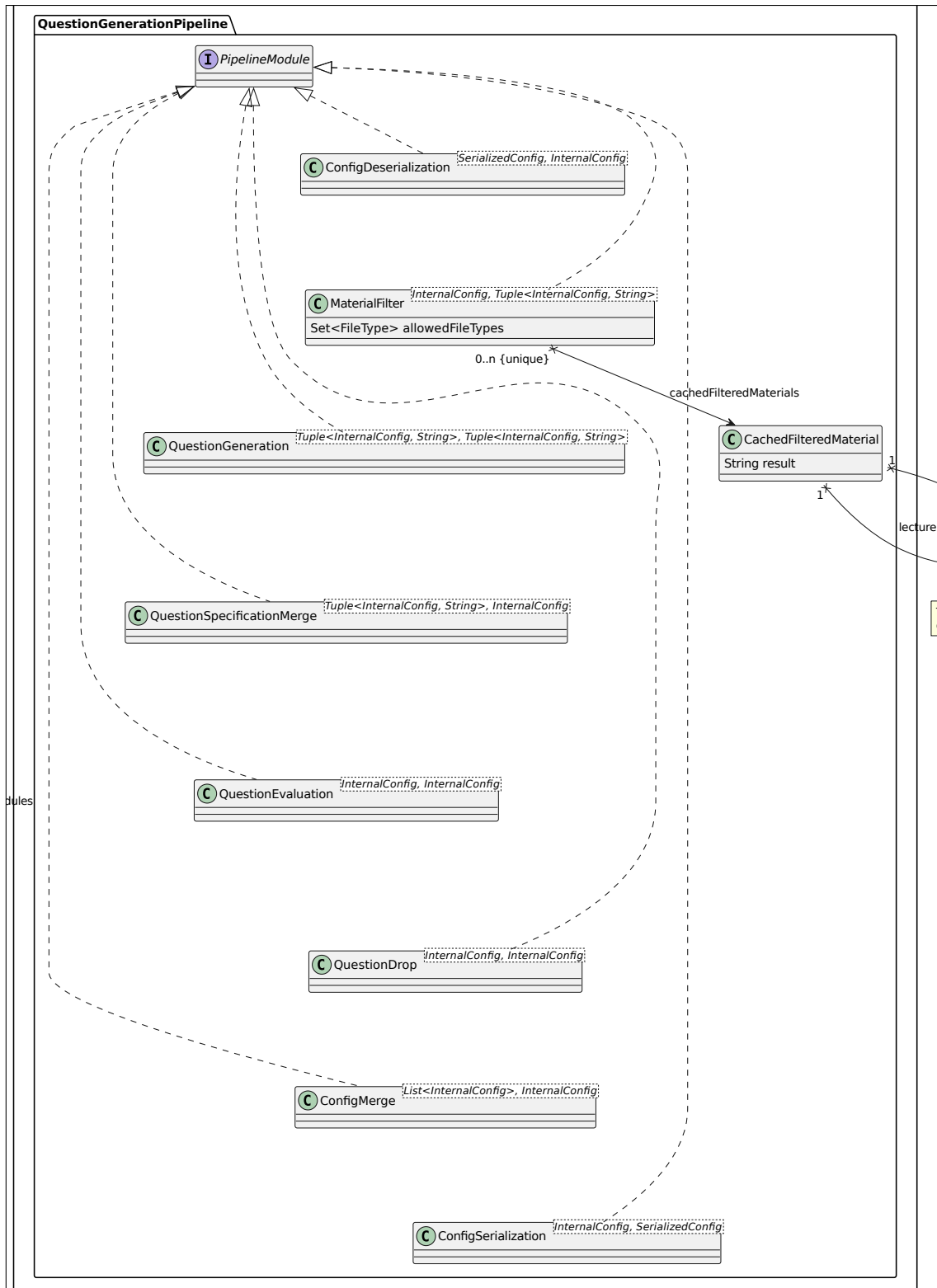


Figure 6.4: Architecture planning UML class diagram: Pipeline server II.

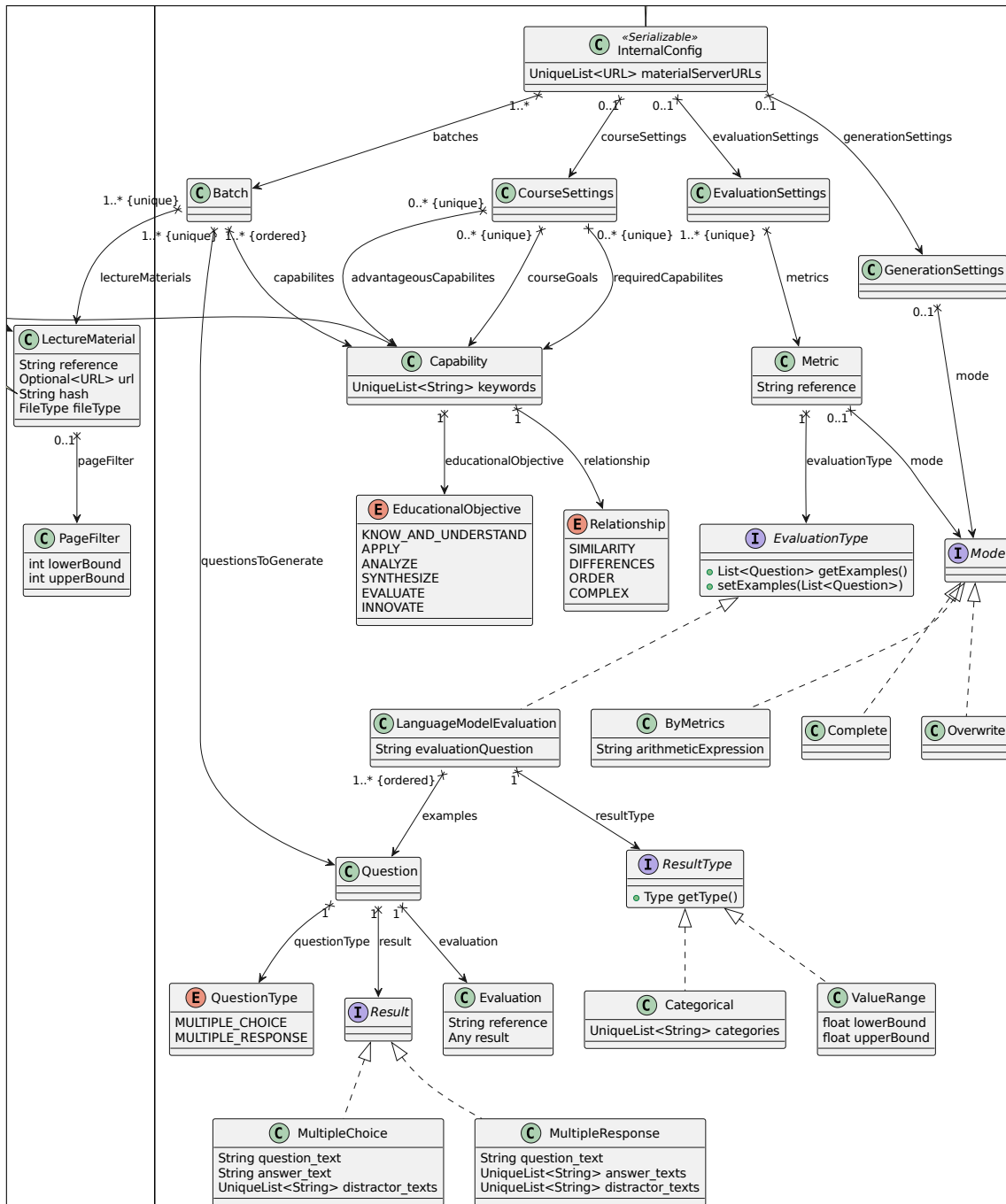


Figure 6.5: Architecture planning UML class diagram: Pipeline server III.

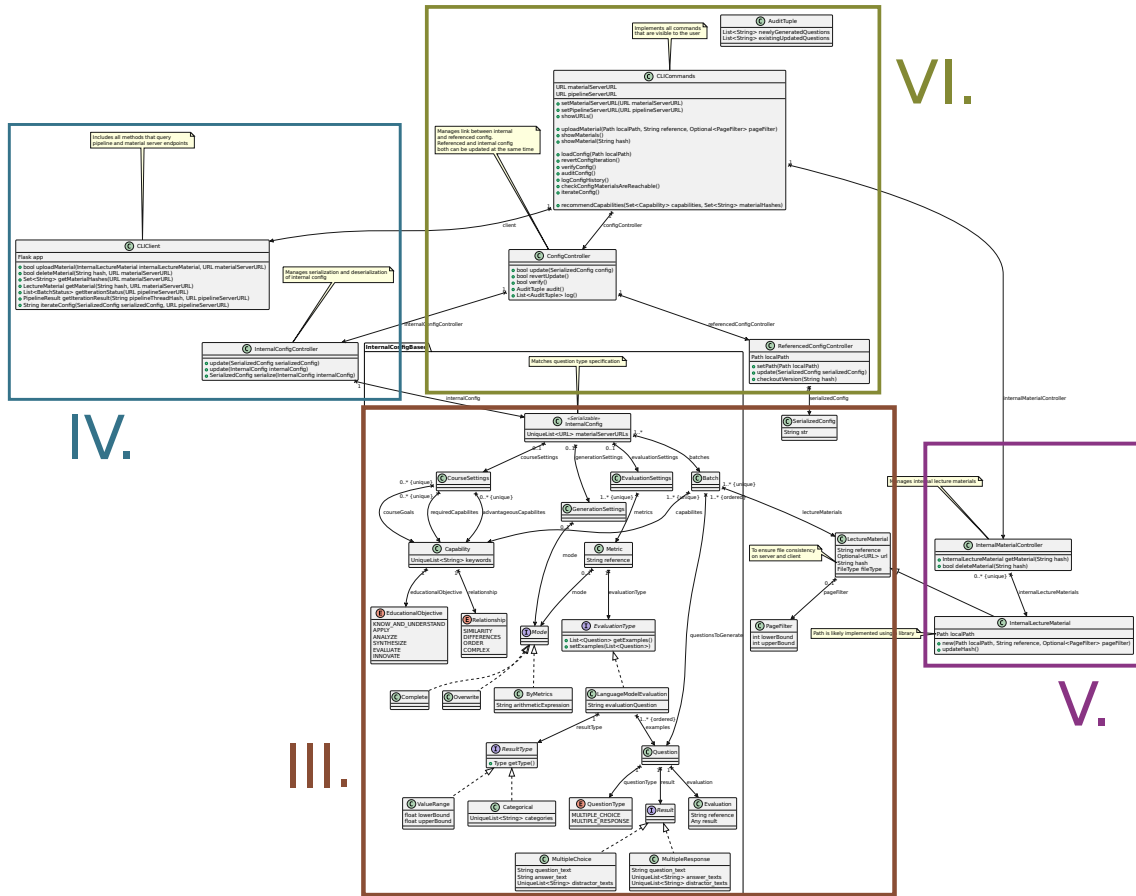


Figure 6.6: Architecture planning UML class diagram: CLI client overview.

A user sets a materialServerURL and pipelineServerURL to specify the server locations in CLICommands (Figure 6.9 part VI). CLICommands enables question specification history management through the ConfigController. ConfigControllers SerializedConfigController saves the question specification as SerializedConfig to disk and internally uses a version management library. ConfigController implements history and log upon SerializedConfig versions provided by the SerializedConfigController

6.2 Development Tools

- Main implementation language:** Python, a language suitable for machine learning (ML) research. Python enables easy access to a variety of datasets through **Hugging Face** [Tra], topic models with [Webo] and gRPC support with **Better Protobuf** [Webax]. Python is widely adopted in research and the industry. The Stackoverflow 2023 developer survey lists Python as the third most popular in the category "Programming, scripting, and markup languages" [Weba]. Python is deliberately chosen to ease further development and research on self-assessment generation with EvalQuiz. EvalQuiz's architecture and design focus on the separation of concerns and encourage reuse. Data classes, method annotations, datatype

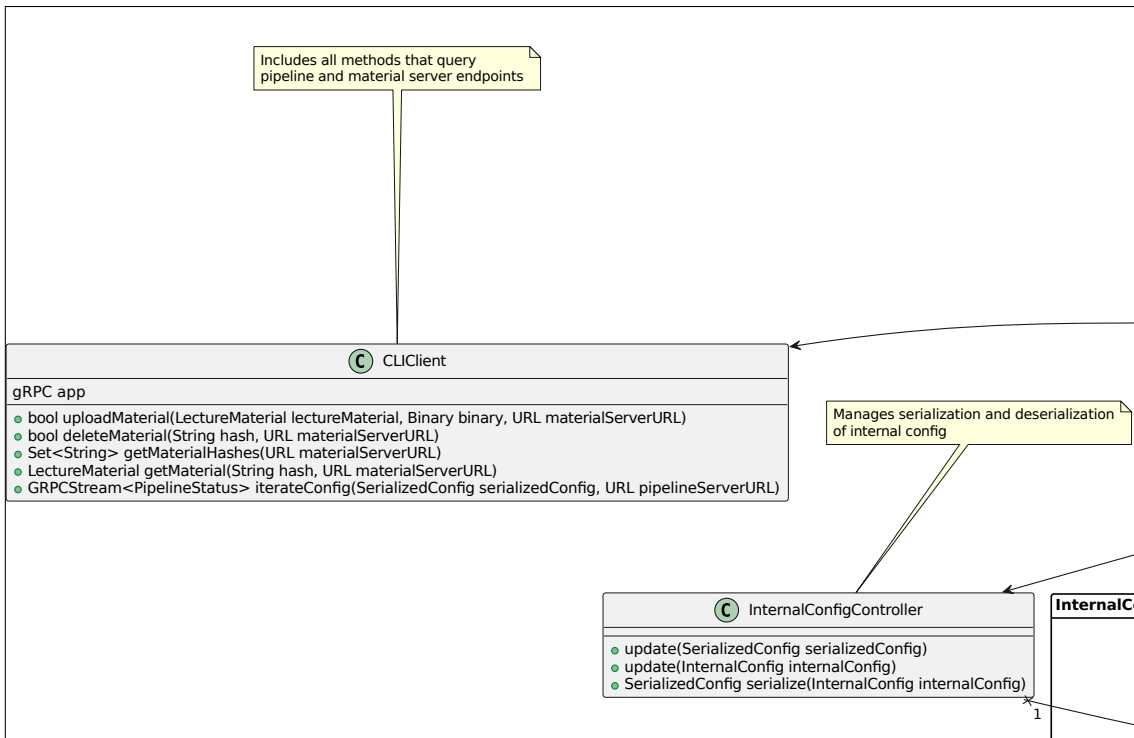


Figure 6.7: Architecture planning UML class diagram: CLI client IV.

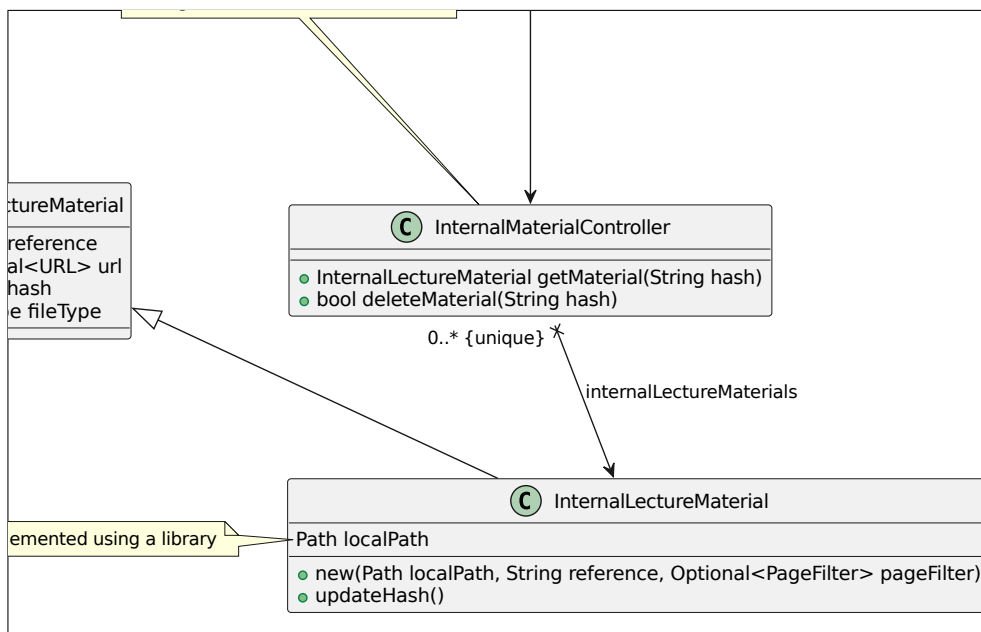


Figure 6.8: Architecture planning UML class diagram: CLI client V.

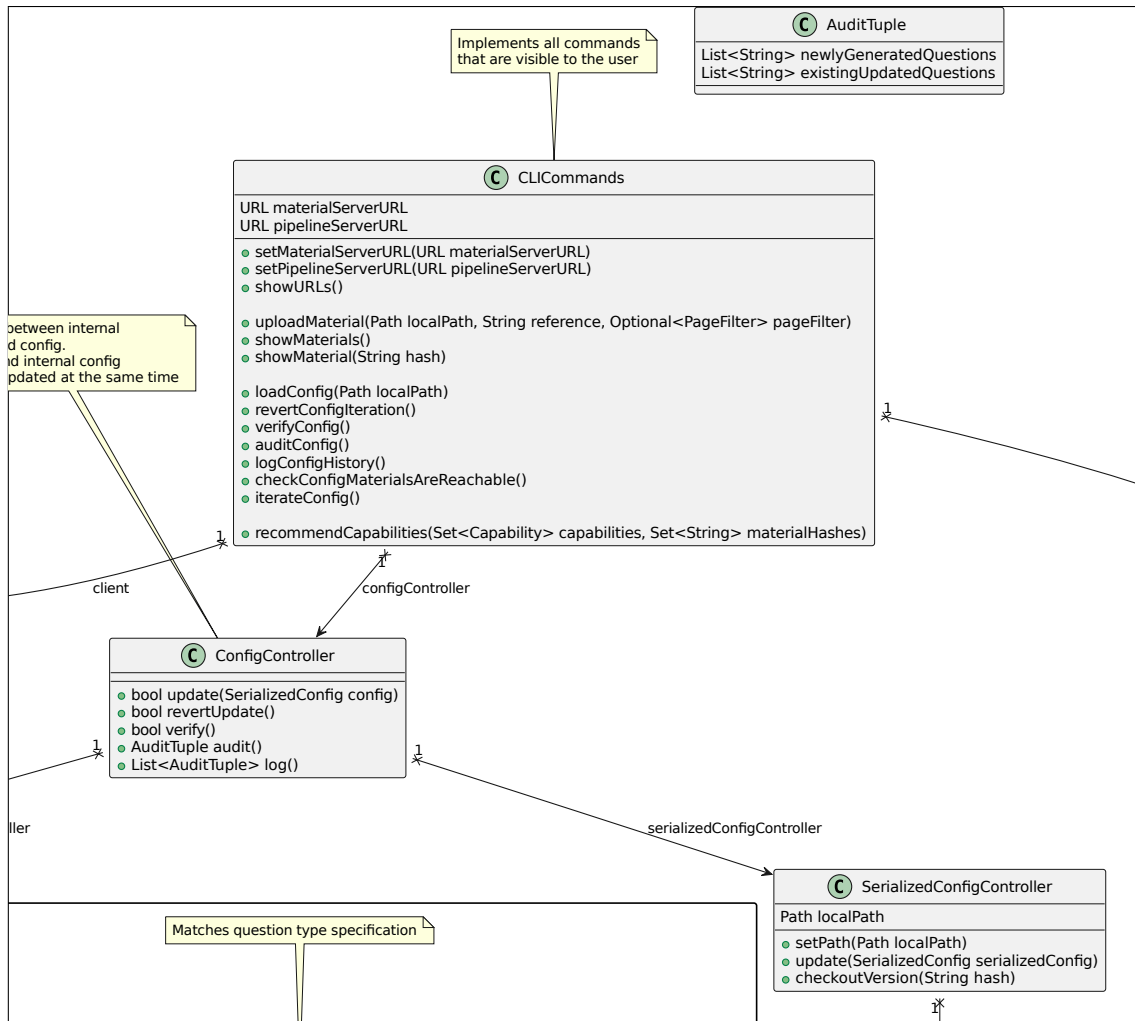


Figure 6.9: Architecture planning UML class diagram: CLI client VI.

comprehensions, etc., reduce boilerplate. Python features a less verbose syntax than other languages. Code readability is one of Python's core principles. Python's ecosystem consists of a large package index (PyPI [Webay]) available through package managers as pip [Weban].

Python is general-purpose and dynamically typed. CPython is Python's reference implementation [Webg]. It compiles Python to an intermediary format executed by a virtual machine [Webg]. CPython manages memory through reference counting and garbage collection [Webg]. System-level programming languages such as Rust [Webbc] or C completely compile code to binaries. Compilation in combination with static typing often results in a smaller package size and memory footprint during execution. EvalQuiz does not require system-level performance. We argue that the EvalQuiz's pipeline runs into IO instead of computational bottlenecks. Language model queries through OpenAI's API are rate limited [Webah]. The `MaterialFilter` module converts documents through invocations of Pandoc [Webam]. Python's global interpreter lock (GIL) prevents code from running concurrently. However, we do not see the GIL as a problem, as concurrent programming adds little benefit as discussed and possibly causes thread safety issues.

Python's optional typing enables fast prototyping. Types do not need to be explicitly stated, reducing verbosity. Optional typing does not require shared interfaces. Objects of different structures can share a method signature and be used interchangeably. Fewer type casts are necessary. Lessons learned through prototypes influence implementation, as with our self-assessment generation 5.2.2 and evaluation prototype 6.6.3.

- **Editor:** VSCode [Webbk], the most popular IDE [Webba]. EvalQuiz utilizes VSCode Development Containers to provide reproducible development environments.

6.3 VSCode Development Containers

EvalQuiz's Python repositories utilize VSCode Development Containers. VSCode attaches to a Docker container. Docker mounts the application files to the container. This is important as Docker containers are volatile. Removing or recreating the container deletes non-persistent data [Webj]. VSCode allows one to attach one container from a Docker Compose configuration. Docker Compose support is essential, as it allows to test database interaction within the development environment. The material server interacts with MongoDB [Webau]. Unit tests ensure that transactions between the material server and MongoDB function as expected. We use Git and GitHub to version the project and implement CI/CD. Section 6.8 goes into detail about the CI/CD setup.

6.4 Server Interfaces

An OpenAPI configuration initially defined server interfaces with REST [Webal]. REST is a resource-based paradigm for interfacing microservices. The advantages of REST are its wide adoption and standardization and the wide variety of tools available. Reflection upon the usage of our services revealed that a remote procedure call (RPC) paradigm fits our needs better. The pipeline server originally provided two endpoints. One for iterating a generation configuration and the other for querying the iteration result status. This creates two problems. A reference is required which must be returned upon calling an iteration. The status then can be queried using the reference. Periodically querying the reference for a pipeline status update is busy waiting. Busy waiting creates an additional load on the server and client. Server-side streaming provided by gRPC solves this problem. PipelineStatus objects are directly streamed to the client. The material server benefits from streamed material uploads and downloads, reducing RAM usage of the server and client. The implementation does not hold the complete file in memory before writing it to disk.

Material server, pipeline server and react client use the question specification and its data types. All three repositories share a Git sub-module containing the question specification as a Protobuf schema. Better Protobuf is used to generate Python classes and server/client stubs from the Protobuf Schema [Webax]. The servers communicate using the gRPC protocol.

6.5 Material Server

This section describes the implementation of the material server. We present the material server's architecture in Section 6.1.1.

6.5.1 Content-Addressable Storage

EvalQuiz uses content-addressable storage (CAS) instead of path-based storage [TKS+03]. The material server manages materials. A material is addressable through the hash of its contents in a pathless and deterministic way. The material server implementation ensures that a material matches the hash under which it is requested. Multiple entities can host material servers. Each entity does not need to be associated with the other entities. Material servers can be hosted by institutes of different universities for example. The user decides from which entity to request materials. The material server specification and implementation are independent. Material servers can host arbitrary files. We encourage reuse for different domains.

We illustrate how materials can be shared and reused with EvalQuiz. A user configures a question specification. The pipeline server's `MaterialFilter` module queries materials according to the question specification. The pipeline server resolves materials in the following order: A material, as defined by the question specification (section B), has an optional `url` field. If set, the material is queried from `url`. If `url` is undefined, global configurations apply. The `MaterialFilter` module then requests `materialServerURLs`. The `MaterialFilter` traverses `materialServerURLs` in order. The order is less important as materials are content-addressable. Receiving the wrong material is improbable if the material server is correctly implemented.

The interplanetary file system (IPFS) provides distributed content-addressable resource management [Webw]. IPFS supports request resolution through request delegation. The requested IPFS node does not need to hold the requested file. It delegates the request to a node with the file if it exists. IPFS nodes delegate requests to other nodes in the same IPFS cluster [Webw]. Material servers can be replaced with IPFS or other content-addressable storage systems in future work. EvalQuiz does not use IPFS due to practical limitations. Setting up an IPFS cluster is out-of-scope for this thesis. We believe that content-addressable storage has the potential for collaboration between institutes in teaching.

6.5.2 Hash function: BLAKE3 [Webf]

BLAKE3 is a cryptographic hash function implemented in Rust [Webf] [Webbc]. BLAKE3 is used for material hashing with its default output size of 256 bits. EvalQuiz conducts hash consistency checks. Section 6.1 explains how hashes enable content-addressable lecture materials. BLAKE3 matches performance and collision prevention requirements for EvalQuiz's hash addressing [Webf]. Figure 6.10 displays a benchmark of BLAKE3 against other cryptographic hashes, where higher is better.

6.6 Pipeline Server

This section describes the implementation of the pipeline server. We present the pipeline server's architecture in Section 6.1.2.

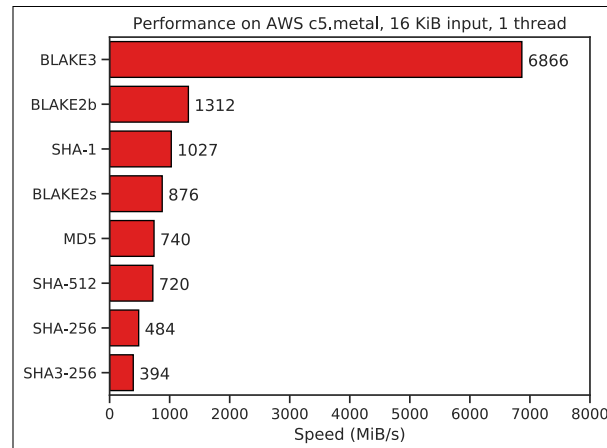


Figure 6.10: BLAKE3 benchmark against state-of-the-art cryptographic hashes: Source: O'Connor et al. [Webf].

6.6.1 Material Filter Pipeline Module

The `MaterialFilter` consists of the `MaterialClient`, `MarkdownConverter` and a `TextExtractor`. The `MaterialClient` is responsible for querying materials from material servers specified by the question configuration. Materials are then converted to markdown by the `MarkdownConverter`. This component uses Pandoc [Webam], an open-source universal document converter. The PyPandoc Python library enables direct integration into Python and also provides a package including Pandoc's binary. All formats Pandoc can convert to markdown are supported as `MaterialFilter` input. Pandoc, unfortunately, does not support .pdfs. We decided to add .pptx support in order to increase our approach's coverage of commonly used lecture material types. In summary, `EvalQuiz` supports files of the following extensions: .md, .pptx, .csv, .tsv, .docx, .epub, .html, .ipynb, .json, .latex, .markdown, .man, .odt, .opml, .org, .ris, .rtf, .rst, .tex. Future versions can extend the list of supported files.

Markdown is the unified representation ready for filtering and later conversion to self-assessment. The `TextExtractor` interface provides core material filtering. `EvalQuiz` features one `TextExtractor` implementation. The `TextExtensionTextExtractor` is implemented upon Algorithm 5.1. Keywords given through `Capability` instances are extended with 5 extra keywords per `Capability`. The topic model infers additional keywords. Finding related terms for each `Capability`, instead of all concatenated keywords, enables us to extend upon the concept described in the specific `Capability`. In the following the input text is matched against all keywords. Each sentence is kept if it contains at least one of the keywords. A max. token length of 1000 tokens is chosen to manage text size for the following generation step. There is a currently not utilized option to restrict the `max_keywords` resulting from the expanded keyword list (5 for each `Capability`). The keyword list is truncated, if `Capability` instances keywords + extended keywords > `max_keywords`. The `TopicExtensionTextExtractor` sorts the list of expanded keywords by their similarity. `max_keywords` truncates keywords less similar to the `Capability` instances. The values of 5 extra keywords per `Capability` and a max. token length of 1000 tokens are chosen arbitrarily.

Choosing a Topic Model

Material filtering requires a topic model capable of online training. A topic model describes relationships between words. Words outside of the model's vocabulary lack the connections required. Online training is necessary to insert connections. We expect lecture materials to contain unknown words, even for large topic models. University subjects are highly specialized and often feature technical terms. Another constraint is that the topic model needs to be capable of calculating a N -Similarity function. N -Similarity is a function calculating N similar words to a bag of words. The bag of words needs to be part of the topic model's vocabulary. Therefore `Capability` instances referring to terms not contained in the topic models vocabulary or material filters input text cannot be processed. Even if the words are synonyms. Furthermore, training and inference need to happen in a few seconds. Longer processing times impact the system's responsiveness. We chose `Word2Vec` as our topic model, which satisfies the stated requirements [Chu17]. `Word2Vec` works on a word-vector representation. A word-vector model encodes words in a multidimensional vector space. Similarity is modeled by proximity calculated with a distance metric. Commonly known distance metrics are cosine similarity, euclidean distance or hamming distance (for binary vector spaces). Word-vector models enable N -Similarity by design. `Word2Vec` works with cosine similarity [Chu17]. The Python library `Gensim` provides different topic models, one of which is `Word2Vec`.

Custom Trained Word2Vec Model

We trained a `Word2Vec` model in order to provide a large scientific vocabulary. Training data is the "CShorten/ML-ArXiv-Papers" data set provided through the Huggingface Python library 6.1 [Webad]. The data set consists of 100000 papers filtered from the Cornell University arXiv data set, featuring approximately 1.7 mio. papers. The papers are from different science technology engineering and mathematics (STEM) fields.

Remark

Interface is used to describe a Python ABC class with only abstract methods interchangeably.

Portable Document Format (PDF) Conversion Drawbacks

PDF is a commonly used document format for lecture materials. The advantages are the widespread support and size. Raw document formats from text processing software can be easily converted to PDF. PDF converts raw text to a content stream and creates vector graphics for document elements [Webap]. Raster graphics can also be embedded [Webap]. Converting PDF to text processing formats is challenging. Text processing formats are common formats in which text documents are edited. For example: `.docx`, `.pptx`, `.tex`, `.md`. Wang et al. present an ML approach for converting PDF back to \LaTeX math (PDF2LaTeX) [WL20]. The system's 81.1% character recognition performance makes it hard to recommend for real-world projects. We require the conversion of whole documents from PDF to a text processing format. `Pandoc` can convert most text processing formats into each other [Webam].

Tests with Microsoft Document Intelligence Studio "Read" recognition concludes that simple text recognition works. More complex document structure, graphics and \LaTeX math on the other hand is lost [Webab]. Document Intelligence Studio provides fixed-format document recognition.

Fixed-format document scanning enables faster processing of bureaucracy, not applicable to general lecture materials. Mathpix offers a commercial document conversion solution. It features full document conversion from PDF to \LaTeX . Simple tests with Mathpix demonstrate its superiority over other approaches. The recompiled document looks nearly indistinguishably from the original PDF, as shown by Figure 6.11 and Figure 6.12. Mathpix also offers a document conversion API. Mathpix currently is in a niche market position reflected in its service pricing. Converting large documents repeatedly with Mathpix is not feasible for EvalQuiz. Future work is needed to fill the gap between PDF2LaTeX to Mathpix with a capable open-source approach.

We now get to the output layer, where we now have ten responses rather than one. The first step is to compute ten different linear models similar to our single model (10.1),

$$\begin{aligned} Z_m &= \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} h_{\ell}^{(2)}(X) \\ &= \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_{\ell}^{(2)}, \end{aligned} \tag{10.12}$$

for $m = 0, 1, \dots, 9$. The matrix \mathbf{B} stores all $129 \times 10 = 1,290$ of these weights.

Figure 6.11: Mathpix conversion text: Source PDF: James et al. [JWHT+13].

We now get to the output layer, where we now have ten responses rather than one. The first step is to compute ten different linear models similar to our single model (10.1),

$$\begin{aligned} Z_m &= \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} h_{\ell}^{(2)}(X) \\ &= \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_{\ell}^{(2)}, \end{aligned}$$

for $m = 0, 1, \dots, 9$. The matrix \mathbf{B} stores all $129 \times 10 = 1,290$ of these weights.

Figure 6.12: Mathpix conversion text: Compiled \LaTeX from converted PDF, source: James et al. [JWHT+13].

Lecturers using EvalQuiz are encouraged to use text processing formats compatible with Pandoc [Webam] or .pptx files. .pptx is supported through PPTX2MD [Webaq]. The quality of the conversion is sufficient, judging from simple conversion tests (Figure 6.13). PPTX2MD fails to convert some of the graphics and the source PPTX. Missing graphics are not a problem. EvalQuiz currently does not utilize the image processing capabilities of GPT-4. Image processing can be an extension to improve question generation, further details in section 6.6.1. We argue that PDF is not mandatory for our system. Lecturers usually have lecture materials in text-processing formats. Lecturers need to edit lectures as content evolves or mistakes appear.

Instanceof

- Was wenn dennoch auf spezielle Klasse reagiert werden muss?
- In der Regel lassen sich Klassen so entwerfen, dass dies nicht notwendig ist
 - bellen() und miauen() ist beides eine Art von lautGeben() (Überoperation finden)
- ansonsten instanceof
 - testet ob ein Objekt vom gegebenen Typ ist
 - Achtung: eine benötigte Verwendung von instanceof weist meistens auf ein Fehler im Design der Vererbungshierarchie hin

```

public class Main {
    public static void main(String[] args) {
        // Das ist keine Instanz von Pet!
        Pet[] pets = new Pet[]{};
        pets[0] = new Dog();
        pets[1] = new Cat();
        for(Pet p : pets) {
            if(p instanceof Cat) {
                ((Cat)p).miauen();
            } else if(p instanceof Dog) {
                ((Dog)p).bellen();
            }
        }
    }
}
    
```

9. Oktober 2023 36

Programmierung & Softwareentwicklung - Vortragsübung - Vererbung

Aufgabe

- Gegeben sei folgende Code Stücke, sowie die normale Hamster Klasse:

```

class ChattyHamster extends Hamster {
    @Override
    public void move() {
        super.write("I'm walking ...");
        super.move();
    }
}

class SelfishChattyHamster extends Hamster {
    @Override
    public void move() {
        super.write("I'm walking first!!! Others second ...");
        super.move();
    }
}

class PolymorphicHamsterGame extends SimpleHamsterGame {
    @Override
    protected void run() {
        game.displayInGameWindow();
        game.initialize();
        final Hamster paule = new Hamster("...");
        final Hamster marry = new ChattyHamster("...");
        final Hamster willi = new SelfishChattyHamster("...");
        List<Hamster> hamsters = Arrays.asList(paule, marry, willi);
        for (final Hamster hamster : hamsters) {
            hamster.move();
        }
    }
}
    
```

Wie lautet die Ausgabe des Programms?

9. Oktober 2023 37

© Sandra Speth Programmierung & Softwareentwicklung - Vortragsübung - Vererbung

Instanceof

- Was wenn dennoch auf spezielle Klasse reagiert werden muss?
 - In der Regel lassen sich Klassen so entwerfen, dass dies nicht notwendig ist
 - bellen() und miauen() ist beides eine Art von lautGeben() (Überoperation finden)
 - ansonsten instanceof
 - testet ob ein Objekt vom gegebenen Typ ist
 - Achtung: eine benötigte Verwendung von instanceof weist meistens auf ein Fehler im Design der Vererbungshierarchie hin

Programmierung & Softwareentwicklung - Vortragsübung - Vererbung

Aufgabe

Gegeben sei folgende Code Stücke, sowie die normale Hamster Klasse:

Wie lautet die Ausgabe des Programms?

```

class ChattyHamster extends Hamster {
    @Override
    public void move() {
        super.write("I'm walking ...");
        super.move();
    }
}

class SelfishChattyHamster extends Hamster {
    @Override
    public void move() {
        super.write("I'm walking first!!! Others second ...");
        super.move();
    }
}

class PolymorphicHamsterGame extends SimpleHamsterGame {
    @Override
    protected void run() {
        game.displayInGameWindow();
        game.initialize();
        final Hamster paule = new Hamster("...");
        final Hamster marry = new ChattyHamster("...");
        final Hamster willi = new SelfishChattyHamster("...");
        List<Hamster> hamsters = Arrays.asList(paule, marry, willi);
        for (final Hamster hamster : hamsters) {
            hamster.move();
        }
    }
}
    
```

(a) PPTX2MD conversion text: Source PPTX: Speth (b) PPTX2MD conversion text: Compiled markdown from converted PPTX, source: Speth et al. [Spe].

Figure 6.13: Comparison of original PPTX and converted markdown.

Image Processing

EvalQuiz currently does not support image processing. But the system is designed with this future use case in mind. The MarkdownConverter converts lecture materials used in EvalQuiz to markdown. The unified representation. This withholds advantages in terms of what the language model can expect. A markdown image syntax describes images embedded in the lecture materials: .

Images converted by Pandoc [Webam] or PPTX2MD [Webaq] extract images from the respective input file. Both converters save the image in the same folder as the output markdown file. Image processing requires a further intermediate step, currently not implemented. Extracted images need to be uploaded to a material server. The algorithm replaces original image names (image_name) with the image hashes (f9f75c3c05c99d69364ae75e028c997fb1a8c209e03a6452efbef6b75784c3ab). The material server needs to be referenced in the question specification. Alterations in the QuestionGeneration pipeline modules MessageComposer have to be made. The MessageComposer matches markdown image tags and extracts hashes in the material names. The MessageComposer resolves image dependencies from the referenced material servers and inserts images as messages. The transformer model can interpret and process images, if capable. GPT-4 is multi-modal and supports image input [Ope23].

6.6.2 Question Generation Pipeline Module

The `QuestionGeneration` module takes a question configuration and filtered text as input. Each Batch is processed in isolation and has a list of `QuestionToGenerate`. The modules generate questions in multiple steps. First, the module decides if generation of the `QuestionToGenerate` is necessary. Section 6.6.2 gives details about reprocessing decisions. Then the `MessageComposer` composes messages. Those messages are then processed by an `LLMClient`. The `model` field in the question configurations `GenerationSettings` decides which `LLMClient` is queried. We support GPT-4 in its default variant, `gpt-4` [Ope23]. The `LLMClient` interface simplifies adding other language models. This includes a locally run language model. A new `LLMClient` implementation can be provided through the `LLMClientRegistry`. `QuestionGeneration` accesses `LLMClient`s through the `LLMClientRegistry`.

After retrieving a result, `QuestionGeneration` parses the result. A regular expression extracts the generated question from the result tags: `<result type=generation></result>`. The results tags contain a question in the specified question type format. The language model is free to comment on its generation outside the result tags. Our prototypes: The question generation prototype and evaluation prototype show that GPT-4 output is often verbose. Section 5.2.2 and 6.6.3 highlight both prototypes respectively. We assume that reinforcement learning with human feedback (RLHF) encourages GPT-4 to prefer longer outputs. Section 2.3.1 explains RLHF. Ranking the output variants in Step 2 of Figure 2.4 influences a language model to prefer specific outputs. We assume that a labeler ranks output variants with additional context information higher, as it helps to understand the model outputs better. The model learns to produce explanatory outputs. The `QuestionGeneration` module parses the output. Explanations are unimportant for `QuestionGeneration`. The result must be correct and in the right format. Extracting generated questions from result tags discards explanation attempts. `QuestionGeneration` reads the `QuestionToGenerates` `QuestionType` and parses the extracted text accordingly.

Message Composer

The `MessageComposer` creates a list of dictionaries representing messages in OpenAI's message format [Webr]. Composed messages consist of a system message, few-shot examples and a query message, shown in Table 6.1.

The system message starts with:

```
You are a question-generation assistant that supports generating questions in multiple fixed formats.
```

```
The question generated by you serves the purpose of helping a student to self-assess, which skills they have acquired.
```

```
You can assume that the student already has acquired the following skills:
```

The `MessageComposer` then lists `required_capabilities` in `CourseSettings` and continues with:

```
Here is more information about the ALL_CAPS formatted instructions used in the skill descriptions:
```

Type	Role	Content
System message	system	Instructions for the assistant
Few-shot example 1	user	MessageComposer query message
Few-shot example 1	assistant	Example assistant response
...
Few-shot example $n - 1$	user	Example user input
Few-shot example $n - 1$	assistant	Example assistant response
Query message	user	MessageComposer query message

Table 6.1: Structure of messages created by Message Composer.

The MessageComposer then explains EducationalObjectives used as acquired skills, in few-shot examples or the query message.

Messages following the system message are few-shot examples. Few-shot examples are of the same QuestionType as the QuestionToGenerate. Few-shot examples represent how the generated output should look like. The QuestionTypeComposer is responsible for providing few-shot examples to the MessageComposer. The MessageComposer then creates messages out of the few-shot examples. A FewShotExample contains a Question, Capability list, filtered text (str) and GenerationResult. The MessageComposer composes the respective user and assistant messages with the QuestionType specific QuestionTypeComposer. The MessageComposer adds all n few-shot examples.

User messages start with:

Your goal is to use the given markdown formatted text input to generate a question in the following JSON format:

The MessageComposer then uses QuestionTypeComposer to explain the QuestionType.

Give your answer in the specified JSON format at all costs!

A student who can answer the generated question successfully should have acquired the following skill set:

The MessageComposer then lists the QuestionToGenerates Capability instances and continues with:

Double-check that the question supports strengthening the previously given skills.

Markdown formatted text input:

```
```md
```

The MessageComposer then inserts the filtered text (str) and closes the markdown environment:

```
```
```

The `QuestionTypeComposer` implemented for each `QuestionType` creates a result template with placeholders for question components. For example: `MultipleChoice` has the `GenerationResult` placeholders: `QUESTION_TEXT`, `ANSWER_TEXT`, `DISTRACTOR_TEXT_1`, `DISTRACTOR_TEXT_2`, `DISTRACTOR_TEXT_3`. This result template is part of a query message describing the placeholders. The query message explains the `QuestionType`.

Implementation of New Question Types

`EvalQuiz` is extendible for new `QuestionTypes`. Adding a new `QuestionType` requires implementing and registering a `QuestionTypeComposer` to the `MessageComposer`. Therefore the `QuestionType`, a `GenerationResult` with placeholders for the `QuestionTypes` fields and `FewShotExamples` need to be provided. Registering the `QuestionType` is as easy as adding it to the `question_type_composers` dictionary of `MessageComposer`.

The question specification needs to be changed to support the new question type. All `EvalQuiz` components (Figure 6.16) have to adapt to the new version including the question type. Changing the Protobuf schema suffices. Rebuilding all Protobuf-based artifacts propagates the newly added question type to the components. This includes recompiling the frontend `JSONSchema`. Section 6.7.3 states the recompilation `JSONSchema` process and its difficulties.

Question Reprocess Decider

The `QuestionReprocessDecider` implements the question specifications mode setting 5.5.1. The `QuestionReprocessDecider` returns a boolean dependent on the mode and `Question`. The boolean indicates if a specific question has to be reprocessed (regenerated or reevaluated). Evaluation results influence the boolean if the `by_metrics` is chosen. The `QuestionReprocessDecider` decides upon `by_metrics` conditionally. If the `Question` satisfies the metric, it should be reprocessed. The `QuestionReprocessDecider` returns `True`.

The `QuestionReprocessDecider` enables self-assessment quality control automation. It allows one to regenerate a question until it meets a specific metric, describing a qualitative standard for example.

Reprocessing describes regeneration. Metrics specified in `by_metrics` can influence regeneration. Additionally, conditional reevaluation is possible. For example, if one evaluation fails, then further evaluations can provide more insight. The additional evaluations are not always necessary, saving computational and memory resources. The user is able to build complex decision trees on regeneration and reevaluation. The `by_metrics` mode option is optional. Generation and evaluation does not require `by_metrics`.

6.6.3 Question Evaluation Pipeline Module

The `QuestionEvaluation` pipeline module is structured similarly to the `QuestionGeneration`. Section 6.6.2 highlights its similar usage of the `QuestionReprocessDecider`. `InternalEvaluations` allow the `QuestionEvaluation` to implement different evaluation logic. An evaluation has its own configuration parameters in the question specification. Our reference implementation of `InternalEvaluation` is `InternalLanguageModelEvaluation`. This implementation uses its own

MessageComposer for composing messages and the LLMClient interface to access language models. The MessageComposer is different from the QuestionGenerations MessageComposer, as it composes a different message. Result parsing works analogously to QuestionGeneration. Result tags are named, as QuestionEvaluation utilizes both `<result type=generation></result>` and `<result type=evaluation></result>` tags. QuestionGeneration works without a type parameter, as it only utilizes `<result type=generation></result>`.

Message Composer

The QuestionEvaluations MessageComposer creates the same structure of messages as shown in Table 6.1.

The system message is:

You are a question evaluation assistant who supports evaluating questions in multiple fixed formats.

The evaluation generated by you serves the purpose of helping a teacher to gather more insight into a specific question.

User messages start with:

Your goal is to evaluate the question in the following JSON format:

The QuestionEvaluations MessageComposer then uses EvaluationTypeComposer to explain the EvaluationType.

Description of how the question should be evaluated:

The MessageComposer then appends the InternalLanguageModelEvaluation evaluation description.

Question to evaluate:

The QuestionEvaluations MessageComposer then uses the GenerationResultTemplate to write the Questions GenerationResult as JSON in `<result type=generation></result>` tags.

Evaluation Prototype

The evaluation prototype assesses if the evaluation as conceptualized is feasible. Two result types, according to the question specification, are tested: `result_type: value_range: [0, 1]` and `result_type: categorical: ["True", "False"]`. The language model receives instruction as follows:

- **numerical_value_0_1**: Rate the given question "QUESTION" with a value between 0 and 1 according to the given criteria. Explain and reflect upon your choice.

Answer in the format:

explanation: EXPLANATION

value: VALUE

Where EXPLANATION is a detailed explanation of why VALUE was chosen.

- **category_tf**: Assign the given question "QUESTION" a specific category according to the given criteria. Explain and reflect upon your choice.

Categories:

- true
- false

Answer in the format:

explanation: EXPLANATION

category: CATEGORY

Where EXPLANATION is a detailed explanation of why CATEGORY was chosen.

The goal is to test the evaluation of ZLW question-wording guidelines [Unia]. Guideline translations:

- **simple_language**: The final decision-making basis for the formulation is the following subjects! Unambiguity and simplicity must be balanced!
 - Formulate briefly, understandably and sufficiently precisely.
 - Not bureaucratic, technocratic or scientific.
 - Avoid foreign words
 - Address the target group
 - Use simple language, without slang, dialect or subculture language
 - Make the item precise in terms of the question's aim and intention.
- **simple_and_positive_questions**: - Negations are linguistically negatively formulated questions. Double negatives (especially in translations) are to be avoided in any case. s) should be avoided at all costs.
 - Avoid negative polarity: Item polarity (scale direction) If possible, do not change. If negative and positive and positive items are present, a separate evaluation is recommended. separate evaluation.
 - Unambiguity: Only assign factual content or thought to each item! or thought! One-dimensional means, that an agreement or disagreement allows only one interpretation. Ambiguity is to be avoided, especially with especially in answer categories, not only in the formulation of questions. question wording. it can also be said that no lo- links should be present in a question (and, or...). (and, or...).
- **insinuation_free**: Insinuations and suggestive questions should be avoided. Insinuations do not only refer to the subject but can also address third parties. A question may be rejected because the respondent does not agree with the insinuation. Suggestive questions restrict the freedom of the answer and restrict the respondent's freedom of response. Do not ask suggestive questions in a way that influences answering behavior.
- **clear_temporal_reference**: Questions have a clear temporal reference. Dates and time spans are precisely stated.

- **concise_categories:** Use answer categories that are concise. Closed questions should have disjunctive (non-overlapping) answer categories. Categories that are precise and cover the solution space.
- **explanation_of_the_unknown:** Clarify unclear terms and necessary knowledge for understanding the question.
- **non_judgemental:** Avoid terms that are associated with strong opinions or emotions. Strongly value-laden terms (peace, war, crime, justice) should be avoided. They provoke more extreme response behavior.
- **comparison_with_valid_scale:** Compared concepts are of the same scale. You can't equate "higher" with "colder," or "degree of rejection" with "brightness". "Do you prefer the Grand Coalition or a Green policy?" cannot be compared unambiguously. The respondent must interpret for himself how he or she matches a grand coalition and a potentially green policy. One is a party constellation for forming a government, the other a political program.

The language model is then queried to evaluate a question according to categories using a result type. Example multiple-choice (Table 6) question:

Which of the following statements about binary trees is true?

- Binary trees can include Cliques
- A binary tree node can reference itself
- Binary trees are traversable

The evaluation prototype features a one-shot example for the category **simple_language**. A question evaluation pair given as two messages incentivizes the language model to adapt its output to the one-shot example. The language model reads the two messages as its own history. Construction of messages:

1. System message: The assistant's role
2. User message: One-shot example: question to evaluate with instructions on evaluation
3. Assistant message: One-shot example: Expected system response
4. User message: Question to evaluate with instructions on evaluation

Zero-shot consists of only two messages, a system and a user message. Table 6.3 depicts gpt-3.5-turbos [Webs] zero-shot output:

gpt-3.5-turbo [Webs] outputs the following:

| guideline | explanation | category |
|-----------|-------------|----------|
|-----------|-------------|----------|

| | | |
|---|---|------|
| simple_language | The question consists of a clear, brief and precise description of the statements about binary trees and asks for the true statement among them. It avoids all kinds of bureaucratic, technocratic or scientific language, and uses simple language without slang. It also targets the intended audience with basic knowledge in regards to binary trees and algorithms | true |
| simple_and
_posi-
_tive_questions | The question presents three statements about binary trees and asks which one is true. The statements are clear and precise, avoiding negative polarity and any ambiguity. The question also addresses the target audience of individuals knowledgeable about binary trees and uses an appropriate language level. Overall, the question fulfills the criteria and can be considered true. | true |
| insinuation_free | The question is clear, specific, and objective. It does not contain any insinuations or suggestive language, and it provides three answer options to choose from. | true |
| clear_temporal_reference | The question adheres to the given criteria. It presents three statements about binary trees and asks the respondent to identify the true one. | true |
| concise_categories | The answer categories are precise and disjunctive. There are no overlapping categories and the possible solutions to the question are included in the answer categories. | true |
| explanation_of
_the_unknown | The question is short, clear and easy to understand. It provides three statements about binary trees without using technical language or including unknown terms. | true |
| non_judgmental | The question is objective and fact-based. It does not include any vocabulary that is associated with strong emotions or opinions. It consists of a list of options that can be evaluated without being influenced by any personal or emotional attitudes. | true |
| comparison_with
_valid_scale | The question is concise and only contains a single clear objective criteria to evaluate the answers on. It is clear what is meant by binary trees, and the statements being compared to each other are on the same scale. | true |

Table 6.2: Evaluation prototype test: One-shot example.

| guideline | explanation | category |
|------------------|--------------------|-----------------|
|------------------|--------------------|-----------------|

| | | |
|-----------------------------------|--|-------|
| test simple_language | The question asks for the true statement about binary trees among the options given. Option 1 ("Binary trees can include Cliques") is false because cliques are complete subgraphs where all nodes are connected to each other, and binary trees are characterized by at most 2 children per node, making it impossible to have cliques in them. Option 2 ("A binary tree node can reference itself") is also false because a binary tree is a tree data structure and by definition, there cannot be a cycle in a tree. Option 3 ("Binary trees are traversable") is true because we can traverse a binary tree in different ways such as Inorder, Preorder, and Postorder traversals. Therefore, the true statement about binary trees is that they are traversable. | true |
| simple_and
_positive_questions | The question provides three statements about binary trees and asks which one of them is true. The first statement claims that binary trees can include Cliques, which is false because Cliques are not part of a binary tree structure. The second statement says that a binary tree node can reference itself, which is also false because a binary tree is a directed acyclic graph (DAG), so a node cannot reference itself directly. The third statement claims that binary trees are traversable, which is true because a binary tree can be traversed in different orders (pre-order, in-order, post-order). Therefore, only one statement is true, making this question unambiguous and one-dimensional. | true |
| insinuation_free | The question provides three statement options about binary trees and asks which one is true. It does not contain any insinuations or suggestive language. The question is clear and straightforward and does not restrict the respondent's freedom to answer. As such, the question is factual and can be evaluated based on its accuracy. | true |
| clear_temporal_reference | The question asks for a factual statement about binary trees and gives three options to choose from. None of the options provided mention any specific dates or time spans, so the temporal reference is not relevant to this evaluation. | false |

| | | |
|-----------------------------|--|------|
| concise_categories | The question provides three statements and asks which one is true about Binary trees. - "Binary trees can include Cliques" is false as Cliques are not allowed in Binary trees. - "A binary tree node can reference itself" is also false as self-referencing nodes violate the definition of a Binary tree. - "Binary trees are traversable" is true as Binary trees can be traversed in different ways, such as Inorder, Preorder, and Postorder traversals. | true |
| explanation_of_the_unknown | The question is testing the knowledge about binary trees, a data structure in computer science. A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. Cliques is not a term related to binary trees, and hence option A is false. Self-referencing is not possible in a binary tree, and hence option B is also false. Binary trees are traversable, meaning that one can visit all the elements in the tree, and hence option C is true. | true |
| non_judgmental | Based on the given statements, it is possible to evaluate whether or not they are true or false without relying on any strong opinions or emotions. The statements are focused on factual information about binary trees, which can be objectively verified. | true |
| comparison_with_valid_scale | The question asks which of the statements about binary trees is true and provides three options for the answer. The statements are all related to binary trees, which are a well-defined and consistent concept in computer science. Therefore, there is no ambiguity in the concepts being compared and they are all on the same scale. | true |

Table 6.3: Evaluation prototype test: Zero-shot example.

gpt-3.5-turbo is able to describe why specific guidelines are followed and assign the according category. For example, the question does not "contain any insinuations or suggestive language", depicted in Table 6.2. gpt-3.5-turbo assigns "true", stating that the question satisfies the guideline **insinuation_free**.

Zero-shot yields a different result than one-shot for guideline **clear_temporal_reference**. The explanation zero-shot is sound, stating that no "specific dates or time spans [are provided]" and concludes that "temporal reference[s are] not relevant for [the] evaluation.", according to Table 6.3.

One-shot argues similarly. An ambiguity in the guideline explanation is a possible reason. The guideline explanation has to state that not every question must have a temporal reference. Temporal references must be precise, only if the question asks about a temporal fact.

The decision to include an evaluation module in EvalQuiz is made based on the obtained results. gpt-3.5-turbo enables evaluation. EvalQuiz uses GPT-4 for evaluation. GPT-4 improves upon gpt-3.5-turbo in many tasks [Ope23].

The question example focuses on true positive testing. Other question examples with obscure formulation or dialect falsified guidelines as **simple_language**. Some guidelines highlight aspects missing in the question text alone. **simple_language** asks about the intended audience. There is no information about the intended audience in the question.

Guideline Evaluation with EvalQuiz

EvalQuiz implements a general `LanguageModelEvaluation`. The user can describe guidelines in the `LanguageModelEvaluation.evaluation_description` field. `LanguageModelEvaluation` supports few-shot examples in form of a list of `GenerationEvaluationResults`.

6.6.4 Question Specification Changes

We changed the question specification by `_metrics` mode from using an arithmetic expression to the `evaluator_type` field, which internally specifies an arithmetic expression. This prevents remote code execution, which is a security risk. The `evaluator_type` field references how the `by_metrics` mode compares `evaluation_reference` with the specified value: `evaluation_result`. Appendix Section B describes `evaluator_types` more in detail. We renamed the `result` to `generation_result` and `evaluation` to `evaluation_result` for clarity and consistency reasons. The `evaluation_result_type: value_range` produces a `float_value` `evaluation_result` and `evaluation_result_type: categorical` produces a `str_value` `evaluation_result`. A question contains a dictionary of `evaluation_result` to support multiple evaluations. Those are mapped by `evaluation_references`, with the reference field removed. We renamed `evaluation_type` to `evaluation`, the super-class of `evaluation` implementations, as the `language_model_evaluation`. We replaced `evaluation` getter and setter with the `few_shot_examples` field, which is a list of `generation_evaluation_results`.

6.7 Frontend

The EvalQuiz frontend uses React and MaterialUI [Webba] [Webz]. Users edit question specifications through a JSONForms component [Webx]. JSONForms takes a JSONSchema, UISchema and data and builds a form component. The form features JSONSchema data-binding, input validation, and rule-based visibility [Webx]. We designed the front-end to evaluate EvalQuiz in cooperation with lecturers from different fields. Our frontend offers a reduced set of configuration options by default, as Figure 6.14 depicts. Interaction becomes simpler. Users can toggle the advanced mode to edit the complete question specification.

The screenshot shows a configuration interface for a question specification subset. At the top, there are three tabs: 'CAPABILITIES', 'LECTURE MATERIALS', and 'QUESTION TO GENERATE'. The 'CAPABILITIES' tab is active. Below the tabs, there is a 'Batches' section with a '1' icon and a trash icon. The main area is titled 'Capabilities' and contains a configuration card for '1 KNOW_AND_UNDERSTAND'. This card has several fields: 'Educational Objective' (KNOW_AND_UNDERSTAND), 'Keywords' (with a plus icon), 'Microservice' (with a trash icon), 'Architecture' (with a trash icon), 'Issue Propagation' (with a trash icon), and 'Relationship' (COMPLEX). At the bottom of the card is a '2 APPLY' button. Below the configuration card, there is a blue button labeled 'ITERATE CONFIG' with a right-pointing arrow.

Figure 6.14: EvalQuiz frontend: Configuration question specification subset.

6.7.1 gRPC Support

gRPC Web faces limitations regarding client and bidirectional streaming [Webu]. gRPC Web requires the setup of the Envoy proxy to connect to gRPC servers [Web1]. Uploading to a material server requires client-side streaming. A simple solution is: "Client flask", an intermediate server translating HTTP requests to gRPC. The client uses Axios to interact with material and pipeline server through "client flask" [Webd]. "Client flask" converts HTTP file uploads to a gRPC stream. "Client flask" connects via channels to the gRPC server endpoints.

Server-client connections through "client flask" cannot utilize gRPC features. "Client flask" converts streams to single requests or responses. gRPC on Node.js improves upon gRPC Web by offering client and bidirectional streaming without a proxy [Webt] [Webu]. Adapting the frontend to gRPC on Node.js can provide streamed feedback on pipeline iteration and streamed material uploads, directly from the client. We were not aware of gRPC on Node.js at the time of implementation.

6.7.2 Custom JSONForms Components

Modifying the JSONForms UISchema cuts down the original generation configuration interface to its core functionality: Config iteration through batch creation. Two custom JSONForm components add to the user experience. Users can select lecture materials in a searchable option list. Our custom component queries the material server for available materials. Materials become select-able drop-down options distinguishable by name and hash (Figure 6.15). The second custom component hides generation results if empty and provides a simple selector for question types. The component presents generation results as code listings, if present. JSONForms registers the custom components and displays them according to the UISchema [Webx].

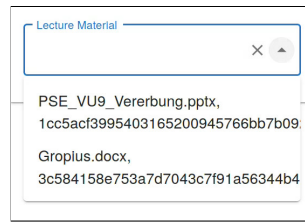


Figure 6.15: EvalQuiz frontend: Material suggestion component.

6.7.3 Protobuf to JSONSchema Conversion

gRPC uses Protobuf for its type definitions [Webax]. EvalQuiz’s Git repositories share those type-definitions in a submodule. Section 6.4 provides a deeper insight into how we use Protobuf. Converting the Protobuf to JSONSchema prevents a manual rewrite of the question definition as JSONSchema. The `protoc-gen-jsonschema` compiler is able to convert Protobuf to JSONSchema [Webar]. The Protobuf schema needs to be converted with the `enforce_oneof` and `enums_as_strings_only` option. `enforce_oneof` ensures that the converter interprets `oneOf` clauses. `enums_as_strings_only` declare that enums only can be input as strings, not numbers. This option is necessary to simplify the generated form by JSONForms. JSONForms renders a drop-down selector instead of a text input with validation for numbers and strings. Available options are visible to the user.

6.8 Deployment and Networking

EvalQuiz’s deployment consists of 7 containers specified in a Docker compose configuration (YAML). The configuration follows a reverse-proxy architecture style exposing only the reverse proxy to the outside. An NGiNX instance implements the reverse proxy [Webag] and redirects incoming requests to the `client react` frontend and `client flask` intermediate server:

```
1 ...
2 location / {
3     proxy_set_header Host $host;
4     proxy_pass http://client-react:3000/;
5 }
6
7 location /server {
8     proxy_set_header Host $host;
9     proxy_pass http://client-flask:8000/;
10 }
11 ...
```

NGiNX running in its own Docker container utilizes Dockers builtin DNS name resolution [Webj]. Docker resolves `client-react` and `client-flask` to their respective IPs in Dockers bridge network. Figure 6.16 visualizes the Compose configurations containers with their `container_names`. All containers in a Docker compose configuration share a default bridge network [Webj] [Gra].

6.8.1 Multistage Builds

”Client flask” is an NGiNX instance hosting static files compiled by React App [Webbb]. Docker compiles the static files with a multistage build [Webj] [Gra]. Stage 1 is the build stage, where build dependencies are set up according to the Dockerfile. Docker copies source code to the build container and sets up environment variables for the build. `npm run build` compiles the static files. Stage 2 represents the runtime environment. Docker creates a separate container on a lightweight image as `nginx:1.17.0-alpine`. Docker copies the compiled files from the build stage and the NGiNX config from the host. NGiNX serves static files according to its config providing the frontend. Dockerfile describing the multistage build:

```

1  ...
2  # Stage 1: Image, environment variable and file setup
3  ...
4
5  RUN npm run build
6
7  # Stage 2
8  FROM nginx:1.17.0-alpine
9
10 COPY ./nginx/nginx.conf /etc/nginx/conf.d/default.conf
11
12 COPY --from=build-stage /evalquiz-client-react/build /usr/share/nginx/html
13
14 CMD nginx -g 'daemon off;'
```

The advantages of multistage builds are smaller runtime images with low memory footprints. NGiNX is capable of serving static files on a production scale [Webag]. The reason why we use two NGiNX instances is to separate concerns. Running two instances is a small overhead as NGiNX is lightweight [Webag]. The reverse proxy instance of NGiNX is responsible for exposing the frontend and ”client flask” to the outside. The inner NGiNX instance is responsible for hosting the front-end’s static files. Code changes imply container rebuilds. For example: A change in the reverse proxy configuration causes a rebuild of the reverse proxy image. Docker does not need to rebuild the frontend, if separated.

6.8.2 Cloud Computing Platform: Microsoft Azure

Microsoft Azure hosts EvalQuiz [Webaa]. The Web App connects to a container registry with EvalQuiz’s Docker images. The Web App reads EvalQuiz’s Docker compose configuration (Figure 6.16) and sets up the containers accordingly. Azure adds SSL encryption and name resolution to a subdomain of `azurewebsites.net`. Azure provides OpenAI API access through Azure OpenAI [Webaa]. Our pipeline server uses GPT-4 with Azure OpenAI.

6.8.3 Continuous Integration (CI)

The pipeline and material server are tested with a similar GitHub workflow: A push to the master branch triggers the workflow [Webp]. The workflow uses a recursive checkout, to resolve the server’s code and submodules. The `devcontainers/ci` action then reads the repositories Development

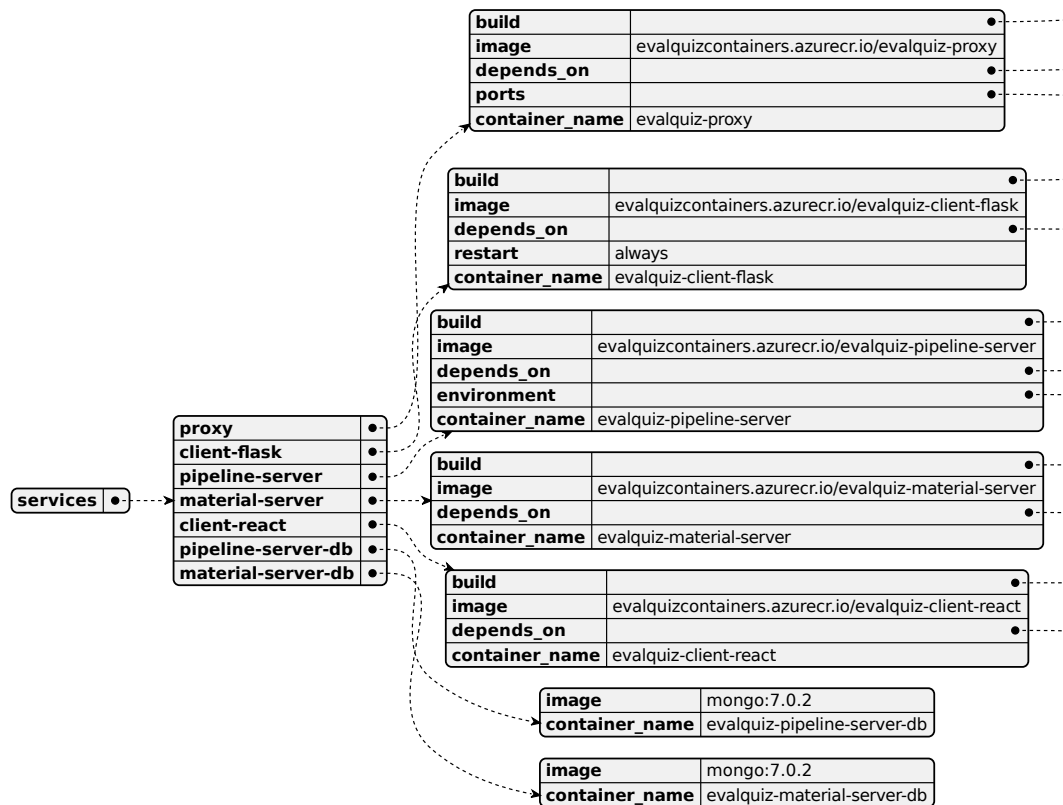


Figure 6.16: EvalQuiz full stack deployment: Docker compose configuration (YAML) visualized with PlantUML [Webao].

Container configuration and starts a Development Container instance [Webi]. CI uses the same environment as development, due to the reasons that Section 6.3 discusses. The workflow conducts four checks:

```

1 # Linting with pyflakes
2 pyflakes .
3 # Type checking with mypy
4 mypy .
5 # Code format checking with black
6 black . --check
7 Testing with pytest
8 pytest .

```

The goal is to improve code quality and maintainability. Neglected typing and failing tests indicate that the code increment should be improved. CI has positive effects on software quality and development productivity [SBZ17].

6.8.4 Continuous Deployment (CD)

All containers in the Compose configuration (Figure 6.16) are continuously deployed through a deploy branch. The CD workflow checks out the repository recursively, logs into the Azure container registry and builds the image according to `dockerfile_production`. The built image is pushed to the container registry. The Azure Web App hosting EvalQuiz has CD enabled in the deployment settings. The push of a new version to the container registry causes the EvalQuiz Web App to update as well. The latest deploy branch commits reflect the deployed system. Updating EvalQuiz is as easy as pushing to GitHub [Webp].

The material and pipeline server feature a second CD workflow respectively. Sphinx documents the material and pipeline servers artifacts [Webbf]. Sphinx reads doc-strings in the Google format and generates static files [Webbf]. Sphinx documents using a Read The Docs (RTD) theme. Pushes to the master branch trigger the documentation workflow. An open source GitHub action (sphinx-notes) converts Sphinx to GitHub Pages V3 [Webbg]. The missing files submodule initially caused documentation generation to fail. sphinx-notes does not support recursive checkouts by default. We have forked sphinx-notes to support recursive checkouts [Webn]. This enables Sphinx to document the material and pipeline servers with their Protobuf definitions and templates. The documentation therefore includes definitions generated from the Protobuf schema as well.

7 Evaluation

We contacted 14 lecturers to evaluate EvalQuiz in a survey format. This chapter gives insight into the study's design, results and threats to validity. An important part of the evaluation is conveying EvalQuiz's conceptually and practically. We want to assess if an iterative self-assessment system can benefit lecturers in creating self-assessment. Similarly, if EvalQuiz provides qualitative results and is intuitive to use.

7.1 Study Design

We evaluate EvalQuiz as deployed in Section 6.8 with a questionnaire to assess EvalQuiz's real-world performance. The survey targets lecturers of different domains and is designed to be accessible. EvalQuiz itself is built for university usage of multiple domains. The EvalQuiz frontend features a tour guide, introducing the participants to the interface. The frontend gives feedback about the system's internal status at any step of the process. A notification component in the frontend implements system feedback. The survey targets a total time of approximately 30 minutes. We collected questionnaire answers with Microsoft Forms [Webac].

We provide a survey manual to inform about the survey's procedure and EvalQuiz's concepts. This document features a small introduction to explain EvalQuiz and the survey procedure, in each step of the process. Participants are encouraged to read EvalQuiz's concepts and walk through the tour guide before trying out the interface on their own. We explain the following concepts that the interface references:

- **Batch** Describes an isolated set of settings for question generation. Capabilities and lecture materials in each batch can influence all generated questions of the respective batch. See question specification in appendix chapter B.
- **Capability** as explained in section 5.3.2. See question specification in appendix chapter B.
- **Lecture Material** represented through a hash of its contents. Can be chosen as one of the uploaded lecture materials. See question specification in appendix chapter B.
- **Question To Generate:** Choice which questions the system generates in a batch. See question specification in appendix chapter B.

Usage and configuration examples highlight EvalQuiz's possible usages to the participant. The examples represent practical applications of EvalQuiz concepts in a problem-goal format. The examples solutions are question specifications:

7.1.1 Usage Example: Summarize As Self-Assessment

- **Problem:** A lecture can consist of multiple thousands of slides. Sometimes it can be hard to find relevant information about a SPECIFIC_KEYWORD.
- **Goal:** Use a large corpus of lecture materials to filter out and generate questions for a SPECIFIC_KEYWORD.

Implementation:

1. Upload all lecture slides or the book that the lecture is based on to the system.
2. Configure question specification:

```
1      {
2      "material_server_urls": [],
3      "batches": [
4      {
5      "capabilities": [
6      {
7      "educational_objective": "KNOW_AND_UNDERSTAND",
8      "keywords": [
9      "SPECIFIC_KEYWORD"
10     ],
11     "relationship": "COMPLEX"
12     }
13     ],
14     "lecture_materials": [
15     {
16     "hash": "6fff8c54113aa085dbc0fc8abff06bd759ee7e66401f464a8233d8a47e8b9bf0",
17     "hash": "0fc8abff06bd759ee7e66401f464a8233d8a47e8b9bf06fff8c54113aa085dbc",
18     // ...
19     // Hashes of all lecture slides uploaded to the system.
20     }
21     ],
22     "question_to_generate": [
23     {
24     "question_type": "MULTIPLE_CHOICE"
25     },
26     {
27     "question_type": "MULTIPLE_CHOICE"
28     },
29     // Multiple questions allow to generate about multiple aspects of the topic.
30     ]
31     }
32     ],
33     }
```

7.1.2 Usage Example: Find Connections Between A And B

- **Problem:** A section highlights two topics KEYWORD_A and KEYWORD_B. Sometimes it can be hard to grasp connections between topics.
- **Goal:** Use the lecture material containing the section to find out how KEYWORD_A relates to KEYWORD_B.

Implementation:

1. Upload all lecture slides or the book that the lecture is based on to the system.
2. Configure question specification:

```

1     {
2       "material_server_urls": [],
3       "batches": [
4         {
5           "capabilities": [
6             {
7               "educational_objective": "KNOW_AND_UNDERSTAND",
8               "keywords": [
9                 "KEYWORD_A",
10                "KEYWORD_B"
11              ],
12              "relationship": "COMPLEX"
13            }
14          ],
15          "lecture_materials": [
16            {
17              "hash": "6fff8c54113aa085dbc0fc8abff06bd759ee7e66401f464a8233d8a47e8b9bf0",
18              // ...
19              // Hash of lecture material containing the section.
20            }
21          ],
22          "question_to_generate": [
23            {
24              "question_type": "MULTIPLE_CHOICE"
25            },
26            {
27              "question_type": "MULTIPLE_CHOICE"
28            },
29            // Multiple questions allow to generate about multiple aspects of the topic connections.
30          ]
31        }
32      ],
33    }

```

7.1.3 Usage Example: Partial Regeneration

- **Problem:** We have generated a variety of questions, but are not satisfied with the outcome of some questions.
- **Goal:** Partially regenerate questions.

Implementation:

1. Generation result:

```

1     {
2       // ...
3       "question_to_generate": [
4         {
5           "question_type": "MULTIPLE_CHOICE",

```

```

6         "generation_result": {
7             "multiple_choice": {
8                 // Result that we are satisfied with.
9             }
10        },
11    },
12    {
13        "question_type": "MULTIPLE_CHOICE",
14        "generation_result": {
15            "multiple_choice": {
16                // Result that we are not satisfied with.
17            }
18        }
19    }
20 ],
21 // ...
22 }

```

2. Adapt to discard generation result:

```

1    {
2        // ...
3        "question_to_generate": [
4            {
5                "question_type": "MULTIPLE_CHOICE",
6                "generation_result": {
7                    "multiple_choice": {
8                        // Result that we are satisfied with.
9                    }
10            }
11        },
12        {
13            "question_type": "MULTIPLE_CHOICE",
14        }
15    ],
16    // ...
17 }

```

7.1.4 Questionnaire Structure

The questionnaire collects information in three sections: **1. Academic Information**, **2. Self-Assessment in General** and **3. EvalQuiz Self-Assessment Generation**. The questionnaire question types are: multiple-response with an optional addable Other field, multiple-choice and free text. Multiple-choice and multiple-response answers are mandatory while free-text answers are optional. Free-text answers only need to be answered: "if applicable". The multiple choice questions consistently offer 5 answer options: **Strongly Agree**, **Agree**, **Neutral**, **Disagree** and **Strongly Disagree**. The answer options are universal and symmetric. Our questionnaire references both the web address of the frontend and the survey manual. Participants are advised to answer **EvalQuiz Self-Assessment Generation** after trying out the EvalQuiz frontend.

1. **Academic Information** asks about the participant's academic field, background and experience in creating lecture materials.

2. **Self-Assessment in General** collects which self-assessment forms the participant is familiar with and self-assessment constraints.
3. **EvalQuiz Self-Assessment Generation** is the longest part of the questionnaire highlighting the experience with EvalQuiz itself.

7.2 Results

We collected 6 results. The average questionnaire completion time is approximately half an hour. All participants are familiar with lecture material creation to a high to very high degree. Our participants worked on multiple-choice, mock exams and blue slides most prominently, visualized in Figure 7.1. We describe blue slides as sections for the student to pause and ponder in our questionnaire. All 6 participants agree that time constraints prevent self-assessment creation, depicted in Figure 7.2. One-third of the participants state that creativity prevents self-assessment creation. The verifiability of questions and current platforms and tools is unproblematic, with one answer respectively. The majority of participants agree that self-assessment is currently lacking in university courses.

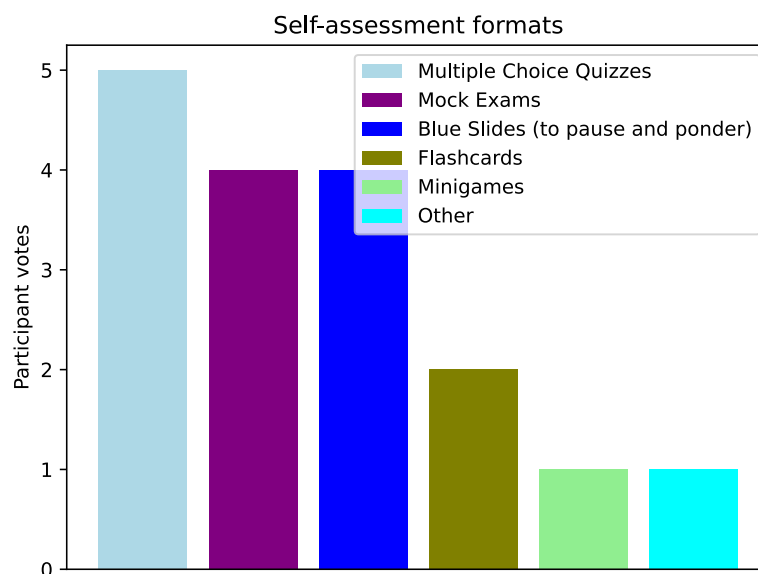


Figure 7.1: Survey results: Self-assessment that the participants have experience working with.

All participants uploaded lecture slides to our system. A single participant tried out EvalQuiz with papers. The intuitiveness of interaction is judged as neutral on average. The subjects agree that EvalQuiz generates questions timely. The majority of volunteers see question generation as reliable with one outlier stating the contrary. The subjects are satisfied with the generated questions' correctness. One participant remarks: *"Slight grammar issues, but no content issues."* 2 participants state that the generation outcome is better in English than in German, while the remaining 4 participants only tested English lecture materials. The participants evaluate the versatility and originality of the generated questions as neutral on average.

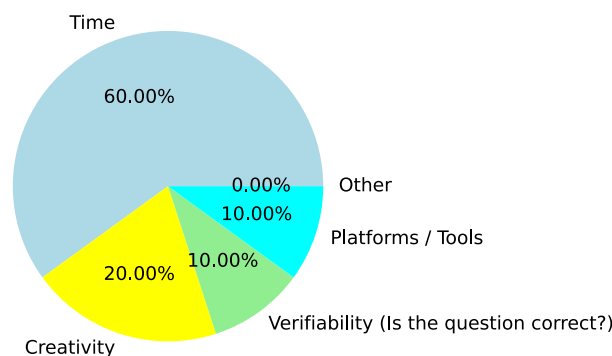


Figure 7.2: Survey results: Self-assessment constraints.

Clearing and regenerating questions are seen as a helpful feature when managing a collection of questions with EvalQuiz. The volunteers strongly agree that questions generated by EvalQuiz reference knowledge from the lecture materials they were generated from. The model takes its instructions to reference lecture materials too literally sometimes, as one participant criticizes: *"i had one question that said: "[...] according to the text?". However as this is a stand alone question, it's unclear what text references to."* The volunteers strongly agree that the system saves time in self-assessment generation. The results, visualized in Figure 7.3, show that lecture materials, batches, keywords in Capabilities and question type in question to generate are most beneficial for question generation.

We collected feedback through free text questions. We asked our participants what they would like to see implemented in the system. Our subjects requested more features to explain the interface elements, such as tooltips. Additionally, multiple participants highlighted that they would like to see more question types. A participant states that a language setting would help the user to gain control over the language output. The participant critiques the lack of control over the output language with German lecture materials. Another participant suggests a pause mechanism for the tour guide, in order to interrupt and continue the tour. The systems should give more feedback on which lecture material the user selected. One participant requests more flexibility in deciding which parts of a question the system should regenerate. A single field of a question could be regenerated instead of the whole question. The participant describes three generation options for multiple-choice: (1) generate the question text only, (2) generate the question text + answer text and (3) generate the complete question: question text + answer text + distractor texts. The same participant suggests version control features to compare different generation alternatives. The interaction can benefit from specifying multiple "question to generate" at once, as the participant comments. Another participant suggests an expert mode in which the user can modify the generation prompt (message composition). The same participant positively comments on Evalquiz: *"Great idea and a nice initial MVP"*.

We collected general remarks about Evalquiz. One subject recommends adding PDF to the supported file types. Another volunteer reports difficulties with processing \LaTeX slides and suggests that manual conversion to markdown eliminates the problems. The participant critiques the lack of

information on how Bloom’s taxonomy relates to question generation results. Another subject remarks that different educational objectives withhold the same generation result and critiques that distractors are sometimes obvious.

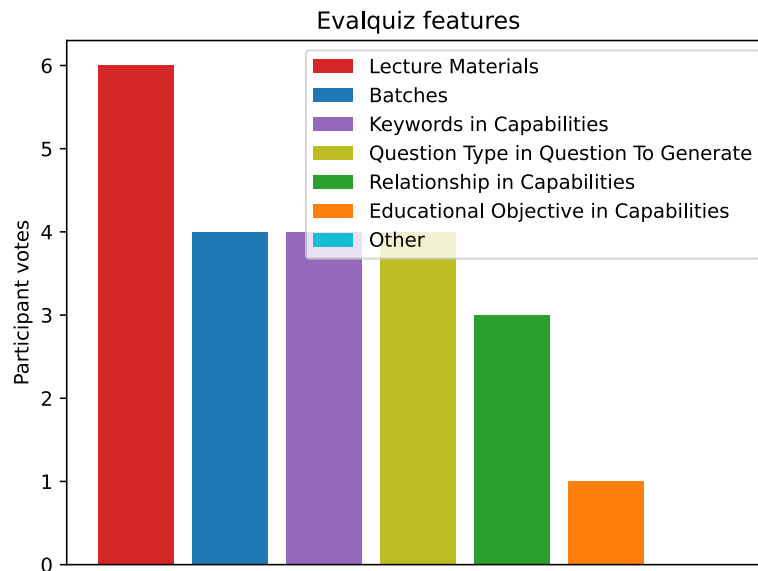


Figure 7.3: Survey results: Beneficial features for question generation.

7.3 Discussion

The evaluation shows that our system improves on the main bottleneck of self-assessment creation: time. Feedback about the generation quality is positive, while drawbacks exist in steerability and intuitiveness. Lecturers are fond of the iterative approach but wish for more control. For example in partial regeneration and the composition of prompts. We assume that the mixed feedback about model steering is due to explanatory boundaries and prompts (message composition). Multiple concepts influence generation and need to be explained. Prompt design influences the model’s behavior: the model should be incentivized to prioritize educational objectives. Lecture materials influence the generation strongly, as reflected by Figure 7.3. We argue that implementing more question types improves steering through educational objectives. The educational objective: apply is currently limited by multiple-choice/multiple-response questions. We expect free text questions to better fit the educational objective: apply. The participants explicitly requested more question types.

The evaluation results provide valuable feedback regardless of the sample size. Individual comments recommend possible improvements. One participant states that they *”would actually use [the system] to get some questions”*. Evaluation is a success for EvalQuiz, especially because EvalQuiz is a novel approach. However, we expected our participants to use lecture material different from lecture slides. Books, wikis, exercises and other lecture materials remain unexplored. We

partially explain our small sample size with the targeted audience. Lecturers are mostly PhD students, postdocs and professors. The participants face significant time limits, which align with our assumptions in Figure 4.3. The survey results underline that time limitations exist in self-assessment creation, depicted in Figure 7.2.

7.4 Threats to Validity

The small sample size of 6 is a threat to validity. All respondents work in the field of computer science, which limits generalizability to other domains of higher education. We expect computer scientists to be more familiar with datatypes such as JSON or YAML than other users. This can be a usability limitation. Both formats are not self-explanatory and depend on commonly used data structure syntax as lists: $[A, B, C]$ and dictionaries $\{A : B\}$. Our subjects mainly uploaded lecture slides to EvalQuiz, which limits the system's generalizability to other lecture materials.

8 Conclusion and Future Work

This chapter summarizes the thesis' key aspects and insights. We discuss how future work can benefit from EvalQuiz's findings. Limitations open research opportunities to further investigate aspects of the system and its theory.

8.1 Summary

The thesis confirms that large language models are a suitable technology for self-assessment creation, underlining their generalization to different problem areas. Large language models are capable of reliably outputting data according to a specification, answering **RQ5**. Our findings add to the existing use cases of language models. The thesis conceptualizes, discusses and implements novel ideas for self-assessment creation. Our survey delivers positive results limited by sample size. The survey shows within its limitations that the generated questions are correct, indicating the true side of **RQ1**. This includes computer science and software engineering **RQ2**. **RQ2** is difficult to answer for languages, as GPT-4 faces its own language constraints discussed in Section 4.4. EvalQuiz can be extended to support different language settings. This requires query translation of every question type. A separate recommendation system is required to answer **RQ6**, interesting for future work. Open sourcing a large language model faces security concerns as discussed in Section 2.2. This limits the currently available options to compete with GPT-4. We expect proprietary models such as GPT-4 to perform best **RQ4**. EvalQuiz has all abstractions in place to support open source models to further investigate **RQ4**, when comparable open source options are available.

EvalQuiz runs into limitations regarding steering and interaction, discussed in Chapter 7. The conducted expert interviews reveal the unused potential that lays within individualized teaching. Tighter integration and re-usability is a reoccurring theme in the thesis, speaking of lecture material sharing, standardizing self-assessment data types and interconnectivity between LMSs with services. An LMS could query the pipeline server directly through the specified interface.

8.2 Benefits

Both students and lecturers benefit from our results, carving the path towards more accessible self-assessment. Lower self-assessment creation boundaries enable lecturers to offer students self-assessment while not losing focus on their duties. EvalQuiz not only benefits creation but also evaluation of self-assessment. Self-assessment evaluation enables lecturers to get a better estimate of the self-assessment's suitability for their course. Future work can benefit from considerations about question types, expert interviews, requirements collection, concepts and the implementation. We encourage the reuse of EvalQuiz's components and ideas. The message composer scheme can be applied to other domains, to generate complex data types consistently.

8.3 Limitations

Our survey evaluation is limited by its sample size. Questionnaire responses are from lecturers of the computer science domain, limiting their generalizability to the population of university lecturers. Our approach is based on Bloom's taxonomy [BK20] and is therefore dependent on the taxonomy's descriptiveness of educational objectives. We use GPT-4 with OpenAI API, which is a limitation due to its proprietary. The material filter does not support PDF lecture materials. We argue that PDF conversion is currently unsatisfactory for our approach, which can be a topic for future work.

8.4 Lessons Learned

We faced problems during implementation, as with most software projects. A bug in production initially limited the document format support. The implementation uses the Linux file system for file storage, which leads to permission and state issues. Our file data flow is complex. Files are uploaded to the material server. The pipeline server then queries files from the material server. The state is additionally held in intermediate caches. This leads to issues with testing and deployment. The file system state was carried over in test runs. Tests were passing due to persistent state on the disk. Development containers could not prevent those difficulties, as the local file system is mounted by each development container. The built container uses state as versioned in the repository. These issues can be prevented by managing all lecture material state with a database. We decided against this, due to concerns with large file handling compatibility and performance of databases. We agree in hindsight that storing files in a database would have prevented some of the encountered problems.

The intermediate gRPC translation layer should be replaced for a gRPC native client implementation, as section 6.7.1 discusses.

8.5 Future Work

We provide a visual material implementation guide in Section 6.6.1. Future work is able to assess visual material integration in combination with GPT-4 or other multi-modal models **RQ3 (Optional)**.

Future work can analyze and improve upon the thesis concepts. The message composer scheme can be valuable for a variety of future applications. For example, applications where language models interact with APIs. Language models can act as compatibility layers between different APIs and the human. This is especially interesting for home automation, plagued by many different standards and incompatible interfaces. There are possibilities to investigate language models interacting with API-exposing services.

Complete automation of self-assessment generation is worth investigating. Future work is able to build a system that does not require any lecturer interaction. Formulating and teaching quality standards to language models can be further researched. We would like to see more research

on conditional regeneration using metric evaluation, as implemented by the evaluation module. The frontend currently only supports a subset of the question specifications and can be extended accordingly.

Future work can extend and further investigate almost all of the pipeline server abstractions. Our initial approach for material filtering can be improved, other language models tested, different message wordings and question types implemented and more. Different few-shot example configurations are worth exploring. The implemented message composer composes messages only English messages so far. Providing non-English lecture materials and messages, as indicated in Section 8.1, remains unexplored. Technologies such as IPFS can replace the material server. Generated self-assessment can be shared through content addressable storage as well. Closer cooperation in digital education between institutes and universities has unexplored potential. Future work can address options to share software (LMS, plugins), infrastructure and content to a higher degree than state-of-the-art. The collected feedback in Section 7.2 can be incorporated with EvalQuiz.

Future work can pursue LMS plugin integration of EvalQuiz. Integration with LMS plugins connects self-assessment generation with lecture material management and distribution. This prevents intermediate imports and exports of generated self-assessment. LMSs could encourage lecturers to increase the "self-assessment coverage" of shared course lecture materials. Automatic evaluation of Capabilities can give insight into the course's structure. Lecturers can benefit from more guidance in course creation.

Bibliography

- [AM19] C. B. Asmussen, C. Møller. “Smart literature review: a practical topic modelling approach to exploratory literature review”. In: *Journal of Big Data* 6.1 (2019), pp. 1–18 (cit. on p. 37).
- [And19] H. L. Andrade. “A critical review of research on student self-assessment”. In: *Frontiers in Education*. Vol. 4. Frontiers Media SA. 2019, p. 87 (cit. on p. 1).
- [ARS+16] J. Araki, D. Rajagopal, S. Sankaranarayanan, S. Holm, Y. Yamakawa, T. Mitamura. “Generating questions and multiple-choice answers using semantic analysis of texts”. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. 2016, pp. 1125–1136 (cit. on pp. 1, 19).
- [BB95] D. Boud, A. Brew. “Developing a typology for learner self-assessment practices”. In: *Research and development in Higher Education* 18.1 (1995), pp. 130–135 (cit. on pp. 1, 5, 17).
- [BK20] B. S. Bloom, D. R. Krathwohl. *Taxonomy of educational objectives: The classification of educational goals. Book 1, Cognitive domain*. longman, 2020 (cit. on pp. 1, 5, 6, 25, 27, 29, 82, 94, 97).
- [BMR+20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901 (cit. on pp. 11, 15).
- [BPH+20] X. Bai, H. Pan, J. Hou, T. Guo, I. Lee, F. Xia. “Quantifying success in science: An overview”. In: *IEEE Access* 8 (2020), pp. 123200–123214 (cit. on p. 1).
- [CB17] B. R. Cook, A. Babon. “Active learning through online quizzes: Better learning and less (busy) work”. In: *Journal of Geography in Higher Education* 41.1 (2017), pp. 24–38 (cit. on p. 1).
- [Chu17] K. W. Church. “Word2Vec”. In: *Natural Language Engineering* 23.1 (2017), pp. 155–162 (cit. on p. 54).
- [CM90] J.-J. Chen, P. Marwedel. “Petri Nets”. In: *IEEE Transactions on knowledge and data engineering* 2.3 (1990), pp. 311–319 (cit. on p. 20).
- [CND+22] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. “Palm: Scaling language modeling with pathways”. In: *arXiv preprint arXiv:2204.02311* (2022) (cit. on pp. 11, 14, 15).
- [Dal21] R. Dale. “GPT-3: What’s it good for?” In: *Natural Language Engineering* 27.1 (2021), pp. 113–118 (cit. on p. 12).

- [DCLT18] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on pp. 11, 13–15, 28).
- [Gra] D. Grazotin. *Infrastructure and Containerization of Web Applications: University of Stuttgart lecture*. Ilias. Accessed: 2023-10-08 (cit. on pp. 42, 68, 69).
- [JWHT+13] G. James, D. Witten, T. Hastie, R. Tibshirani, et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013 (cit. on p. 55).
- [Kar12] H. Kara. *Research and evaluation for busy practitioners: a time-saving guide*. Policy Press, 2012 (cit. on p. 1).
- [KM+06] N. Karamanis, R. Mitkov, et al. “Generating multiple-choice test items from medical text: A pilot study”. In: *Proceedings of the fourth international natural language generation conference*. 2006, pp. 111–113 (cit. on pp. 1, 9, 19, 101).
- [KN19] T. Klein, M. Nabi. “Learning to answer by learning to ask: Getting the best of gpt-2 and bert worlds”. In: *arXiv preprint arXiv:1911.02365* (2019) (cit. on pp. 1, 19).
- [KRS21] K. S. Kalyan, A. Rajasekharan, S. Sangeetha. “Ammus: A survey of transformer-based pretrained models in natural language processing”. In: *arXiv preprint arXiv:2108.05542* (2021) (cit. on pp. 11, 14).
- [LO04] C.-Y. Lin, F. Och. “Looking for a few good metrics: ROUGE and its evaluation”. In: *Ntcir workshop*. 2004 (cit. on p. 19).
- [MM05] R. Murray, K. Morss. “Teaching at university: A guide for postgraduates and researchers”. In: *Teaching at University* (2005), pp. 1–216 (cit. on p. 1).
- [MO03] P. Moulin, J. A. O’Sullivan. “Information-theoretic analysis of information hiding”. In: *IEEE Transactions on information theory* 49.3 (2003), pp. 563–593 (cit. on p. 22).
- [MS15] M. Majumder, S. K. Saha. “A system for generating multiple choice questions: With a novel approach for sentence selection”. In: *Proceedings of the 2nd workshop on natural language processing techniques for educational applications*. 2015, pp. 64–72 (cit. on pp. 1, 17–19).
- [NBM+22] H. A. Nguyen, S. Bhat, S. Moore, N. Bier, J. Stamper. “Towards Generalized Methods for Automatic Question Generation in Educational Domains”. In: *Educating for a New Future: Making Sense of Technology-Enhanced Learning Adoption: 17th European Conference on Technology Enhanced Learning, EC-TEL 2022, Toulouse, France, September 12–16, 2022, Proceedings*. Springer. 2022, pp. 272–284 (cit. on pp. 1, 20).
- [Nie94] J. Nielsen. “Enhancing the explanatory power of usability heuristics”. In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 1994, pp. 152–158 (cit. on p. 24).
- [Ope23] OpenAI. *GPT-4 Technical Report*. 2023. DOI: [10.48550/ARXIV.2303.08774](https://doi.org/10.48550/ARXIV.2303.08774). URL: <https://arxiv.org/abs/2303.08774> (cit. on pp. 8, 12, 17, 56, 57, 66).
- [PRWZ02] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu. “Bleu: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318 (cit. on p. 19).

- [RL77] C. V. Ramamoorthy, H. F. Li. “Pipeline architecture”. In: *ACM Computing Surveys (CSUR)* 9.1 (1977), pp. 61–102 (cit. on p. 20).
- [RSR+20] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P.J. Liu. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 5485–5551 (cit. on pp. 11, 14, 20, 28).
- [RWC+19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9 (cit. on p. 19).
- [SBZ17] M. Shahin, M. A. Babar, L. Zhu. “Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices”. In: *IEEE access* 5 (2017), pp. 3909–3943 (cit. on p. 70).
- [Spe] S. Speth. *Programierung und Software Entwicklung, Vortragsübung: University of Stuttgart*. Ilias. Accessed: 2023-10-08 (cit. on p. 56).
- [SSS21] Z. Sokhanvar, K. Salehi, F. Sokhanvar. “Advantages of authentic assessment for improving the learning experience and employability skills of higher education students: A systematic literature review”. In: *Studies in Educational Evaluation* 70 (2021), p. 101030 (cit. on p. 1).
- [Tay53] W.L. Taylor. ““Cloze procedure”: A new tool for measuring readability”. In: *Journalism quarterly* 30.4 (1953), pp. 415–433 (cit. on pp. 9, 14).
- [TCY] D. C. TSAI, W. J. CHANG, S. J. YANG. “Short Answer Questions Generation by Fine-Tuning BERT and GPT-2”. In: () (cit. on pp. 1, 19).
- [TKS+03] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T. C. Bressoud, A. Perrig. “Opportunistic Use of Content Addressable Storage for Distributed File Systems.” In: *USENIX Annual Technical Conference, General Track*. Vol. 3. 2003, pp. 127–140 (cit. on p. 52).
- [Tra] H. F. Transformers. *Hugging Face Transformers*. <https://downloads.cs.stanford.edu/nlp/software/tmt/tmt-0.4/>. Accessed: 2023-03-02 (cit. on pp. 11, 48, 103).
- [Unia] Z. f. L. u. W. |. z. Universität Stuttgart. “Leitfaden Items und Fragebogenkonstruktion”. In: () (cit. on pp. 1, 17, 25, 29, 40, 61, 93, 94).
- [Unib] Z. f. L. u. W. |. z. Universität Stuttgart. “Lernzieltaxonomie mit Beispielen”. In: () (cit. on pp. 5, 17).
- [Unic] Z. f. L. u. W. |. z. Universität Stuttgart. “Motivationsbedingungen: Ideen für die Umsetzung”. In: () (cit. on pp. 1, 17, 40, 94).
- [VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 9, 10, 14, 15, 17).
- [Weba] Webpage. *2023 Stackoverflow Developer Survey*. <https://survey.stackoverflow.co/2023/#technology>. Accessed: 2023-10-09 (cit. on pp. 48, 51).
- [Webb] Webpage. *Anki*. <https://docs.ankiweb.net/editing.html?highlight=cloze#cloze-deletion>. Accessed: 2023-03-16 (cit. on p. 9).

Bibliography

- [Webc] Webpage. *Anki cloze deletion*. <https://docs.ankiweb.net/editing.html?highlight=cloze#cloze-deletion>. Accessed: 2023-03-16 (cit. on pp. 9, 39, 102).
- [Webd] Webpage. *Axios*. <https://axios-http.com/docs/intro>. Accessed: 2023-10-10 (cit. on p. 67).
- [Webe] Webpage. *Black*. <https://github.com/psf/black>. Accessed: 2023-10-09 (cit. on p. 103).
- [Webf] Webpage. *BLAKE3*. <https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf>. Accessed: 2023-10-09 (cit. on pp. 52, 53, 103).
- [Webg] Webpage. *CPython*. <https://github.com/python/cpython>. Accessed: 2023-10-10 (cit. on p. 50).
- [Webh] Webpage. *CS.Ratelimit*. <https://github.com/CrowdStrike/cs.ratelimit>. Accessed: 2023-10-09 (cit. on p. 103).
- [Webi] Webpage. *Dev Container Build and Run (devcontainers/ci)*. <https://github.com/devcontainers/ci>. Accessed: 2023-10-10 (cit. on p. 70).
- [Webj] Webpage. *Docker*. <https://www.docker.com/>. Accessed: 2023-10-10 (cit. on pp. 51, 68, 69).
- [Webk] Webpage. *Eclipse QVTo*. <https://wiki.eclipse.org/QVTo>. Accessed: 2023-04-03 (cit. on p. 35).
- [Webl] Webpage. *Envoy*. <https://www.envoyproxy.io/>. Accessed: 2023-10-10 (cit. on p. 67).
- [Webm] Webpage. *European Union: General Data Protection Regulation (GDPR)*. <https://gdpr.eu/what-is-gdpr/>. Accessed: 2023-10-16 (cit. on p. 40).
- [Webn] Webpage. *Fork of Sphinx to GitHub Pages V3*. <https://github.com/legendofa/sphinx-notes-recursive>. Accessed: 2023-10-10 (cit. on p. 71).
- [Webo] Webpage. *Gensim*. <https://github.com/RaRe-Technologies/gensim>. Accessed: 2023-10-09 (cit. on pp. 48, 103).
- [Webp] Webpage. *GitHub Workflows*. <https://docs.github.com/en/actions/using-workflows>. Accessed: 2023-10-10 (cit. on pp. 69, 71).
- [Webq] Webpage. *Google Scholar*. <https://scholar.google.com/>. Accessed: 2023-10-15 (cit. on p. 17).
- [Webr] Webpage. *GPT 4*. <https://openai.com/product/gpt-4>. Accessed: 2023-03-16 (cit. on pp. 11, 12, 24, 28, 34, 35, 57).
- [Webs] Webpage. *GPT models*. <https://platform.openai.com/docs/models/gpt-4>. Accessed: 2023-03-16 (cit. on pp. 11–13, 35, 62).
- [Webt] Webpage. *gRPC on Node.js*. <https://github.com/grpc/grpc-node/tree/master>. Accessed: 2023-10-10 (cit. on p. 67).
- [Webu] Webpage. *gRPC Web*. <https://github.com/grpc/grpc-web>. Accessed: 2023-10-10 (cit. on p. 67).
- [Webv] Webpage. *Ilias Open Source LMS*. <https://www.ilias.de/en/>. Accessed: 2023-10-16 (cit. on p. 1).

- [Webw] Webpage. *Interplanetary File System: IPFS*. <https://www.ipfs.tech/>. Accessed: 2023-10-08 (cit. on p. 52).
- [Webx] Webpage. *JSONForms*. <https://jsonforms.io/>. Accessed: 2023-10-10 (cit. on pp. 66, 67).
- [Weby] Webpage. *Jsonpickle*. <https://github.com/jsonpickle/jsonpickle>. Accessed: 2023-10-09 (cit. on p. 103).
- [Webz] Webpage. *MaterialUI*. <https://mui.com/material-ui/>. Accessed: 2023-10-10 (cit. on p. 66).
- [Webaa] Webpage. *Microsoft Azure*. <https://azure.microsoft.com>. Accessed: 2023-10-10 (cit. on p. 69).
- [Webab] Webpage. *Microsoft Document Intelligence Studio*. <https://formrecognizer.appliedai.azure.com/studio>. Accessed: 2023-10-08 (cit. on p. 54).
- [Webac] Webpage. *Microsoft Forms*. <https://forms.office.com>. Accessed: 2023-10-16 (cit. on p. 73).
- [Webad] Webpage. *ML-ArXiv-Papers dataset*. <https://huggingface.co/datasets/CShorten/ML-ArXiv-Papers>. Accessed: 2023-10-08 (cit. on p. 54).
- [Webae] Webpage. *MyPy*. <https://github.com/python/mypy>. Accessed: 2023-10-09 (cit. on p. 103).
- [Webaf] Webpage. *Natural Language Toolkit*. <https://www.tensorflow.org/>. Accessed: 2023-04-03 (cit. on p. 103).
- [Webag] Webpage. *NGiNX*. <https://nginx.org/en/>. Accessed: 2023-10-10 (cit. on pp. 68, 69).
- [Webah] Webpage. *OpenAI libraries*. <https://platform.openai.com/docs/libraries>. Accessed: 2023-03-16 (cit. on pp. 12, 41, 50, 103).
- [Webai] Webpage. *OpenAI tokenizer*. <https://platform.openai.com/tokenizer>. Accessed: 2023-03-16 (cit. on pp. 12, 35, 103).
- [Webaj] Webpage. *OpenAI: Aligning language models to follow instructions*. <https://openai.com/research/instruction-following>. Accessed: 2023-10-11 (cit. on p. 13).
- [Webak] Webpage. *OpenAI: Learning from human preferences*. <https://openai.com/research/learning-from-human-preferences>. Accessed: 2023-10-11 (cit. on p. 13).
- [Webal] Webpage. *OpenAPI Specification*. <https://spec.openapis.org/oas/latest.html>. Accessed: 2023-04-03 (cit. on pp. 42, 51).
- [Webam] Webpage. *Pandoc Document Converter*. <https://pandoc.org/>. Accessed: 2023-10-08 (cit. on pp. 50, 53–56, 103).
- [Weban] Webpage. *pip package installer*. <https://pypi.org/project/pip/>. Accessed: 2023-10-09 (cit. on p. 50).
- [Webao] Webpage. *PlantUML*. <https://plantuml.com>. Accessed: 2023-10-10 (cit. on p. 70).
- [Webap] Webpage. *Portable Document Format (PDF)*. <https://pdfa.org/resource/iso-32000-pdf/>. Accessed: 2023-10-08 (cit. on p. 54).
- [Webaq] Webpage. *PPTX2MD*. <https://github.com/ssine/pptx2md>. Accessed: 2023-10-08 (cit. on pp. 55, 56, 103).

Bibliography

- [Webar] Webpage. *Protobuf to JSON-Schema compiler*. <https://github.com/chrusty/protoc-gen-jsonschema>. Accessed: 2023-10-10 (cit. on p. 68).
- [Webas] Webpage. *Protocol Buffers*. <https://protobuf.dev/>. Accessed: 2023-10-09 (cit. on p. 103).
- [Webat] Webpage. *PyFlakes*. <https://github.com/PyCQA/pyflakes>. Accessed: 2023-10-09 (cit. on p. 103).
- [Webau] Webpage. *PyMongo*. <https://github.com/mongodb/mongo-python-driver>. Accessed: 2023-10-09 (cit. on pp. 51, 103).
- [Webav] Webpage. *PyPandoc*. <https://pypi.org/project/py pandoc/>. Accessed: 2023-10-08 (cit. on p. 103).
- [Webaw] Webpage. *PyTest*. <https://github.com/pytest-dev/pytest>. Accessed: 2023-10-09 (cit. on p. 103).
- [Webax] Webpage. *Python Better Protobuf Library*. <https://github.com/danielgtaylor/python-betterproto>. Accessed: 2023-10-09 (cit. on pp. 48, 51, 68, 103).
- [Webay] Webpage. *Python Package Index (PyPI)*. <https://pypi.org/>. Accessed: 2023-10-09 (cit. on p. 50).
- [Webaz] Webpage. *PyTorch*. <https://pytorch.org/>. Accessed: 2023-04-03 (cit. on p. 103).
- [Webba] Webpage. *React*. <https://react.dev/>. Accessed: 2023-10-10 (cit. on p. 66).
- [Webbb] Webpage. *React App*. <https://create-react-app.dev/>. Accessed: 2023-10-10 (cit. on p. 69).
- [Webbc] Webpage. *Rust Programming Language*. <https://www.rust-lang.org/>. Accessed: 2023-10-10 (cit. on pp. 50, 52, 103).
- [Webbd] Webpage. *Scrum Guide*. <https://www.scrum.org/resources/scrum-guide>. Accessed: 2023-04-03 (cit. on p. 35).
- [Webbe] Webpage. *Setuptools*. <https://github.com/pypa/setuptools>. Accessed: 2023-10-09 (cit. on p. 104).
- [Webbf] Webpage. *Sphinx*. <https://github.com/sphinx-doc/sphinx>. Accessed: 2023-10-09 (cit. on pp. 71, 103).
- [Webbg] Webpage. *Sphinx to GitHub Pages V3*. <https://github.com/sphinx-notes/pages>. Accessed: 2023-10-10 (cit. on p. 71).
- [Webbh] Webpage. *Stanford Topic Modeling Toolbox (TMT)*. <https://downloads.cs.stanford.edu/nlp/software/tmt/tmt-0.4/>. Accessed: 2023-03-23 (cit. on p. 18).
- [Webbi] Webpage. *Tensorflow*. <https://www.tensorflow.org/>. Accessed: 2023-04-03 (cit. on p. 103).
- [Webbj] Webpage. *University of Stuttgart, overview of computer science courses*. <https://www.f05.uni-stuttgart.de/informatik/studierende/bachelor/>. Accessed: 2023-03-26 (cit. on p. 28).
- [Webbk] Webpage. *Visual Studio Code*. <https://code.visualstudio.com/>. Accessed: 2023-10-10 (cit. on p. 51).

- [WL20] Z. Wang, J.-C. Liu. “Pdf2latex: A deep learning system to convert mathematical documents from pdf to latex”. In: *Proceedings of the ACM Symposium on Document Engineering 2020*. 2020, pp. 1–10 (cit. on p. 54).
- [WWS+22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, D. Zhou. “Chain of thought prompting elicits reasoning in large language models”. In: *arXiv preprint arXiv:2201.11903* (2022) (cit. on p. 15).

A Didactic Specialist Interviews

This chapter provides the full didactic specialist interview memory protocols.

A.0.1 General Self-Assessment Interview

We present an interview about self-assessment in general:

Q1. *Are there general rules for academic question creation?*

A1. Question creation for Exams, exercises, etc. is usually guideline centered. It is important to consider each question in relation to its surrounding questions to arrange the sequence in which questions are asked carefully. Asking questions is an act of communication. **Questions that are topically similar should adjacent or at least close to each other [Unia] F-REQ-11. Topic jumps should be prevented as well as possible F-REQ-12.** In order to connect sections of different topics, transition elements should be designed. Topic sections are marked by headlines. The inquiry purpose is best stated at the beginning of the document, followed by an "icebreaker" question, sparking interest and motivation to continue [Unia].

An "icebreaker" question ideally is [Unia]:

- Connected with the following topics
- Connected to the person involved
- Simple to answer
- Answerable with none or small amounts of previous knowledge

The most important questions should be placed in the first third of the inquiry. F-REQ-13 Each topic should be assessed from general to specific questions [Unia]. Context from previous questions can mislead or bias to specific answers. More information about context effects can be found in the "Item and question construction guidelines" of the ZLW University of Stuttgart [Unia], as well as additional question-wording guidelines.

Q2. *How to estimate the difficulty of a specific question?*

A2. Statistical analysis on exam results enables question difficulty assessment. There are also methods presented in social science literature.

Q3. *How to assess how much time is required for answering a specific question?*

A3. Measuring time during a test-wise solve has proven itself as effective.

Q4. *How to formulate questions in a precise manner?*

A4. ZLW question wording guidelines [Unia] **F-REQ-14**:

- Simple and clear language
- Simple and short questions
- Positive language
- Not suggestions or assumptions

Q5. *Is there a specific order in which questions should be created?*

A5. There are specific 'dramaturgy' rules, see ZLW [Unia] to improve motivation and maintain a certain flow while answering questions. Multiple motivational factors play a role in question creation.

According to ZLWs "Motivational requirements" [Unic]:

- **Autonomy**: Choices and room to explore for students. Self-directed learning by planning, creation, and action.
- **Perception of competence**: Constructive feedback about progress. Suitable choice of comparison standard (social, individual, by criteria). Adapted level of difficulty in exercises and exams.
- **Social inclusion**: Acceptance of and empathy for students. Patience: acknowledgment of individual strengths and weaknesses while offering individual support. Mutually respectful interaction.
- **Thematic interest of the teacher**: Articulation and emotional involvement regarding subject. Enthusiasm and passion for teaching the specific subject and its topics.
- **Thematic relevance**: Problem oriented and practically relevant teaching. Authentic and realistic perspective. Highlighting the connection between theory and practice. Informing about interdisciplinary context. Practical examples
- **Instruction quality**: Precise statement of educational objectives: standards and/or criteria. Clear course structure. Detailed and comprehensible instructions. Adaptive difficulty level. 'Dramaturgy' considerations lead from concrete to abstract or the other way around.

Q6. *How to decide which course parts to assess?*

A6. Questions can be composed based on educational objectives [BK20] **F-REQ-15**, which focus on specific lecture parts. The idea is to **measure to what degree an educational objective is satisfied F-REQ-16**. Educational objectives are not easy to measure, as they have different aspects and depths. For example: What does it mean to have understood a concept? A definition for understanding can be interpreted in different ways.

Q7. *How can students benefit the most from self-assessment?*

A7. This is yet to be explored and empirically tested. Following guidelines for course and question creation posed by educational sciences is a good starting point. Currently, there still is a lot of unleashed potential within different forms of individualized teaching. Speaking of offering students with different previous capabilities the option to study at their own pace.

A.0.2 Interaction Prototype Interview

We present an interview about the interaction prototype, described in Section 4.2.2:

Q8. *How to approach question generation? One question after the other or a whole document with questions at once?*

A8. The goal of question creation plays an important role in figuring out which approach suits best. A system allowing both approaches can help answer this question. User analytics can help with data collection and interpretation.

Q9. *How important is traceability for question generation? To know which lecture material generates which question.*

A9. Traceability is most relevant when there are doubts about the result's validity. Referencing and rereading the source might help with clarifying misconceptions.

Q10. *Are there scenarios where generation should exclude specific lecture material parts?*

A10. If the lecturer plans to leave out specific parts for the exam.

B Question Specification

YAML is chosen as the specification syntax due to its advantages. It is human-readable and machine-readable. However, the implemented system uses equivalent JSON. We describe the question specification in YAML, as it is more concise than JSON.

The specification should be extendable and updatable, therefore the first field under the root is version, initially set to "1.0".

The specifications next part states general information about the course, mandatory for self-assessment generation. This section formulates course goals. Furthermore, general assumptions about student background and former capabilities are specified. `required_capabilities` describes course preconditions and `advantageous_capabilities` optional capabilities that benefit the student.

`course_goals`, `required_capabilities` and `advantageous_capabilities` can be chosen as one of 6 possible capability categories: `KNOW_AND_UNDERSTAND`, `APPLY`, `ANALYZE`, `SYNTHESIZE`, `EVALUATE`, `INNOVATE`. Those categories reflect the educational objectives according to Bloom's taxonomy [BK20]. `relationship` describes the relationship between keywords, for multiple keywords. Possible relationships are: `SIMILARITY`, `DIFFERENCE`, `ORDER`. If it is unclear which aspect of the relationship to highlight, `COMPLEX` can be chosen.

Example:

```
course_settings:
  course_goals:
    - capability:
      category: "KNOW_AND_UNDERSTAND"
      keywords:
        - "keyword_1"
        - "keyword_2"
      relationship: "SIMILARITY"
  required_capabilities:
    - capability:
      category: "EVALUATE"
      keywords:
        - "keyword_3"
        - "keyword_4"
      relationship: "COMPLEX"
  advantageous_capabilities:
    - capability:
      ...
```

`generation_settings` define the global behavior of question generation.

Example:

B Question Specification

```
course_settings:
...
generation_settings:
  mode:
    by_metrics:
      evaluation_reference: "language_clarity"
      evaluator_type: "le"
      evaluation_result:
        float_value: 1.0
```

There are three generation modes:

- complete is the default mode. All questions with a missing result section are generated.
- overwrite regenerates all questions, disregarding existing result sections.
- by_metrics, applies an evaluator of evaluator_type

complete and overwrite are objects without any parameters. by_metrics requires an evaluation_reference, evaluator_type and evaluation_result. The evaluation_reference references the result of an evaluation. Each question maps evaluation_results to evaluation_references in the evaluation_results dictionary. evaluator_type specifies the comparison. The following comparisons are supported:

```
"eq": "x == y"
"neq": "x != y"
"geq": "x >= y"
"leq": "x <= y"
"ge": "x > y"
"le": "x < y"
"is": "x is y"
"is not": "x is not y"
"in": "x in y"
"part_of": "y in x"
```

evaluation_settings describe how generated questions are evaluated. Resulting metrics give lecturers more information to decide about generated questions. Evaluation metrics can be used by a learning management system to categorize and personalize questions.

Example:

```
generation_settings:
...
evaluation_settings:
  - metric:
      reference: "language_clarity"
      evaluation:
        language_model_evaluation:
          ... # parameters
      mode: "complete"
```

A metric consists of a unique reference, an evaluation, and parameters based on the evaluation. mode behaves similar to the mode parameter in generation_settings.

language_model_evaluation enables to use self-reflective capabilities of a language model to evaluate its output.

Example:

```
evaluation:
  language_model_evaluation:
    model: "gpt-4"
    evaluation_evaluation_description: "The given question is formulated precisely and uses standard
    ↪ English."
    few_shot_examples:
      ...
    evaluation_result_type:
      value_range:
        lower_bound: 0.0
        upper_bound: 1.0
```

model describes the language model used for the evaluation. evaluation_description states how questions are evaluated. few_shot_examples is a list of generation_evaluation_results. generation_evaluation_results include a generation_result and a evaluation_result.

Example:

If the evaluation_result_type is a value_range: [lower_bound, upper_bound] is required, which describes the maximum and minimum of value_range. Another option is the categorical evaluation_result_type, which is a set of categories:

```
evaluation_result_type:
  categorical:
    - "Low precision"
    - "Average precision"
    - "High precision"
```

Next, a unique list of material server URLs describes where to resolve materials. Those material servers represent endpoints where materials can be queried using a hash (content addressable storage).

```
evaluation_settings:
  ...
  materialServerURLs:
    - "materialserver1.example.com"
    - "materialserver2.example.com"
```

The general format of the question specification is designed to resolve question formulations that are inconsistent, in order to iteratively improve the questions that are output. For each batch of generated questions, all input parameters are preserved and can be adjusted in order to regenerate a question.

batches is a unique list of batch instances.

A batch consists of materials, capabilities and questions_to_generate. capabilities are of the same datatype as in course_goals, required_capabilities and advantageous_capabilities under the course_settings.

```
materialServerURLs:
  ...
  batches:
    - batch:
      materials:
```

B Question Specification

```
...
capabilities:
...
questions_to_generate:
...
- batch:
...
...
```

materials is a unique list of material datatypes.

The material consists of a unique for each file reference, optional url, hash and optional page_filter, defining an interval of pages (both included). The hash is calculated with the material itself, to enable detection of possible inconsistencies upon regeneration. All metadata is retrieved upon download. A url is optional, as requesting materials through specified material servers only requires the hash. To enable linking resources from arbitrary servers via HTTP, URLs can be added.

Example:

```
material:
  reference: "textbook.pdf"
  url: "example.com/textbook.pdf"
  hash: "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
  page_filter: [12, 340]
```

Materials can also be directly referenced by the reference name for ease of use, but need to be fully specified in materials following batches

Example:

```
batches:
- batch:
  materials:
    - "textbook.txt"
    - "lecture.pdf"
  ...
...
materials:
- material:
  reference: "lecture.pdf"
  hash: "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
```

To enable capability-based generation, a unique capability list is used, with at least one capability required.

Example:

```
capabilities:
- capability:
  category: "EVALUATE"
  keywords:
    - "keyword_1"
- capability:
  category: "KNOW_AND_UNDERSTAND"
  keywords:
    - "keyword_2"
    - "keyword_3"
```

```
relationship: "SIMILARITY"
```

```
...
```

questions_to_generate is a list of question datatypes.

A question consists of a question_type, describing the specific type of question and result, which is different for each question type and an evaluation section.

Example:

```
question:
```

```
- question_type: "MULTIPLE_CHOICE"
- generation_result: # Result of question generation.
...
- evaluation_results: # Dictionary with evaluation results.
...
```

evaluation_results reference evaluation_results with evaluation_references. Evaluation results can be either strings using the field: str_value or floating point numbers using float_value.

Example:

```
evaluation_results:
  language_clarity:
    float_value: 0.4
```

The question specification describes which question types are supported. The following question types are described in further detail.

Currently supported question types:

- **Multiple-Choice**

A Multiple-Choice question describes a question_text and a set of possible answers, rendering 1 exact answer and 1..n distractors. Distractors are answers that are not true. This is modeled as the following: The question_text, as well as the answer_text, are modeled as strings. The distractor_texts is a set of Strings.

Example:

- **Multiple-Response**

Consist of a sentence formulating a question, 0..n answer options of which 0..n answers can be true. Wrong answers are described as distractors [KM+06].

Example:

```
...
- type: "MULTIPLE_RESPONSE"
- generation_result:
  "question_text": "Which atoms are contained in water?",
  "answer_texts": ["H (Hydrogen)", "O (Oxygen)"],
  "distractor_texts" ["He (Helium)", "Fe (Iron)"]
```

The question specification can be extended with further types. We want to convey that our approach generalizes well beyond checkbox-type questions. Currently not included, but proposed cloze variant:

- **Multiple-Choice Cloze**

Example:

```
...
- type: "multiple_choice_cloze"
- generation_result:
  "question_text": "Water is an inorganic compound with the chemical formula {{c1::H_2O}}"
  "distractor_texts": [{"CH_4", "C_8H_{10}N_{40}_2", "CO"}]
```

Cloze question formatting is described in detail by the Anki documentation [Webc]. Cloze questions are formatted as `{{ID::Cloze::Hint}}`, where ID is formatted as `cn` with $n \in \{1, \dots, k\}$ starting from 1 and increasing in steps of 1, Cloze describes the hidden text. Hint describes a hint that can be shown, without revealing the answer.

We can construct an example with multiple cloze fields and a hint:

```
...
- generation_result:
  "question_text": "The {{c1::registrar::enables to lease second-level domains}}
  ↪ communicates the authoritative nameservers to the domain registry for the
  ↪ {{c2::top-level}} zone."
  "distractor_texts": [
    ["internet service provider", "government", "signing agency"],
    ["first-level", "second-level", "entire"]
  ]
```

Distractors are ordered similarly to the cloze fields. The additional hint is described after the `::`: "Enable to lease second-level domains".

All data types are defined in `evalquiz.proto` of the `evalquiz_proto` repository, see chapter C.

C Implementation Libraries and Repositories

- **OpenAI API** [Webah] offers access to a variety of OpenAIs language models.
- **NLTK (Natural Language Toolkit)** [Webaf] is a NLP text-processing library for tokenization, stemming, tagging, parsing, semantic reasoning, etc.
- **Better Protobuf** [Webax], a gRPC generator library mapping Protocol Buffer [Webas] definitions to Python classes. gRPC server/client stub generation from Protobuf schemas. Builtin binary and JSON serialization. MyPy support.
- **BLAKE3** bindings for Rust library [Webf] [Webbc]. BLAKE3 is a highly parallelizable cryptographic hash based on the BLAKE architecture.
- **PyTest** [Webaw] testing framework for python. Supports unit tests, async testing, debugging in tests, fixtures, and parameterized tests.
- **MyPy** [Webae] duck typing checker for Python type annotations.
- **PyFlakes** [Webat] static code analysis tool. Finds unused variables, imports not reachable code and more.
- **Black** [Webe] code formatter.
- **Jsonpickle** [Weby] serializes and deserializes Python objects to JSON.
- **Sphinx** [Webbf] with Read The Docs (RTD) theme. A document engine compatible with Google-style doc strings. Sphinx features auto-indexing of classes and methods with type annotations.
- **PyMongo** [Webau], the official MongoDB driver for python.
- **Gensim** [Webo] library for topic modeling, document indexing and similarity retrieval.
- **Tiktoken** [Webai], OpenAIs official tokenizer used for their language models.
- **PPTX2MD** [Webaq] a PPTX to markdown converter. Transforms titles, tables, lists, formatted text, pictures and more to markdown.
- **PyPandoc** [Webav] lightweight wrapper for Pandoc [Webam]. Provides version including Pandoc binary.
- **Datasets** by Hugging Face [Tra]. Part of one of the most popular ML libraries offering ease-of-use Transformer models, datasets and integration with PyTorch [Webaz] and Tensorflow [Webbi], for model usage and training.
- **CS.Ratelimit** [Webh], a thread-safe rate limiter for API requests.

- **Setuptools** [Webbe], a packaging library based on python's builtin distutils. Used for specifying package metadata.

Technical documentation:

- <https://meitrex.github.io/evalquiz-pipeline-server/index.html>
- <https://meitrex.github.io/evalquiz-material-server/>

EvalQuiz GitHub repositories:

- <https://github.com/MEITREX/evalquiz-full-stack>
- <https://github.com/MEITREX/evalquiz-client-react>
- <https://github.com/MEITREX/evalquiz-client-flask>
- <https://github.com/MEITREX/evalquiz-pipeline-server>
- <https://github.com/MEITREX/evalquiz-material-server>

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Herrenberg, 25.10.2023

place, date, signature

A handwritten signature in black ink, reading "Julian Kriehinger", written over a horizontal line.