Universität Stuttgart

Die MarieQurie-Methode zur Empfehlung von Quantenressourcen

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

Vorgelegt von

Marie Olivia Salm

aus Herrenberg

Hauptberichter:	Prof. Dr. Dr. h. c. Frank Leymann
Mitberichter:	Prof. Dr. Stefanie Scherzinger

Tag der mündlichen Prüfung: 15. Oktober 2024

Institut für Architektur von Anwendungssystemen

der Universität Stuttgart

2024

Inhaltsverzeichnis

Zusammenfassung

Heutige Quantencomputer unterliegen hohen Fehlerraten, wodurch ihr Einsatz zur Lösung realer Probleme noch eingeschränkt ist. Dennoch findet Quantencomputing in immer mehr Bereichen Anklang, wodurch der praktische Nutzen in verschiedenen Anwendungsszenarien getestet wird. Ebenso steigt die Zahl verschiedener Hardwareanbieter und deren Angebot an zur Verfügung stehender Quantencomputer. Aufgrund fehlender Standards besteht eine Heterogenität der Softwareprodukte und Schnittstellen zur Programmierung von Quantencomputern, welches die Entwicklung und Ausführung von Quantenalgorithmen erschwert. Fehlt Nutzenden zudem die nötige Expertise im Quantencomputing, bildet außerdem die Selektion eines geeigneten Quantencomputers für die Ausführung große Schwierigkeiten.

Um den genannten Schwierigkeiten entgegenzutreten, stellt diese Arbeit die MarieQurie-Methode zur automatisierten Empfehlung von Quantencomputern, Quantenalgorithmenimplementierungen, Quantenschaltkreisen und Quantencompilern, im Folgenden Quantenressourcen genannt, vor. Die Methode selektiert zunächst Implementierungen für gewählte Quantenalgorithmen. Für die Ausführung von Quantenschaltkreisen, die von den Implementierungen basierend auf zu lösenden Probleminstanzen erzeugt werden, ordnet die MarieQurie-Methode geeignete Quantencompiler und

-computer diesen nach den Anforderungen der Nutzenden zu. Die Selektion der Quantenressourcen basiert auf Vorhersagen maschinell lernender Algorithmen anhand historischer Ausführungen. Um die Softwareheterogenität zu lösen, wendet die MarieQurie-Methode ein Übersetzungsverfahren an, sodass die Quantenschaltkreise mit diversen Quantencompilern und -computer kompatibel sind. Bei der Untersuchung der Ausführbarkeit von Quantenschaltkreisen bezüglich Fehlern, die bei der Ausführung auftreten können, werden verschiedene Arten der Initialisierung von Probleminstanzen für Quantencomputer präsentiert, welche die ausführbarkeitsbeeinflussende Größe der Quantenschaltkreise mitbestimmen. Aufgrund der zusätzlich größenverändernden Kompilierung der Quantenschaltkreise mit den gewählten Quantencompilern bestimmt die MarieQurie-Methode deren Ausführbarkeit anhand der resultierenden Kompilationen unter Berücksichtigung aktueller Quantencomputermetrikwerte. Bei mehreren ausführbaren Kompilationen werden präferenzbasierte Priorisierungsmechanismen angewendet, um den Nutzenden bei der Wahl geeigneter Kompilationen für die Ausführung zu unterstützen.

Für die automatisierte Umsetzung der MarieQurie-Methode wird die Architektur des MarieQurie-Frameworks sowie eine prototypische Implementierung dessen vorgestellt. Das Framework wird mit diversen Experimenten sowie Testpersoneninterviews evaluiert.

Abstract

Today's quantum computers are prone to high error rates, limiting their usage in solving real-world problems. Nevertheless, quantum computing becomes increasingly popular in more and more areas, which means that its practical benefits are being tested in various application scenarios. Also, the number of different hardware providers and their offerings of available quantum computers grows. Furthermore, due to the lack of standards, there is a heterogeneity of software products and interfaces for programming quantum computers, which makes the development and execution of quantum algorithms more difficult. In addition, if users are missing the necessary expertise in quantum computing, the selection of a suitable quantum computer for execution causes major difficulties.

To tackle these difficulties, this thesis presents the MarieQurie method for the automated recommendation of quantum computers, quantum algorithm implementations, quantum circuits, and quantum compilers, hereafter referred to as quantum resources. The method first selects implementations for chosen quantum algorithms. For the execution of quantum circuits generated by the implementations based on problem instances to be solved, the MarieQurie method assigns suitable quantum compilers and computers to these according to the user's requirements. The selection of quantum resources is

based on predictions of machine-learning algorithms using historical executions. To solve the software's heterogeneity problem, the method applies a translation procedure so that the quantum circuits are compatible with various quantum compilers and computers. When investigating the executability of quantum circuits concerning errors that can occur during execution, different types of initialization of problem instances for quantum computers are presented, determining the executability-influencing size of the quantum circuits. Due to the additional size-changing compilation of the quantum circuits with the selected quantum compilers, the MarieQurie method determines their executability based on the resulting compilations, considering up-to-date quantum computer metric values. When several executable compilations are present, preference-based prioritization mechanisms are applied to support the user in selecting suitable compilations for execution.

For the automated implementation of the MarieQurie method, the architecture of the framework and a prototypical implementation is presented. The MarieQurie framework is evaluated based on various experiments, as well as interviews with test users.

Danksagungen

Ich möchte mich von Herzen bei all denjenigen bedanken, die mich in den letzten Jahren bei der Entstehung dieser Dissertation begleitet haben. Mein ganz besonderer Dank gilt meinem Doktorvater Prof. Dr. Dr. h. c. Frank Leymann, der mich in dieser Zeit stets in meinem Vorhaben ermutigt hat und jederzeit ein offenes Ohr für mich hatte. Bei Prof. Dr. Stefanie Scherzinger möchte ich mich für die Übernahme des Mitberichts bedanken.

Auch meinen Kollegen am Institut möchte ich einen tiefen Dank aussprechen, die meine Zeit dort für mich in vielerlei Hinsicht einzigartig und wertvoll gestaltet haben. Hierbei möchte ich mich bei Prof. Dr. Uwe Breitenbücher und Dr. Karoline Wild bedanken, die sich anfangs für mich der Rolle der Mentoren annahmen und stets einen wertvollen Rat wussten. In diesem Zuge möchte ich außerdem Dr. Benjamin Weder für die zahlreichen Denkanstöße und Ratschläge sowie dem unermüdlichen Beistand als Freund und Kollege danken. Mein Dank gilt ebenfalls Philipp Wundrack, auf dessen Unterstützung ich stets zählen konnte. Manuela Weigold danke ich für die harmonische und produktive Zusammenarbeit. Des Weiteren möchte ich mich bei Martin Beisel, Ghareeb Falazi, Dr. Lukas Harzenetter, Alexander Mandl, Julian Obst, Felix Truger und Daniel Vietz für die schöne und abwechslungsreiche Zeit am Institut sowie die tolle Kollegialität bedanken.

Ein besonderes Dankeschön geht an meine Eltern, Geschwister, meinen Schwager, meine weiteren Familienmitglieder und meine Freunde für ihren nie endenwollenden mentalen Beistand und Rat. Dabei möchte ich insbesondere meinem Vater Ingolf Salm dafür danken, dass er stets an mich glaubt, mich dazu motiviert, meinen Weg zu gehen und mich immer wieder dazu anregt, Situationen mit mehr Leichtigkeit zu betrachten. Tobias Groß möchte ich für viele Jahre seiner Unterstützung danken. Nicht zuletzt gilt mein Dank Marvin Denk, der mich während der Fertigstellung dieser Dissertation geduldig und mit voller Hingabe durch Höhen und Tiefen begleitet hat und mir die notwendige Abwechslung bot. Bei seinen Eltern möchte ich mich außerdem für ihre Offenherzigkeit und ihr Verständnis bedanken.

Kapitel 1

Einleitung & Motivation

Quantencomputing ist in den verschiedensten Fachgebieten als eine relevante und zukunftsträchtige Technologie in den Fokus gerückt [LBF19; Pla24]. Mit den Gesetzen der Quantenmechanik, nach denen Quantencomputer agieren, ist es möglich, verschiedene Probleme beispielsweise des Finanzsektors oder der Naturwissenschaften signifikant effizienter zu lösen, als es bisher klassische Computer ermöglichen [Nat19; RP11]. Erste Experimente haben diesen *Quantenvorteil* bereits belegen können [Aru+19; ZWD+20]. Dennoch sind heutige Quantencomputer durch ihre Fehleranfälligkeit geprägt, wodurch nur wenige Operationen präzise ausgeführt und kleine Zahlen an *Quantenbits (Qubits)* zur Verfügung gestellt werden können [Pre18]. Diese Beeinträchtigungen verhindern heutzutage die Anwendbarkeit von Quantencomputing auf relevante Probleminstanzen.

Nichtsdestotrotz bieten Hersteller stets neue Quantencomputer an [LaR19]. Jeder Quantencomputer verfügt über individuelle Hardwareeigenschaften und unterscheidet sich unter anderem in den Fehlerraten, Operationen und verfügbaren Qubits von anderen. Insbesondere ist die Qubitzahl ein limitierender Faktor für die Ausführung von Quantenalgorithmen, wodurch

Nutzende häufig von langen Wartezeiten für den von Hardwareanbietern regulierten Rechenzugriff auf den qubitstarken Quantencomputern ausgehen müssen. Diese zeichnen sich jedoch nicht zwingend durch geringere Fehlerraten aus, da die Kontrollierbarkeit mit zunehmender Qubitzahl abnimmt und so weniger Operationen hintereinander ausgeführt werden können [LB20].

Es gibt bereits eine Vielzahl an Anwendungsfällen mit Implementierungen von Quantenalgorithmen zur Ausführung auf Quantencomputern. Quantenalgorithmen sind hybrid und enthalten sowohl Rechenanteile für den klassischen als auch für den Quantencomputer [BL22; LB20]. Entsprechend bestehen die Implementierungen aus klassischen und Quantenprogrammen, deren Daten- und Kontrollfluss mit sogenannten Workflows orchestriert werden können [LR00; VBLW22; WBLW20]. Mit den klassischen Programmen werden unter anderem die klassischen Daten der gegebenen Probleminstanz aufbereitet, um sie in einen Quantenzustand zu kodieren, sowie *Quantenschaltkreise* für die Berechnung auf dem Quantencomputer generiert [Ley19; WBLS20]. Außerdem werden Ausführungsergebnisse des Quantencomputers gegebenenfalls klassisch weiterverarbeitet [BBG+23; BL22; LB20; WBLV22]. Die Problemgröße beziehungsweise Eingabelänge der zu lösenden Instanz beeinflusst dabei die Anzahl der notwendigen Operationen und Qubits eines Quantenschaltkreises. Vor der Ausführung wird der Schaltkreis durch die Kompilierung auf die spezifische Hardware des gewählten Quantencomputers angepasst. Hierfür bildet ein *Quantencompiler* den auszuführenden Schaltkreis auf die verfügbaren Operationen und Qubits des Quantencomputers ab und wendet Optimierungsmethoden an, um die Anzahl der Operationen zu minimieren [SDC+20]. Dennoch führt das Vorgehen meist zu einer Vergrößerung des Schaltkreises und gefährdet aufgrund der hohen Fehlerraten eine präzise Berechnung auf heutiger Quantenhardware [LB20].

Quantenimplementierungen beziehungsweise -schaltkreise werden meist mit *Software Development Kits (SDKs)* entwickelt, kompiliert und ausgeführt [LaR19]. Durch SDKs werden Nutzenden Werkzeuge an die Hand gegeben, welche ihnen die Programmierung von Quantencomputer durch zusätzliche Abstraktionen in Form von vorgefertigten Funktionen erleichtern.

Hersteller liefern meist eigene SDKs für die Entwicklung und Ausführung auf ihren Quantencomputern aus. Diese SDKs enthalten unter anderem die spezifischen Operationen der proprietären Quantencomputer. Die individuellen Hardwareeigenschaften wie Fehlerraten von Quantencomputern anderer Hersteller werden üblicherweise nicht bereitgestellt. Herstelleragnostische SDKs [AG20; SDC+20] unterstützen den Zugang zu Quantencomputern verschiedener Hersteller, dennoch sind nicht alle Hersteller integriert. Hinzukommt, dass jedes SDK zumeist seinen eigenen Quantencompiler mit individuellen Abbildungs- und Optimierungsmethoden zur Verfügung stellt, sodass sich die kompilierten Schaltkreise zwischen den verschiedenen Compilern signifikant in Größe und Struktur unterscheiden können [KIMK22; SDC+20]. Bisher wurde keine Programmiersprache als Standard für die Quantencomputerprogrammierung eingeführt. Stattdessen unterstützen SDKs meist ihre eigenen Formate beziehungsweise Programmbibliotheken, die auf höherer Abstraktionsebene wie Java oder Python oder auf tieferer, ähnlich zu klassischem Assembler angesiedelt sind [BLF+21].

Die Heterogenität der SDKs verhindert eine bereits entwickelte Implementierung ohne zusätzlichen Aufwand auf ein beliebig anderes SDK zu portieren. Dies erschwert unterschiedliche Quantencompiler heranzuziehen, um verschiedene Kompilationen des erzeugten Quantenschaltkreises vergleichen und eine ressourcensparsame Kompilation für die darauffolgende Ausführung zu wählen. Außerdem wird erschwert, den Schaltkreis auf Quantencomputern beliebiger Hersteller auszuführen. Zusätzlich findet die Implementierung von Quantenschaltkreisen mit SDKs häufig auf der hardwarenahen Ebene von Quantenoperationen statt, wodurch seitens der Nutzenden ein großes Vorwissen in Mathematik und Quantencomputing vorausgesetzt wird, um die Programmierung zu beherrschen. Alternativ kann auf bestehende Implementierungen zurückgegriffen werden. Bei vorhandener Implementierung, müssen Nutzende ermitteln, welcher der über das zugehörige SDK verfügbaren Quantencomputer zur Ausführung verwendbar ist. Jedoch bedarf es dafür Kenntnisse über die Eigenschaften der Quantencomputer und des Schaltkreises, um präzise Ausführungsergebnisse zu erlangen.

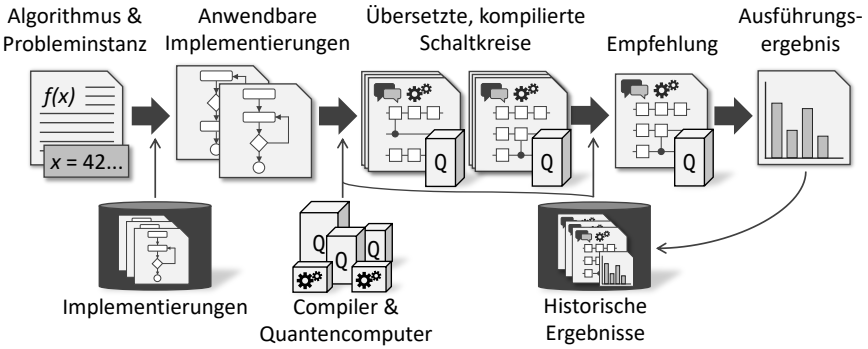

Abbildung 1.1: Ziel: Automatisierte Empfehlung von Quantenressourcen

1.1 Ziel der Arbeit

Das Ziel der vorliegenden Arbeit ist es, diese Probleme mit einer Methode zur Übersetzung, Kompilierung, Empfehlung und Ausführung von Quantenschaltkreisen gegebener Quantenalgorithmenimplementierungen für geeignete Quantencomputer zu lösen. Präsentiert wird dieses Ziel in Abbildung 1.1: Als Ausgangspunkt wählen Nutzende einen geeigneten Quantenalgorithmus sowie die zu lösende Probleminstanz. Daraufhin werden vorhandene Implementierungen des Algorithmus selektiert. Den gewählten Implementierungen wird die Probleminstanz übergeben, um die daraus resultierenden Quantenschaltkreise zu generieren. Die Schaltkreise werden bei Bedarf in die Formate übersetzt, die von den SDKs der verfügbaren Quantencompiler benötigt werden. Anschließend werden die Schaltkreise durch die Compiler auf die Qubits und Operationen existierender Quantencomputer abgebildet. Die Kompilationen werden bezüglich ihrer Ausführbarkeit analysiert, sodass Nutzenden, falls vorhanden, geeignete Kompilationen mit zugehörigen Quantencomputern empfohlen werden und diese ausgeführt werden können. Die Ergebnisse werden schließlich evaluiert und als Datenbasis für die Analyse und die Selektion von Schaltkreisen, Compilern und Quantencomputern für zukünftige Methodenaufrufe verwendet.

Das beschriebene Ziel ermöglicht automatisiert geeignete Quantenressourcen für die Anwendung eines Quantenalgorithmus auf eine zu lösende Probleminstanz zu selektieren und diese den Nutzenden bereitzustellen. Dabei abstrahiert die Methode von den technischen Details sowie der heterogenen Handhabung der zunehmenden Anzahl an verschiedenen Quantenimplementierungen, Quantencompiler und Quantencomputer hinweg. Dies vereinfacht den Zugang und die Anwendung des Quantencomputings für Personen mit geringer Expertise. Nichtsdestotrotz kann die Methode auch Personen mit umfangreicher Expertise unterstützen, um beispielsweise den manuellen Aufwand zu reduzieren, einen geeigneten Quantencompiler und -computer für die Ausführung einer spezifischen Implementierung zu wählen. Durch das automatisierte Übersetzen der erzeugten Quantenschaltkreise in die Formate der SDKs verfügbarer Quantencompiler wird außerdem die Abhängigkeit eines Schaltkreises zum ursprünglich verwendeten SDK gelöst. Damit hebt sich auch die Abhängigkeit des Schaltkreises zum Compiler und den Quantencomputern auf, sodass eine Portabilität dessen zwischen den verschiedenen SDKs ermöglicht wird. Zudem ist es möglich, die Eigenschaften über Größe und Struktur der resultierenden Kompilationen direkt miteinander vergleichen zu können. Die Methode erlaubt Nutzenden außerdem, Schaltkreise auf mehreren Quantencomputern auszuführen und die Ergebnisse gegenüberzustellen sowie die Abhängigkeiten von den Eigenschaften der Kompilationen und Quantencomputer zu analysieren. Durch das Einbeziehen der Resultate vergangener Methodenaufrufe kann kontinuierlich untersucht werden, welche der Eigenschaften von Schaltkreisen, Kompilationen und Quantencomputern die Präzision der Ausführungsergebnisse beeinflussen, um diese bei der Empfehlung zu berücksichtigen.

1.2 Problemstellung und Forschungsbeiträge

In diesem Abschnitt werden die Problemstellungen und Forschungsbeiträge zur Umsetzung des vorgestellten Ziels zur Empfehlung von Quantenressourcen detailliert beschrieben. Ein Überblick der Forschungsbeiträge ist

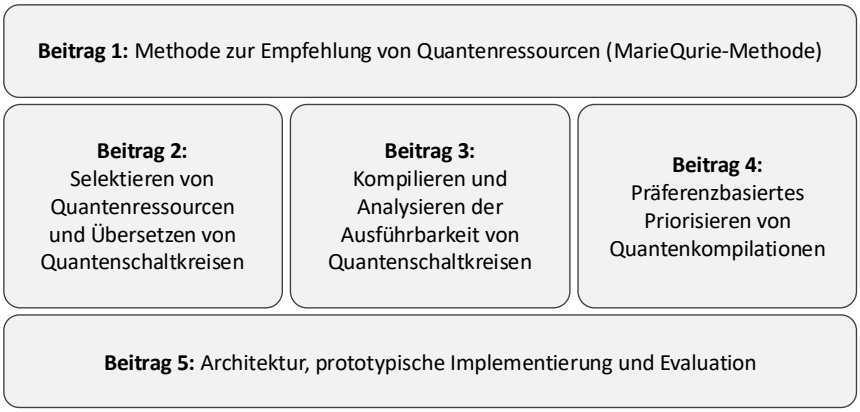

Abbildung 1.2: Übersicht der Forschungsbeiträge

in Abbildung 1.2 zu sehen: Forschungsbeitrag 1 stellt die MarieQurie-Methode zur Empfehlung von Quantenressourcen vor. Die Umsetzung und Automatisierung der Methode wird durch die darauffolgenden Beiträge 2 bis 4 adressiert. Forschungsbeitrag 2 präsentiert die Selektion geeigneter Quantenalgorithmenimplementierungen, Quantenschaltkreise, Quantencompiler als auch Quantencomputer. Außerdem stellt dieser Beitrag die Übersetzung der erzeugten Quantenschaltkreise in verschiedene Formate vor. Forschungsbeitrag 3 präsentiert unter anderem diverse Arten der Eingabedateninitialisierung für Quantencomputer, die die Größe und somit die Ausführbarkeit von Schaltkreisen bezüglich möglicher Fehlerbeeinträchtigungen beeinflussen. Daraufhin wird die Kompilierung von Schaltkreisen mit verschiedenen Quantencompilern und -computern vorgestellt und die Ausführbarkeit der Quantenkompilationen bestimmt. Forschungsbeitrag 4 zeigt die Priorisierung ausführbarer Kompilationen anhand von Anforderung der Nutzenden. Um die MarieQurie-Methode sowie die anderen Forschungsbeiträge zu realisieren, stellt Forschungsbeitrag 5 die Architektur, prototypische Implementierung und Evaluation des MarieQurie-Frameworks vor. Nachfolgend werden die Forschungsbeiträge im Detail präsentiert.

1.2.1 Methode zur Empfehlung von Quantenressourcen (MarieQurie-Methode)

Die Implementierung von Quantenalgorithmen setzt Expertise in Mathematik, Informatik und Quantencomputing voraus. Bei der Ausführung beeinflusst mitunter die Wahl des Quantencomputers und des -compilers die Präzision der Ergebnisse. Diese Wahl benötigt neben der Expertise aufwändige Analysen der vorliegenden Quantenressourcen. Eine weitere technische Hürde stellt das große SDK-Angebot zur Programmierung und Ausführung dar: Nutzende schränken sich mit der Wahl eines SDK auf eine Untermenge an verfügbaren Quantencomputern ein. Ein Wechsel des SDKs bedeutet für gewöhnlich das Adaptieren der Implementierung. Gegenwärtig existiert noch kein Vorgehen, das über die Heterogenität der gegebenen Quantentechnologien hinweg abstrahieren und die Nutzenden bei der Wahl und Anwendung von Quantenressourcen für eine gegebene Probleminstanz anhand ihrer Anforderungen unterstützen kann.

Forschungsbeitrag 1 (Methode zur Empfehlung von Quantenressourcen (MarieQurie-Methode)). In der vorliegenden Arbeit wird die MarieQurie-Methode (Recommendation of quantum resources for given quantum algorithms and problem instances) präsentiert, welche Nutzende gesamtheitlich von der Wahl der Algorithmenimplementierung bis hin zur Kompilierung und Ausführung auf einem geeigneten Quantencomputer für eine gegebene Probleminstanz unterstützt. Die Methode ist (i) technologieunabhängig von spezifischen Quantenhardwareherstellern und SDKs, (ii) unterstützt Nutzende bei der Wahl geeigneter Quantenressourcen und (iii) kann semiautomatisiert als auch vollständig automatisiert verwendet werden. Durch die automatisierte Empfehlung von Quantenressourcen wird Nutzenden der manuelle Aufwand bei der Analyse und Ausführung abgenommen. Dabei werden Benutzerfehler reduziert und die Anwendung von Quantencomputing vereinfacht.

1.2.2 Selektieren von Quantenressourcen und Übersetzen von Quantenschaltkreisen

Durch die enge Kopplung einer Quantenalgorithmusimplementierung mit dem verwendeten SDK können aufgrund des genutzten Formats und der Operationen sowie des Zugangs zu Hardwareanbietern nur bestimmte Quantencompiler und -computer für die Lösung einer Probleminstanz verwendet werden. Mit der Übersetzung des Schaltkreises wird die Verwendung verschiedener Compiler ermöglicht, sodass diverse Kompilationen zur Erzielung möglichst präziser Ausführungsergebnisse zur Auswahl stehen können. Jedoch nimmt durch die wachsende Vielfalt an Quantencompilern und -computern die Anzahl möglicher Kompilierungskombinationen zu. Dies erschwert Nutzenden Kombinationen zu wählen, die präzise Ausführungsergebnisse versprechen und auf Servicequalitäten wie Wartezeiten bis zur Ausführung achten. Zudem ist die Übersetzung und Kompilierung aller möglichen Kombinationen äußerst rechenintensiv.

Forschungsbeitrag 2 (Selektieren von Quantenressourcen und Übersetzen von Quantenschaltkreisen). Im zweiten Forschungsbeitrag werden anhand des gewählten Quantenalgorithmus und der zu lösenden Probleminstanz (i) geeignete Quantenimplementierungen, Quantenschaltkreise, Quantencompiler und Quantencomputer selektiert. Für die Quantenressourcenselektion werden vergangene Ausführungen automatisiert analysiert und geeignete Kombinationen nach den Anforderungen der Nutzenden gewählt. Zudem werden (ii) die Quantenschaltkreise in die SDK-Formate selektierter Quantencompiler übersetzt. Die automatisierte Selektion und Übersetzung nimmt für Nutzende den Aufwand ab, sich in verschiedenen Technologien einzuarbeiten und Implementierungen umzuschreiben.

1.2.3 Kompilieren und Analysieren der Ausführbarkeit von Quantenschaltkreisen

Ob Quantenschaltkreise in Hinblick auf die Präzision der Ergebnisse ausführbar sind, hängt mitunter von der Größe der zu lösenden Probleminstanz und dessen Kodierung auf dem Quantencomputer ab. Für die wiederkehrende Problematik, klassische Daten in Quantencomputer zu kodieren, wurden verschiedene Lösungsansätze entwickelt, die jedoch unterschiedlich hohe Ressourcenanforderungen stellen. Außerdem hängt die Ausführbarkeit von der Kompilierungsmethode ab, die sich zwischen existierenden Quantencompilern unterscheidet [KIMK22]. Zudem bringt die Berechnung auf den Quantencomputern Wartezeiten und Kosten mit sich, sodass die Ausführbarkeit der Quantenkompilationen auf diesen zuvor bestimmt werden sollte. Bestehende Arbeiten zeigen, dass sich die Fehler eines Quantencomputers aus diversen Faktoren zusammensetzten [BBC+17; MBB+18], wie genau ist bisher jedoch unbekannt [RK19; TQ19b]. So können nur Abschätzungen hinsichtlich der Ausführbarkeit der Kompilationen getroffen werden.

Forschungsbeitrag 3 (Kompilieren und Analysieren der Ausführbarkeit von Quantenschaltkreisen). Im dritten Forschungsbeitrag wird (i) die Ausführbarkeit von Quantenschaltkreisen untersucht. Es werden (ii) bewährte Kodierungen von klassischen Daten für Quantencomputer in Form von Mustern vorgestellt und deren Auswirkungen auf die Umsetzung von Quantenalgorithmen präsentiert. Zudem werden (iii) Quantenschaltkreise mit gewählten Quantencompilern und -computern kompiliert sowie (iv) die Ausführbarkeit der Quantenkompilationen automatisiert geprüft und selektiert. Dabei werden die Eigenschaften der vorhandenen Quantencomputer und Kompilationen automatisiert erfasst. Durch die Automatisierung reduziert sich für Nutzende der Aufwand, verschiedene Schnittstellen abzufragen und die Ausführbarkeit von Kompilationen zu untersuchen.

1.2.4 Präferenzbasiertes Priorisieren von Quantenkompilationen

Bei Vorliegen ausführbarer Quantenkompilationen können Nutzende entscheiden, welche dieser ausgeführt werden sollen. Jedoch müssen sie entsprechend eigener Prioritäten sowohl die Eigenschaften der Quantencompiler und -computer als auch die Servicequalitäten des Hardwareherstellers berücksichtigen. Forschungsbeitrag 2 selektiert bereits Quantenressourcen anhand der Anforderungen von Nutzenden, allerdings sind die Eigenschaften der Quantenschaltkreise durch den Kompilierungsvorgang verändert, sodass akkuratere Analysen durchgeführt werden können. Hierfür müssen die Quantenkompilationen auf ihre Eigenschaften hin untersucht werden und in Relation mit den Eigenschaften des jeweiligen Quantencomputers gebracht werden. Diese Untersuchung setzt umfassende Kompetenzen und Erfahrungswerte basierend auf bisherigen Ausführungen voraus. In Kombination mit der Berücksichtigung von Servicequalitäten, stehen Nutzende vor der Aufgabe, eine geeignete Entscheidung zu fällen.

Forschungsbeitrag 4 (Präferenzbasiertes Priorisieren von Quantenkompilationen). Der vierte Forschungsbeitrag stellt ein Konzept zur Priorisierung von Quantenkompilationen für Quantencomputer basierend auf Anforderungen von Nutzenden vor. Hierbei ermöglicht das Konzept (i) eigene oder (ii) vordefinierte Präferenzen wie kurze Wartezeiten und präzise Ausführungsergebnisse anzugeben. Zusätzlich kann (iii) die Stabilität der Platzierungen in der resultierenden Rangliste analysiert werden. Bei der Priorisierung für zukünftig präzise Ergebnisse werden historische Ausführungen verwendet, um den Einfluss der verschiedenen Eigenschaften von Kompilationen und Quantencomputern automatisiert ermitteln zu können. Das Konzept ermöglicht Nutzenden interaktiv die Rangliste berechnen zu lassen und die Priorisierung kritisch betrachten und anpassen zu können. Die Selektion einer Kompilation kann jedoch auch voll-automatisiert ohne Zwischeninteraktionen durch Nutzende ausgeführt werden.

1.2.5 Architektur, prototypische Implementierung und Evaluation

Forschungsbeitrag 5 umfasst die (i) Architektur, (ii) prototypische Implementierung und (iii) Evaluation des MarieQurie-Frameworks zur Validierung der Forschungsbeiträge 1 bis 4. Der Forschungsbeitrag ermöglicht die Automatisierung der MarieQurie-Methode aus Forschungsbeitrag 1 zur Empfehlung von Quantenressourcen.

> **Forschungsbeitrag 5** (Architektur, prototypische Implementierung und Evaluation). Mit dem fünften Forschungsbeitrag werden alle vorherigen Forschungsbeiträge umgesetzt. Dabei wird die Architektur des MarieQurie-Frameworks sowie dessen Prototyp vorgestellt. Das präsentierte Framework wird mit einer Evaluation getestet. Insgesamt setzt sich das MarieQurie-Framework wie folgt zusammen: (i) Einem Vorhersage- & Priorisierungsservice zur Voraussage geeigneter Quantenressourcen und zur Berechnung von Ranglisten sowie (ii) einem Service zur Übersetzung in verschiedene Formate. Des Weiteren besteht es aus (iii) verschiedenen SDK-Services zur Analyse, Kompilierung und Ausführung, (iv) dem Provenance-System QProv [WBL+21] zur Sammlung aktueller Quantencomputerdaten und (v) dem NISQ Analyzer zur Selektion von Quantenressourcen. Das Framework ist Plug-in-basiert und kann mit beliebigen Technologien, Methoden und Eigenschaften erweitert werden.

1.3 Veröffentlichungen im Rahmen der Arbeit

Im Rahmen dieser Promotion wurden die folgenden begutachteten Publikationen auf Konferenzen und Workshops veröffentlicht, welche die erbrachte Forschung dieser Arbeit darlegen.

1. [SBB+20] M. Salm, J. Barzen, U. Breitenbücher, F. Leymann, B. Weder und K. Wild. „The NISQ Analyzer: Automating the Selection of

Quantum Computers for Quantum Algorithms“. In: *Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2020)*. Springer, 2020, S. 66–85

2. [SBLW20] M. Salm, J. Barzen, F. Leymann und B. Weder. „About a Criterion of Successfully Executing a Circuit in the NISQ Era: What $wd \ll 1/\epsilon_{\text{eff}}$ Really Means“. In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020)*. ACM, 2020, S. 10–13
3. [SBL+21] M. Salm, J. Barzen, F. Leymann, B. Weder und K. Wild. „Automating the Comparison of Quantum Compilers for Quantum Circuits“. In: *Proceedings of the 15th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2021)*. Springer, 2021, S. 64–80
4. [SBLW22a] M. Salm, J. Barzen, F. Leymann und B. Weder. „Prioritization of Compiled Quantum Circuits for Different Quantum Computers“. In: *Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2022)*. IEEE, 2022, S. 1258–1265
5. [SBLW22b] M. Salm, J. Barzen, F. Leymann und P. Wundrack. „Optimizing the Prioritization of Compiled Quantum Circuits by Machine Learning Approaches“. In: *Proceedings of the 16th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2022)*. Springer, 2022, S. 161–181
6. [SBLW23] M. Salm, J. Barzen, F. Leymann und P. Wundrack. „How to Select Quantum Compilers and Quantum Computers Before Compilation“. In: *Proceedings of the 13th International Conference on Cloud Computing and Services Science (CLOSER 2023)*. SciTePress, 2023, S. 172–183

Außerdem sind die nachfolgenden begutachteten Publikationen ebenfalls Teil dieser Arbeit.

1. [WBLS20] M. Weigold, J. Barzen, F. Leymann und M. Salm. „Data

Encoding Patterns For Quantum Algorithms“. In: *Proceedings of the 27th Conference on Pattern Languages of Programs (PLoP ’20)*. HILLSIDE, 2020, S. 1–11

2. [WBLS21c] M. Weigold, J. Barzen, F. Leymann und M. Salm. „Expanding Data Encoding Patterns For Quantum Algorithms“. In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2021, S. 95–101
3. [WBLS21b] M. Weigold, J. Barzen, F. Leymann und M. Salm. „Encoding patterns for quantum algorithms“. In: *IET Quantum Communication* 2.4 (2021), S. 141–152

Darüber hinaus wurden weitere Publikationen im Rahmen der Forschungsarbeit am Institut veröffentlicht:

1. [WBL+20] B. Weder, J. Barzen, F. Leymann, M. Salm und D. Vietz. „The Quantum Software Lifecycle“. In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020)*. ACM, 2020, S. 2–9
2. [WBLS21a] B. Weder, J. Barzen, F. Leymann und M. Salm. „Automated Quantum Hardware Selection for Quantum Workflows“. In: *Electronics* 10.8 (2021), S. 1–18
3. [WBL+21] B. Weder, J. Barzen, F. Leymann, M. Salm und K. Wild. „QProv: A provenance system for quantum computing“. In: *IET Quantum Communication* 2.4 (2021), S. 171–181
4. [OBB+23] J. Obst, J. Barzen, M. Beisel, F. Leymann, M. Salm und F. Truger. „Comparing Quantum Service Offerings“. In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Bd. 02. IEEE, 2023, S. 181–184

1.4 Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen für die Analyse und Selektion von Quantenressourcen präsentiert. Kapitel 3 stellt die MarieQurie-Methode vor. Kapitel 4 zeigt zum einen die Selektion von Quantenimplementierungen, Quantenschaltkreisen, Quantencompilern und Quantencomputern für gewählte Quantenalgorithmen anhand gegebener Probleminstanzen. Zum anderen werden die erzeugten Quantenschaltkreise gewählter Implementierungen übersetzt. Die Kodierung klassischer Daten sowie die Kompilierung von Quantenschaltkreisen mit selektierten Quantencompilern und -computern wird in Kapitel 5 präsentiert. Außerdem wird in diesem Kapitel die Bestimmung der Ausführbarkeit von Quantenkompilationen auf zugehörigen Quantencomputern vorgestellt. Kapitel 6 zeigt die Priorisierung von Kompilationen basierend auf den Eigenschaften und Ergebnissen vergangener Ausführungen. In Kapitel 7 wird die Architektur und der Prototyp der MarieQurie-Methode vorgestellt. Anschließend wird in Kapitel 8 die Evaluation des Prototyps präsentiert. Kapitel 9 zeigt schließlich die Zusammenfassung sowie den Ausblick auf zukünftige Forschungsbeiträge.

Kapitel 2

Grundlagen für die Analyse und Selektion von Quantenressourcen

In diesem Kapitel werden die Grundlagen des Quantencomputings und die praktische Anwendung dessen sowie unterschiedliche Selektionsverfahren und der Einsatz von Mustern präsentiert. Dazu werden in Abschnitt 2.1 die Eigenschaften von Quantenschaltkreisen und -computern sowie die Kompilierung und Ausführung der Schaltkreise erläutert. Außerdem wird die Messung der Performanz von Quantencomputern betrachtet. Abschnitt 2.2 gibt eine Übersicht über die SDKs zur Softwareentwicklung für Quantencomputer und deren Heterogenität. In Abschnitt 2.3 werden Mechanismen zur Vorhersage und zur Priorisierung von Quantenressourcen präsentiert. Abschnitt 2.4 zeigt das Konzept von Mustern und dessen Format.

2.1 Quantencomputing

In diesem Abschnitt werden die Grundlagen des Quantencomputings erläutert und zugehörige Begriffe definiert.

2.1.1 Grundlagen des Quantencomputings

In der Quanteninformatik ist die kleinste Recheneinheit ein Qubit, welches analog zum klassischen Bit die Zustände $|0\rangle$ oder $|1\rangle$ annehmen kann [NC11]. Hierbei wird ein Quantenzustand $|\psi\rangle$ als Vektor beschrieben und mittels Dirac-Notation präsentiert [RP11]. Das Qubit kann sich als Linearkombination dargestellt in beiden Zuständen befinden, auch *Superposition* genannt, wodurch unendlich viele Zustände gleichzeitig angenommen werden können:

$$|\psi\rangle = \alpha_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \alpha_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \alpha_0 |0\rangle + \alpha_1 |1\rangle, \text{mit } \alpha_0, \alpha_1 \in \mathbb{C} \text{ und } |\alpha_0|^2 + |\alpha_1|^2 = 1 \tag{2.1}$$

$\{|0\rangle, |1\rangle\}$ bilden eine Orthonormalbasis dieses Vektorraums [NC11]. Ausgelesen wird ein Qubit, indem auf ihm eine Messoperation angewendet wird. Bei der Messung nimmt das Qubit entweder den Zustand $|0\rangle$ oder $|1\rangle$ an, und das Ergebnis wird in einem klassischen Bit zur Verfügung gestellt [Kas21]. Mit welcher Wahrscheinlichkeit $|0\rangle$ oder $|1\rangle$ in der Standardbasis $\{|0\rangle, |1\rangle\}$ gemessen wird, ist abhängig von den Koeffizienten der Linearkombination, auch *Amplituden* genannt. Befindet sich ein Qubit beispielsweise in einer gleichgewichteten Superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, ist die Wahrscheinlichkeit $|0\rangle$ oder $|1\rangle$ zu messen jeweils $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$. Mehrere Qubits bilden zusammen ein *Quantenregister*, wobei der Zustand eines Quantenregisters mit n Qubits wie folgt beschrieben werden kann [RP11]:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \,, \text{mit } \sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \tag{2.2}$$

Wobei i in Binärdarstellung ist. Sind mindestens zwei α_i nicht 0, befindet sich der Zustand in Superposition [RP11]. Ist ein Zustand $|\psi\rangle$ mit n Qubits *separabel*, lassen sich die Zustände als Tensorprodukt $|\psi\rangle = |\psi_0\rangle \otimes ... \otimes |\psi_k\rangle = |\psi_0 ... \psi_k\rangle$ mit $k < n$ darstellen [Kas21]: Ein Beispiel ist der separable Zwei-Qubit-Zustand $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) = |0\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Kann $|\psi\rangle$ nicht als Tensorprodukt dargestellt werden, ist er *verschränkt*: Wird beispielsweise eines der zwei Qubits im Zustand $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ gemessen, nimmt das gesamte Register den Zustand $|00\rangle$ oder $|11\rangle$ an.

Im Quantencomputing existieren verschiedene Rechenmodelle wie das *adiabatische* [ADK+08; FGGS00], *One-Way* [RB01] und *gatterbasierte* [NC11] Modell. In der vorliegenden Arbeit wir das gatterbasierte Rechenmodell betrachtet, welches als universelles Modell angesehen wird [GKS+22; Pre12]. Berechnungen auf Qubits werden mit Quantengattern durchgeführt [NC11]. Ein Quantengatter entspricht einer unitären Matrix und ist *reversibel* [RP11]. Angewendet auf einem einzelnen Qubit, spricht man von einem *Ein-Qubit-Gatter*, auf mehreren Qubits von einem *Multi-Qubit-Gatter*. Ein Beispiel für ein Multi-Qubit-Gatter ist das Zwei-Qubit-Gatter *CNOT*, welches die *Verschränkung* zweier Qubits ermöglicht, sodass diese miteinander korrelieren [Kas21; NC11]. Die Verschränkung wird als eine der Ursachen für den Quantenvorteil von Quantenalgorithmen im Vergleich zu klassischen Algorithmen angesehen [NC11].

2.1.2 Quantenschaltkreise

Quantenalgorithmen, wie beispielsweise die Algorithmen von Shor [Sho97] und Grover [Gro96], sind hybrid und bestehen aus klassischen und Quantenanteilen [BL22; LB20; Ley19; WBLV21]. Quantenanteile werden durch Quantenschaltkreise realisiert. Klassische Anteile beinhalten unter anderem

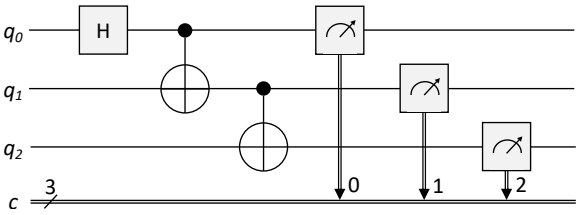

Abbildung 2.1: Darstellung eines Quantenschaltkreises

die Vorberechnungen zur Kodierung der klassischen Daten in einen Quantenzustand und die Nachbearbeitung der Ergebnisse ausgeführter Quantenschaltkreise [Ley19; WBLS20].

Definition 2.1 (Quantenschaltkreis – informell). Ein Quantenschaltkreis beschreibt die Zustandsänderung eines Quantensystems und besteht aus Qubits, Quantengattern, Messoperationen und klassischen Bits [NC11].

Ein beispielhafter Schaltkreis in Abbildung 2.1 enthält, von links nach rechts gelesen, ein Ein-Qubit-Gatter, zwei CNOTs und drei Messoperationen. Qubits, hier q_0, q_1, q_2, werden durch horizontale Linien dargestellt, auf denen die Operationen ausgeführt werden [NC11]. Klassische Bits c werden durch eine doppelte Linie vereint abgebildet. Verschiedene *Metriken* beschreiben die Größe und Struktur eines Quantenschaltkreises. Die *Breite* eines Schaltkreises definiert die Qubitzahl, die für die Umsetzung eines gegebenen Quantenalgorithmus nötig ist [LB20]. Die *Tiefe* beschreibt die Anzahl hintereinander auszuführenden Schichten an Gattern. Zusätzlich kann die *Tiefe* beispielsweise in Hinblick auf die *Multi-Qubit-Gatter* bestimmt werden [KIMK22; SDC+20]. Zudem wird die *Anzahl der Operationen* gezählt [Kas21]. Diese kann aufgeteilt werden in die *Anzahl* der *Ein-Qubit-* und *Multi-Qubit-Gatter* sowie *Messoperationen* [CMP21; KIMK22; SDC+20]. In Abbildung 2.1 hat der Schaltkreis eine Breite und Tiefe von drei und eine Multi-Qubit-Gattertiefe von zwei Schichten. Es sind insgesamt sechs Operationen, davon ein Ein-Qubit-Gatter, zwei Multi-Qubit-Gatter und drei Messoperationen enthalten.

Andere Arbeiten betrachten weitere Metriken zur Analyse von Schaltkreisen: Tomesh et al. [TGO+22] präsentieren beispielsweise Metriken zur Messung der Parallelität und des Verschränkungsverhältnisses eines Schaltkreises, um die Auslastung von Quantencomputern zu vergleichen, welche auch von Quetschlich, Burgholzer und Wille [QBW23c] verwendet werden. Verschiedene Arbeiten zählen neben der allgemeinen Breite Qubits, die ausschließlich als Hilfestellung bei Berechnungen herangezogen werden, sogenannte *Ancilla-Qubits* [CMP21; NC11; TM19]. Cruz-Lemus, Marcelo und Piattini [CMP21] stellen weitere feingranulare Schaltkreismetriken vor, wie beispielsweise die jeweiligen Vorkommnisse einzelner Gattertypen sowie deren Verhältnisse zueinander. Die Brauchbarkeit dieser Metriken wollen sie zukünftig experimentell validieren. Auch die Arbeit von Maslov und Miller [MM05] präsentiert diverse Metriken zur teils gewichteten Zählung von Gattern, sodass Multi-Qubit-Gatter mehr gezählt werden als Ein-Qubit-Gatter.

2.1.3 Quantencomputer

Die Prozessoren *(engl. „Quantum Processing Units (QPUs)“)* heutiger Quantencomputer, auch *Noisy Intermediate-Scale Quantum (NISQ)* Rechner genannt, sind störanfällig und stellen nur wenige kontrollierbare Qubits zur Verfügung, welches die Größe ausführbarer Quantenschaltkreise einschränkt [Pre18]. Um Multi-Qubit-Gatter auf Qubits ausführen zu können, müssen die Qubits miteinander verbunden sein [TQ19b]. NISQ-Rechner weisen jedoch häufig eine eingeschränkte Konnektivität zwischen den Qubits auf, da sowohl die Verbindungen als auch beieinanderliegende Qubits sich gegenseitig stören [MSSD21; RK19; TQ19b]. Durch die auftretenden Fehler ist der Zustand eines Qubits nur für eine limitierte Zeit, der *Dekohärenzzeit*, stabil [Sho95]. Dabei beschreibt die Metrik T_1, für wie lange ein Qubit sich im angeregten Zustand $|1\rangle$ befindet, bevor es in den Zustand $|0\rangle$ übergeht [TQ19b]. T_2 gibt an, wie schnell der Zustand eines Qubits durch das Umfeld verändert wird.

Um abschätzen zu können, wie viele Gatter innerhalb der Dekohärenzzeit ausgeführt werden können, bevor die Präzision einer Berechnung aufgrund

der auftretenden Fehler abnimmt, werden die *Gatterzeiten* von Ein- und Multi-Qubit-Gattern gemessen [Sho95; SZR16; Unr95]. Die Ausführung von Operationen verursacht weitere Fehler, welche durch *Ein-* und *Multi-Qubit-Gatterfehlerraten* sowie *Messfehlerraten* beschrieben werden [KLR+08; TQ19a]. Für die Gegenwahrscheinlichkeit zur Fehlerrate wird häufig der Begriff *Fidelität* verwendet [SWS15]. Multi-Qubit-Gatter weisen die höchsten Gatterfehlerraten auf [SDC+20; TQ19b]. Aufgrund der Hardwareeinschränkungen unterstützen Quantencomputer nur Untermengen an möglichen Gattertypen, jedoch können mit einer endlichen Menge dieser bereits beliebige Berechnungen ausgeführt werden [Boo12; NC11]. Verglichen zu den Gattern benötigen Messoperationen wesentlich mehr Zeit bei der Ausführung, weshalb sie die höchste Fehlerrate haben [LB20; TQ19a]. Quantencomputer werden in regelmäßigen Abständen neu kalibriert, sodass sich die Fehlerraten und die Dekohärenzzeiten ständig verändern [TQ19b].

Nebst Gattern variieren existierende Quantencomputer in der Anzahl Qubits, deren Konnektivität, den Fehlerraten sowie Dekohärenzzeiten und somit in ihrer Performanz untereinander [LBF+20; MSSD21; OBB+23; PRY+22]. Quantencomputer werden häufig über die Cloud zur Nutzung bereitgestellt [LBF+20; Sod18]. Um die Ausführungsanfragen zahlreicher Nutzenden zu verwalten, verwenden Hardwareanbieter beispielsweise Warteschlangen oder Zeitfenster, um den Zugriff aufzuteilen [LaR19; VBL+21].

2.1.4 Kompilierung von Quantenschaltkreisen

Um Quantenschaltkreise auszuführen, müssen diese mit einem Quantencompiler auf die Hardware des jeweiligen Quantencomputers abgebildet werden [HSST18; ZPW19]. Zum einen weist ein Quantencompiler die logischen Qubits aus dem auszuführenden Schaltkreis den physikalischen Qubits des Quantencomputers zu [ZPW19]. Sind durch die eingeschränkte Konnektivität (Abschnitt 2.1.3) physikalische Qubits, auf denen Multi-Qubit-Gatter angewendet werden sollen, nicht direkt miteinander verbunden, müssen die betroffenen Zustände über einen Pfad von miteinander verbundenen Qubits

mithilfe von SWAP-Gattern transferiert werden [TQ19b]. Hierfür werden zusätzliche Multi-Qubit-Gatter benötigt, welche wiederum die Fehlerraten erhöhen. Es wird angenommen, dass dieser Abbildungsvorgang NP-schwer ist, sodass basierend auf demselben Schaltkreis verschiedene Kompilationen resultieren können [CDD+19; SBB+20; SSCQ18]. Zum anderen ersetzt ein Quantencompiler die logischen Gatter, die nicht von der Hardware unterstützt werden, durch eine Unterroutine physikalischer Gatter, welches wiederum zum Anstieg der Gatterzahl im Schaltkreis führen kann [Boo12; FBW18; HC18; HSST18; LB20; ZPW19].

Neben der Kompilierung führen Quantencompiler in der Regel Optimierungen am Schaltkreis durch, um dessen Größe für präzisere Ergebnisse weitmöglichst zu minimieren [HC18; SDC+20]. Optimierungsstrategien sind die Reduktion und die Parallelisierung von Gattern, um die Breite sowie die Tiefe, und somit die benötigte Ausführungszeit des Quantenschaltkreises, zu verringern [HC18; LB20; SDC+20]. Hierbei können Optimierungen sowohl vor als auch nach der Kompilierung am Schaltkreis vorgenommen werden [AG20; HSST18; MDMN08; SDC+20].

2.1.5 Ausführung von Quantenschaltkreisen

Quantenschaltkreise werden mehrmals ausgeführt, sodass sich Wahrscheinlichkeitsverteilungen der aus Bitfolgen bestehenden Messergebnisse ergeben, welche häufig mittels Histogrammen dargestellt werden [LB20; LBF+20]. Wie häufig ein Schaltkreis auf einem Quantencomputer ausgeführt werden soll, ist durch die Anzahl *Schüsse (engl. „shots“)* oder auch *Versuche (engl. „trials“)* definierbar [LaR19]. Die Ergebnispräzision eines Quantencomputers kann mit unterschiedlichen Metriken bestimmt werden. Ein Beispiel ist *Heavy Output Generation Probability* [AC16], bei der die am häufigsten auftretenden Messergebnisse der fehlerfreien Lösung auch bei den gestörten Ausführungen am häufigsten resultieren sollten [MSSD21]. *Kreuzentropie* [BIS+18] berechnet die Distanz zwischen der Wahrscheinlichkeitsverteilung der fehlerfreien und der gestörten Schaltkreisausführung [MSSD21]. Hingegen summiert die

ℓ_1-*Norm Distanz* die Differenz zwischen den einzelnen Wahrscheinlichkeiten der vorkommenden Bitfolgen der idealen und fehlerbehafteten Ausführungen auf. Fehlerfreie Ausführungen können bis zu einer gewissen Größe durch Quantensimulatoren berechnet werden, die mit klassischen Ressourcen das Verhalten von Quantencomputern simulieren [CZX+18; LB20].

Um im Allgemeinen die Ähnlichkeit zweier Histogramme zu bestimmen, existieren unter anderem χ^2-*Distanz* [PW10], *Korrelation* [SSS17] und *Histogrammschnitt* (engl. „*Histogram Intersection (HI)*“) [SB91]. In der vorliegenden Arbeit wird HI angewendet, um die Präzision von Messresultaten zu bewerten, da sich dieser durch dessen Effizienz und einfache Umsetzung auszeichnet [SSS17]. Die Formel lautet wie folgt [SB91]:

$$H(I,M) = \frac{\sum_{j=0}^{n-1} \min(I_j, M_j)}{\sum_{j=0}^{n-1} M_j} \tag{2.3}$$

I_j und M_j stellen für die vorliegende Arbeit dar, wie oft die Bitfolge j der zwei Histogramme I und M gemessen wurde [SB91]. Wobei n der Anzahl der aufgetretenen Bitfolgen entspricht. Das jeweilige Minimum wird aufsummiert und normalisiert, sodass ein Wert zwischen 0 und 1 entsteht. 0 steht für eine leere Schnittmenge von I und M, 1 für eine vollständige Übereinstimmung. Abbildung 2.2 stellt diese Übereinanderlegung der beiden Ergebnishistogramme von Simulator und Quantencomputer zur Berechnung des HI dar. Im Allgemeinen ist es schwierig festzulegen, ab welchem Wert von einem präzisen Ergebnis beziehungsweise einer erfolgreichen Ausführung gesprochen werden kann [BY20]. Aufgrund dessen werden die HI-Werte der Ausführungsergebnisse in der vorliegenden Arbeit miteinander verglichen und entsprechende Ranglisten erstellt (Forschungsbeiträge 2 und 4).

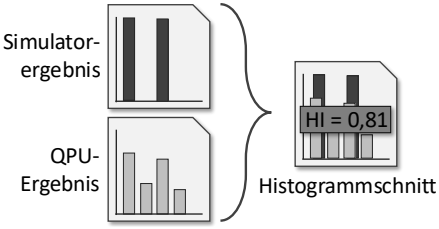

Abbildung 2.2: Messung der Präzision von Ausführungsergebnissen mittels Histogrammschnitt

2.1.6 Messung der Performanz von Quantencomputern

Um die Leistungsfähigkeit heutiger NISQ-Rechner miteinander vergleichen zu können und die Größenordnung ausführbarer Quantenschaltkreise für präzise Ergebnisse einzustufen, gibt es verschiedene Metriken [SBLW20]. Zunächst werden zwei Metriken präsentiert, die als Grundlage für die Arbeit dienen. Anschließend werden weitere Metriken diskutiert.

2.1.6.1 Total Quantum Factor

Sete, Zeng und Rigetti [SZR16] stellen die Metrik *Total Quantum Factor (TQF)* vor, welche eine grobe Abschätzung über die maximal ausführbare Schaltkreisgröße definiert:

$$\mathrm{TQF} = \frac{T_q}{t_g} \cdot n_q \tag{2.4}$$

Hierbei steht T_q für die durchschnittliche Dekohärenzzeit T_1 aller verfügbaren Qubits n_q, und t_g entspricht der längsten Gatterzeit des Quantencomputers [SZR16]. Während n_q die maximale Breite eines Quantenschaltkreises bestimmt, schätzt T_q/t_g dessen maximale Tiefe ab, bevor die Ausführung erheblich durch auftretende Fehler beeinflusst wird und korrekte Ergebnisse nicht mehr eindeutig erkennbar sind [SBL+21; SBLW20; SZR16].

TQF bestimmt eine grobe obere Schranke über die Ausführbarkeit von Schaltkreisen auf individuellen Quantencomputern, da Gatter mit langen Gatterzeiten TQF signifikant beeinflussen, auch wenn diese nicht häufig verwendet werden [SBLW20]. Zudem beschränkt sich TQF nur auf Gatter- und Dekohärenzzeiten und berücksichtigt keine Fehlerraten (Abschnitt 2.1.3).

2.1.6.2 Quantum Volume

Bishop et al. [BBC+17] und Moll et al. [MBB+18] präsentieren die Performanzmetrik *Quantum Volume (QV)*, welche ebenfalls ein Maß ist, um die Größenordnung erfolgreich ausführbarer Quantenschaltkreise anzugeben:

$$V_Q = \max_{n' \leq n} \min[n', \frac{1}{n' \epsilon_{\text{eff}}(n')}]^2 \tag{2.5}$$

Die Anzahl n der verfügbaren Qubits auf dem betrachteten Quantencomputer muss größer oder gleich n', der Breite des auszuführenden Schaltkreises, sein [MBB+18]. ϵ_{eff} ist die effektive Fehlerrate des Quantencomputers und kann für den Fall vieler Ausführungen verschiedener Quantenschaltkreise der Tiefe eins als gemittelte Zwei-Qubit-Gatterfehlerrate abgeschätzt werden [BBC+17; MBB+18]. Mit der Annäherung an ϵ_{eff} anhand von Ausführungen werden die benötigten zusätzlichen Gatter aufgrund der unvollständigen Konnektivität und implementierten Gattertypen der Hardware (Abschnitt 2.1.3) sowie die realisierbare Parallelisierung, beeinflusst durch den Compiler (Abschnitt 2.1.4), berücksichtigt [MBB+18]. Die maximal ausführbare Tiefe wird durch $d \simeq 1/n' \epsilon_{\text{eff}}(n')$ definiert. Wird von einem konstanten $\epsilon_{\text{eff}}(n')$ ausgegangen, sinkt d mit der Zunahme von n'. ϵ_{eff} steht aufgrund weiterer Fehler, wie der gegenseitigen Beeinflussung der Qubits (Abschnitt 2.1.3), in Abhängigkeit zu n'. Verglichen mit TQF (Abschnitt 2.1.6.1) fließen bei QV statt Gatter- und Dekohärenzzeiten, auftretende Fehler sowie die Effizienz der Kompilierungs- und Optimierungsmethoden des Compilers in die Abschätzung mit ein [BBC+17; MBB+18; PBE22; SBLW20].

Um QV in der Praxis zu bestimmen, wird für gewöhnlich die folgende Formel angewandt [CBS+19; JJB+21; PDF+21]:

$$V_Q = 2^{\min(d,n')} \tag{2.6}$$

V_Q wird bestimmt, indem mit der Tiefe d und Breite n' gleich große Zufallsschaltkreise ($d = n'$) generiert und diese inkrementell vergrößert werden, solange mehr als Zweidrittel der Ergebnisse korrekt berechnet wurden [CBS+19; LJV+21]. Das d beziehungsweise n' der zuletzt erfolgreich ausgeführten Schaltkreisgröße wird in die Gleichung (2.6) eingesetzt.

Allerdings kritisieren Blume-Kohout und Young [BY20], dass die Schaltkreise von Quantenalgorithmen selten quadratischen Formen entsprechen und Tiefe als auch Breite unabhängig voneinander betrachtet werden sollten. Stattdessen sollten zusätzliche Formen wie beispielsweise periodisch und algorithmische Schaltkreise betrachtet werden, um unterschiedliche Fehlerverhalten zu berücksichtigen [BMB+22; BY20; LJV+21; RK19]. Des Weiteren ist nicht explizit geklärt, ob die Werte von d und n' sich auf die Maße der logischen oder der physikalischen Schaltkreise beziehen [MBC+22; SBLW20]. McKay et al. [MHP+23] geben zudem an, dass QV nur auf die performantesten Qubits angewendet wird und dadurch nicht die durchschnittliche Leistung eines Quantencomputers mit einer großen Anzahl an Qubits gemessen wird. Baldwin et al. [BMB+22] und Pelofske, Bärtschi und Eidenbenz [PBE22] zeigen, dass QV eine geeignete Metrik für den Vergleich aktueller NISQ-Rechner ist, jedoch nicht alleinstehend zur Beschreibung der Performanz herangezogen werden sollte. Miller et al. [MBC+22] erweitern den Ansatz des QV und klassifizieren gegebene Quantenalgorithmen entsprechend ihres Größenwachstums in Abhängigkeit zur Qubitzahl. Für jede Klasse ist eine eigene QV-Formel definiert, sodass das QV, wie in Gleichung (2.6) beschrieben, nicht exponentiell ansteigt, wenn auch nur schrittweise größere Tiefen und Breiten ausführbar sind.

2.1.6.3 Weitere Performanzmetriken und -benchmarks

Neben TQF und QV existieren noch weitere Metriken um die Leistung existierender Quantencomputer miteinander vergleichen zu können. Wack et al. [WPJ+21] zeigen zum Beispiel auf, dass die Performanz von Quantencomputern nicht nur durch die erfolgreich ausführbare Schaltkreisgröße mittels QV (Gleichung (2.5)) eingestuft werden, sondern zusätzlich der Durchsatz bei der Ausführung berücksichtigt werden sollte. Hierfür definieren sie die Metrik *Circuit Layer Operations per Second (CLOPS)*, welche die Anzahl an Schaltkreisen zählt, die in einem festen Zeitrahmen von einem Quantencomputer ausgeführt werden können. CLOPS berücksichtigt sowohl Hardwareeigenschaften wie beispielsweise Gatterzeiten als auch Softwareaspekte wie die Steuerung des Quantencomputers und die Antwortzeiten zur Übermittlung der Ergebnisse an die Nutzenden. Die Metrik $\bar{\gamma}$ [IBM22; PSW22; TBG17] stuft das Fehlermodell eines Quantencomputers sowie den nötigen Aufwand zur Fehlerminderung ein. Das Fehlermodell wird durch das Ausführen mehrerer Schaltkreise approximiert [IBM22]. Verringert wird $\bar{\gamma}$ durch Hardwareoptimierungen, wie beispielsweise verbesserter Gatterfehlerraten und Dekohärenzzeiten. McKay et al. [MHP+23] präsentieren *Layer Fidelity (LF)*. LF kombiniert verschiedene Arten des Quantencomputerbenchmarkings und bemisst skalierbar die Fidelität eines Quantencomputers und mittels *Error per Layered Gate (EPLG)*, die Fehler der Gatter in Schaltkreisen.

Metriken, um die Zuverlässigkeit von Quantencomputern zu evaluieren, sind unter anderem *Probability of Successful Trial (PST)* [LMR+17; TQ18; TQ19a; TQ19b], welches die Wahrscheinlichkeit einer fehlerfreien Schaltkreisausführung definiert, und *Mean Instructions Before Failure (MIBF)* [TQ18], das die Anzahl ausführbaren Operationen bis zum Auftreten von Fehlern beschreibt. *Inference Strength (IST)* [TQ19a] bemisst das Verhältnis von richtigen und falschen Ausführungsergebnissen eines Schaltkreises. Alle drei Metriken werden durch Messergebnisse ausgeführter Schaltkreise für den jeweiligen Quantencomputer bestimmt [TQ18; TQ19b]. Eine Metrik, die nicht auf Schaltkreis-, jedoch auf Anwendungsebene ansetzt, ist *Q-score* [MAA21]. Diese spiegelt die Performanz von Quantencomputern basierend auf dem

Optimierungsproblem MaxCut wider, welches durch den jeweiligen Quantencomputer gelöst und entsprechend schrittweise vergrößert wird.

Für die Performanzmessung von Quantencomputern auf Operationsebene, gibt es Benchmarkingmethoden, wie *Randomized Benchmarking* [KLR+08; MGE12], *Cycle Benchmarking* [EWP+19] und *Tomography* [CN97]. Darauf wird in dieser Arbeit nicht näher eingegangen. Neben den unterstützten Hardwareanbieter, die die berücksichtigten Operationseigenschaften (Abschnitt 4.3.2) bereitstellen, könnte die MarieQurie-Methode mittels Plug-ins um verschiedene Benchmarkingmethoden erweitert werden.

Die in diesem Abschnitt vorgestellte Metriken und Benchmarks ermöglichen zwar den Vergleich der Performanz heutiger NISQ-Rechner, jedoch geben sie keine Einschätzung darüber, ob ein beliebiger Quantenschaltkreis erfolgreich auf einem bestimmten Quantencomputer ausführbar ist, ohne Messergebnisse zuvor ausgeführter Schaltkreise vorliegen zu haben.

2.1.7 Zusammenfassung

Durch die besonderen physikalischen Eigenschaften der Superposition und Verschränkung kann Quantencomputing für bestimmte Problemstellungen einen Effizienzvorteil gegenüber klassischem Computing aufweisen. Aktuelle Quantencomputer sind jedoch durch hohe Fehlerraten und niedrige Qubitzahlen eingeschränkt, wodurch die Größe ausführbarer Quantenschaltkreise beeinträchtigt ist. Um den Umfang und die Struktur von Schaltkreisen zu untersuchen, gibt es eine Vielzahl an Metriken, ebenso für die Hardwareressourcen und Fehlerraten der Quantencomputer. Vor der Ausführung eines Schaltkreises muss dieser für den jeweiligen Quantencomputer kompiliert werden, wodurch sich der Schaltkreis in seiner Größe und Struktur verändern kann. Die Kompilierung ist NP-schwer, weshalb Kompilationen desselben Schaltkreises sich unterscheiden können. Ausgeführt wird eine Kompilation schließlich mehrmals, um eine Wahrscheinlichkeitsverteilung der fehlerbehafteten Messergebnisse zu erhalten. Um die Performanz von Quantencomputern miteinander vergleichen zu können, wurden verschiedene

Metriken entwickelt. Darunter auch erste Ansätze, um die Größe erfolgreich ausführbarer Quantenschaltkreise für einen bestimmten Quantencomputer ermitteln zu können. Diese betrachten jedoch entweder nur wenige Hardwareeigenschaften oder beschränken sich beim Benchmarking nur auf eine bestimmte Art von Schaltkreisen.

Im Rahmen der vorliegenden Arbeit werden die vorgestellten Schaltkreis- und Hardwaremetriken verwendet, um (i) die Präzision zukünftiger Ausführungsergebnisse vorherzusagen und geeignete Quantenressourcen vor der Kompilierung zu bestimmen (Forschungsbeitrag 2). Des Weiteren wird (ii) ein erster Ansatz zur Bestimmung der oberen Schranke erfolgreich ausführbarer Schaltkreisgrößen für Quantencomputer vorgestellt und basierend auf der Leistungsmetrik TQF [SZR16] (iii) die Ausführbarkeit von Kompilationen auf den Quantencomputern geprüft (Forschungsbeitrag 3). Zudem werden die Metriken genutzt, um (iv) die Kompilationen anhand nutzerbasierten Anforderungen priorisieren zu können (Forschungsbeitrag 4).

2.2 Softwareentwicklung für Quantencomputer

In diesem Unterkapitel werden SDKs zur Softwareentwicklung für Quantencomputer und deren Heterogenität dargestellt.

2.2.1 Software Development Kits für Quantencomputing

SDKs für Quantencomputing werden häufig zur Implementierung, Kompilierung und Ausführung von Quantenalgorithmen auf Quantencomputern herangezogen [LaR19]. Für die Implementierung von Quantenalgorithmen stellen SDKs hardwarenahe oder höhere Programmiersprachen als Formate zur Verfügung [BLF+21; SCPP22]. Hardwarenahe Sprachen ähneln klassischem Assembler und beschreiben meist Sequenzen einzelner Quantenoperationen [LaR19; VBLW21]. Daraus resultieren in solch einem Format Quantenschaltkreise für spezifische Probleminstanzen. Höhere Programmiersprachen wie Python erleichtern die Lesbarkeit und unterstützen auch nicht-

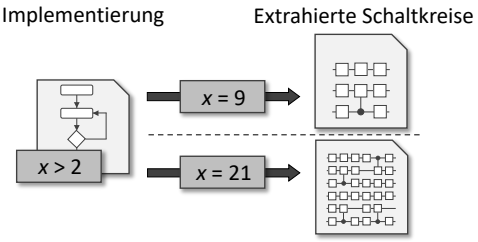

Abbildung 2.3: Generierung und Extrahierung von Quantenschaltkreisen durch eine Quantenimplementierung mit unterschiedlichen Probleminstanzen

lineare Programmstrukturen [LaR19; SCPP22; VBLW21]. So kann mit einer höheren Sprache nicht nur ein konkreter Schaltkreis definiert, sondern basierend auf Eingabewerten, Schaltkreise für variable Probleminstanzen mithilfe von Implementierungen generiert werden [BLF+21; WBL+20; WBLV22]. Programme einer Implementierung definieren die Operationen, die zur Initialisierung des Quantenzustands mit den zu kodierenden klassischen Daten der gegebenen Probleminstanz benötigt werden [Ley19; WBLS20]. Auf den initialisierten Quantenzustand werden entsprechend die Operationen der Quantenberechnung definiert. Somit ist, wie Abbildung 2.3 zeigt, die Schaltkreisgröße von der gegebenen Probleminstanz abhängig [SBB+20]. Implementierungen können zudem Programme zur Nachbereitung der Messergebnisse enthalten [BBG+23; BL22; LB20; WBLV22].

Definition 2.2 (Quantenimplementierungen – informell). Quantenimplementierungen beschreiben Quantenalgorithmen und generieren Quantenschaltkreise anhand übergebener Probleminstanzen. Die Quantenimplementierungen enthalten die Vor- und Nachverarbeitung der Quantenschaltkreise. Die Größe von Quantenschaltkreisen ist abhängig von den Probleminstanzen.

Die erzeugten Quantenschaltkreise können schließlich für einen ausgewählten Quantencomputer mit dem zumeist von dem SDK mitgelieferten Quantencompiler kompiliert und daraufhin ausgeführt werden [FBW18].

Definition 2.3 (Quantenressourcen – informell). Quantenressourcen beschreiben in dieser Arbeit, anders als beispielsweise in Arbeiten von Ravi et al. [RSMC21] und Suchara et al. [SKF+13], im Allgemeinen zusammengefasst Quantenimplementierungen, Quantenschaltkreise, Quantenkompilationen sowie Quantencompiler und Quantencomputer.

2.2.2 Heterogenität der heutigen Quantensoftwarelandschaft

Viele Hardwareanbieter, wie beispielsweise Google mit *Cirq* [Cir23], Rigetti mit *Forest* [Rig21] und IBM mit *Qiskit* [Qis23], stellen ein proprietäres SDK für die Entwicklung und Ausführung von Implementierungen mit ihren Quantencomputern bereit [BLF+21; LaR19]. Diese unterstützen neben allgemein verbreiteten Gattern auch die spezifischen Gattertypen der eigenen Quantencomputer, die von Hersteller zu Hersteller unterschiedlich sind. Tabelle 2.1 stellt folgend einen Ausschnitt existierender proprietärer und Tabelle 2.2 einen Ausschnitt herstelleragnostischer SDKs dar. Arbeiten von Fingerhuth, Babej und Wittek [FBW18] und Serrano et al. [SCPP22] präsentieren ausführliche Studien.

SDK	**Eingangsformat**	**Ausgangsformat nach Kompilierung**	**Cloud- & Hardwareanbieter**	**Modifizierbare Kompilierung**
Cirq [Cir23]	Cirq*, JSON, OpenQASM, Quil, Quirk	Cirq*, JSON, OpenQASM, Quil, Quirk	AQT, Azure Quantum, Google, IonQ, Pasqal, Rigetti	✓
Forest [SCZ16]	PyQuil*, Quil	Quil	Rigetti	✗

Tabelle 2.1: Übersicht proprietärer SDKs und ihren Eigenschaften. Aufbauend auf [SBL+21]. Pythonbibliotheken sind mit * markiert. (Fortsetzung auf der nächsten Seite)

SDK	Eingangs-format	Ausgangs-format nach Kompilierung	Cloud- & Hardware-anbieter	Modifizierbare Kompilierung
Qiskit [Qis23]	OpenQASM, Qiskit*	OpenQASM, Qiskit*	Amazon Braket, AQT, Azure Quantum, IBM Quantum, IonQ, IQM, Quantinuum, Rigetti	✓

Tabelle 2.1: Übersicht proprietärer SDKs und ihren Eigenschaften. Aufbauend auf [SBL+21]. Pythonbibliotheken sind mit * markiert. (Fortsetzung)

Wie in Tabelle 2.1 gezeigt, akzeptieren Forest und Qiskit Implementierungen in ihren proprietären Formaten, wobei es sich bei PyQuil [Kar+20] und Qiskit um Python Bibliotheken handelt und bei Quil [SCZ16] und OpenQASM [CBSG17] um hardwarenahe Sprachen. Cirq unterstützt neben der eigenen Pythonbibliothek Cirq noch weitere Formate für die Kompilierung und Ausführung von Implementierungen. Während über Cirq und Qiskit auf verschiedene Anbieter zugegriffen werden kann, erlaubt Forest nur den Zugriff auf die Rigetti-Hardware. Bei Forest kann keine benutzerdefinierte Zielgattermenge für die Kompilierung bestimmt werden (letzte Spalte von Tabelle 2.1), sodass nur auf die Gattertypen der Rigetti Quantencomputer kompiliert werden kann.

SDK	Eingangs-format	Ausgangs-format nach Kompilierung	Cloud- & Hardware-anbieter	Modifizierbare Kompilierung
Amazon Braket [Ama23]	Blackbird, Braket*, JAQCD, OpenQASM, Qiskit*, Strawberry Fields*, PennyLane	Blackbird, JAQCD, OpenQASM, Quil, Strawberry Fields*	IonQ, OQC, QuEra, Rigetti, Xanadu	✓
pytket [SDC+20]	Braket*, Cirq*, OpenQASM, PennnyLane, PyQuil*, PyZX*, Qiskit*, Quipper, qujax*	Braket*, Cirq*, IonQ-JSON, OpenQASM, Qiskit*, ProjectQ*, PyZX*, PyQuil*, qujax*, Qulacs*, Q#	AQT, Amazon Braket, Azure Quantum, IonQ, IQM, Quantinuum, Rigetti, IBM Quantum	✓
staq [AG20]	OpenQASM	Cirq*, OpenQASM, ProjectQ*, Quil, Q#	✗	✗

Tabelle 2.2: Übersicht herstelleragnostischer SDKs und ihren Eigenschaften. Aufbauend auf [SBL+21]. Pythonbibliotheken sind mit * markiert. (Fortsetzung auf der nächsten Seite)

SDK	Eingangsformat	Ausgangsformat nach Kompilierung	Cloud- & Hardwareanbieter	Modifizierbare Kompilierung
ProjectQ [SHT18]	ProjectQ*	ProjectQ*	Amazon Braket, AQT, Azure Quantum, IBM Quantum, IonQ	✓

Tabelle 2.2: Übersicht herstelleragnostischer SDKs und ihren Eigenschaften. Aufbauend auf [SBL+21]. Pythonbibliotheken sind mit * markiert. (Fortsetzung)

Tabelle 2.2 zeigt, dass Amazon Braket [Ama23] sowie pytket [SDC+20] viele Formate unterstützen, während ProjectQ [SHT18] nur die eigene Pythonbibliothek anbietet. Staq [AG20] ermöglicht die Übersetzung von OpenQASM in andere Formate, bietet jedoch keine Anbieteranbindung, sondern erlaubt nur die Modellierung von Quantencomputern inklusive Qubitkonnektivitäten und Fehlerraten mit Gattern der IBM Quantum (IBMQ) Quantencomputer.

Grundsätzlich erlauben alle der in Tabelle 2.1 und Tabelle 2.2 betrachteten SDKs die Modellierung benutzerdefinierter Quantencomputer, auf deren Eigenschaften schließlich kompiliert und die Ausführung simuliert werden kann. Allerdings ist die Definition von Dekohärenzzeiten und Fehlern häufig eingeschränkt und die Formate, mit denen die Hardwareeigenschaften beschrieben werden, unterscheiden sich zwischen den SDKs [SBL+21]. In Bezug auf die in den SDKs mitgelieferten Compiler zeigen Arbeiten, dass sich die Größe und Struktur von Kompilationen derselben Schaltkreise aufgrund verschiedener Abbildungs- und Optimierungsmethoden teils signifikant unterscheiden [KIMK22; MSSD21; SDC+20].

2.2.3 Zusammenfassung

Die aktuelle Softwarelandschaft für Quantencomputer ist geprägt durch heterogene SDKs [Sod18]. Sie unterscheiden sich in unterstützten Formaten, sodass eine Implementierung nicht zwangsweise auf Quantencomputern beliebiger Anbieter ausgeführt werden kann, ohne diese übersetzen zu müssen [BBB+23c]. Quantencomputer können eingeschränkt nachmodelliert werden, jedoch ist das Transferieren des Quantencomputermodells zu anderen SDKs aufgrund verschiedener Formate häufig mühsam und die Daten sind aufgrund von Rekalibrierungen (Abschnitt 2.1.3) regelmäßig veraltet. Allerdings ist die Betrachtung verschiedener Quantencomputer aufgrund unterschiedlicher Hardwareeigenschaften fundamental für eine erfolgreiche Ausführung eines Schaltkreises. Auch die Kompilierung hat großen Einfluss auf die Ausführbarkeit, weshalb mehrere Compiler betrachtet werden sollten.

In der vorliegenden Arbeit werden verschiedene SDKs angebunden, damit (i) mehrere Compiler für die Kompilierung von Quantenschaltkreisen betrachtet werden, um geeignete Kompilationen empfehlen zu können (Forschungsbeitrag 2). Hierfür werden (ii) die aus Implementierungen erzeugten Quantenschaltkreise in die Formate der jeweiligen SDKs übersetzt. Zudem werden (iii) durch die angebundenen SDKs verschiedene Quantencomputer für die Kompilierung und Ausführung unterstützt, um für Implementierungen geeignete Hardware anzubieten. Des Weiteren können Nutzende (iv) zur Beeinflussung der Methode bestimmen, ob ihr Fokus beispielsweise auf kurzen Wartezeiten oder präzisen Ergebnissen liegt.

2.3 Mechanismen zur Vorhersage und Priorisierung

Im Folgenden werden zunächst mehrere Methoden zur Vorhersage von Ergebnissen erläutert. Anschließend werden verschiedene Priorisierungsmethoden zur Erstellung von Ranglisten präsentiert.

2.3.1 Methoden zur Vorhersage von Ergebnissen

Für die Vorhersage kontinuierlicher Werte als *Zielgrößen* mit quantitativen Eingabedaten, sogenannten *unabhängigen Variablen* [Mei20] anhand eines gegebenen Datensatzes, werden im *Maschinellen Lernen (ML)* Regressionsalgorithmen angewendet [JWHT21; SSP16]. Enthält der zum Trainieren verfügbare Datensatz Zielgrößen als auch unabhängigen Variablen, wird von überwachtem Lernen gesprochen [JWHT21]. Bei Bestehen nominaler kategorialer unabhängiger Variablen müssen diese zuvor beispielsweise mittels *One-Hot-Kodierung* in quantitative unabhängige Variablen überführt werden [CVK18; SBLW23]: Jede Kategorie wird durch einen Merkmalsvektor beschrieben, dessen Dimensionen den insgesamt n Kategorien entspricht. Die i-te Dimension des Vektors der i-ten Kategorie ist 1, die restlichen Dimensionen sind 0, wobei $0 \leq i \leq n$ gilt.

2.3.1.1 Multiple Lineare Regression

Bei der *multiplen linearen Regression* werden zunächst die Regressionskoeffizienten der unabhängigen Variablen basierend auf den Trainingsdaten abgeschätzt [JWHT21]. Dabei definieren die Koeffizienten den Zusammenhang zwischen den unabhängigen Variablen und den Zielgrößen. Die Koeffizienten werden schließlich auf die unabhängigen Variablen der Testdaten angewendet, um deren bisher unbekannte Zielgrößen vorhersagen zu können.

2.3.1.2 Support Vector Regression

Support Vector Regression (SVR) für Regressionsprobleme stammt von der bekannten Support Vector Machine [Vap95] ab. Für das Lernen schätzt SVR eine Funktion, die schließlich die Zielgrößen der Testdaten vorhersagen soll [AK15]. Dabei werden Trainingsdatenpunkte, die über- beziehungsweise unterhalb der zu bestimmenden Funktion liegen, bestraft. Datenpunkte auf der Funktion werden nicht bestraft. Um die Sensibilität des Vorgehens zu mildern, wird ein kleinstmöglicher Schlauch um die Funktion gelegt, der so

viele Trainingsdatenpunkte enthalten soll wie möglich. Hiermit werden nur Datenpunkte bestraft, die außerhalb des Schlauchs liegen.

2.3.1.3 K-nächste-Nachbarn Regression

K-nächste-Nachbarn Regression (KNNR) ähnelt KNN für Klassifizierungsprobleme [JWHT21]. Um die Zielgrößen von Testdatenpunkten zu bestimmen, werden für jeden Testdatenpunkt die K nächstgelegenen Trainingsdatenpunkte betrachtet. Hierbei wird die Zahl K im Vorhinein definiert. Die Zielgröße des Testdatenpunkts wird schließlich mit der durchschnittlichen Zielgröße dieser K betrachteten Trainingsdatenpunkte berechnet.

2.3.1.4 Entscheidungsbäume

Entscheidungsbäume entstehen durch das wiederholte aufteilen der unabhängigen Variablen des Trainingsdatensatzes in zwei sich nicht überschneidende Bereiche, die sich durch die Mittelwerte aus den enthaltenen Zielwerten beschreiben lassen [DF00; JWHT21]. Unabhängige Variablen aus dem Testdatensatz werden von der Wurzel beginnend nacheinander mit den Grenzwerten der Bereiche verglichen und entsprechend diesen zugeordnet. Bei der letzten Ebene des Entscheidungsbaums wird der vorherzusagenden Zielgröße der Mittelwert des Bereichs zugewiesen.

2.3.2 Methoden zur Berechnung von Ranglisten

Um bei einer endlichen Menge an *Alternativen* Nutzenden bei der Wahl einer geeigneten Alternative unterstützen zu können, werden häufig *multikriterielle Entscheidungsanalysemethoden (MCDA-Methoden)* (engl. „*Multi-Criteria Decision Analyses methods*“) eingesetzt [SDH+14; WJZ+19]. Die Methoden finden Anwendung in Bereichen wie Cloud Computing [FJJB18], Nachhaltigkeit [BACA14] und Finanzen [AB17]. Hierbei werden für die Wahl relevante Eigenschaften der Alternativen mit *Kriterien* beschrieben [WJZ+19]. MCDA-Methoden ermöglichen es mehrere, sich möglicherweise widersprechende

Ziele anstreben zu können [GL14]. Präferenzen der Nutzenden können durch die *Gewichtung* der einzelnen Kriterien definiert werden.

Aufgrund der Vielzahl diverser MCDA-Methoden wurde für die vorliegende Arbeit das *MCDA Method Selection Tool* [WJZ+19; WJZ+24] für die Wahl geeigneter MCDA-Methoden verwendet. Demnach sollen die für die MarieQurie-Methode geeigneten MCDA-Methoden vollständige oder partielle Ranglisten statt einzelnen Alternativen oder Klassifizierungen zurückgeben [WJZ+19; WJZ+24]. Des Weiteren sollen die MCDA-Methoden durch die Angabe von quantitativen Gewichten Nutzendenpräferenzen miteinbeziehen können. Auch die zu betrachtenden Kriterien, welche den Metriken in Abschnitt 4.3 entsprechen, sind quantitativer und nicht qualitativer oder relativer Natur. Anhand dieser Eigenschaften empfiehlt das *MCDA Method Selection Tool* 30 entsprechende MCDA-Methoden. Das MarieQurie-Framework unterstützt davon initial die zwei MCDA-Methoden *Technique for Order Preference by Similarity to Ideal Solution (TOPSIS)* [HY81] und *Preference Ranking Organization METHod for Enrichment of Evaluations II (PROMETHEE II)* [BM05], wobei weitere Methoden Plug-in-basiert eingebunden werden können [GL14; Sał20; SBLW22a]. Beide MCDA-Methoden berechnen konkrete Punktzahlen für Rangplatzierungen, die für das präzise, automatisierte Bestimmen von Gewichten notwendig sind (Abschnitt 2.3.3) [SBLW22b].

2.3.2.1 TOPSIS

Bei der MCDA-Methode TOPSIS wird für jede Alternative der euklidische Abstand zum Optimum, bestehend aus den jeweils besten Kriterienwerten aller gegebenen Alternativen, und zum Pessimum aus den entsprechend schlechtesten Werten berechnet [BACA14]. Dabei werden die gegebenen Gewichte der Kriterien mit den jeweiligen Abständen multipliziert. Für den Vergleich der Kriterien werden die Werte und Gewichte normalisiert und aus den Abständen zu Optimum und Pessimum wird für jede Alternative der relative Abstand zum Optimum errechnet. Die Rangliste resultiert aus den nach relativen Abständen absteigend sortierten Alternativen.

2.3.2.2 PROMETHEE II

Mit PROMETHEE wird jede Alternative mit jeweils jeder anderen anhand der festgelegten Kriterien verglichen [CGS13]. Um bestimmen zu können, welche der beiden betrachteten Alternativen der anderen überlegen ist, werden pro Kriterium Grenzwerte spezifiziert und zunächst die Abstände der Alternativwerte zu diesen Grenzwerten berechnet [GL14]. Anschließend werden die Abstände mit den entsprechenden Gewichten multipliziert, aufaddiert sowie normiert. Der *Ausgangsfluss* bestimmt, wie viele Prozentpunkte eine Alternative allen anderen überlegen ist [CGS13; GL14]. Der *Eingangsfluss* bestimmt, um wie viele Prozentpunkte die anderen Alternativen gegenüber der betrachteten überlegen sind. Mit dem für PROMETHEE II spezifischen *Nettofluss* wird schließlich die Rangliste der Alternativen errechnet, in dem der Eingangsfluss vom Ausgangsfluss abgezogen wird [GL14].

2.3.3 Methoden zur Bestimmung von Gewichten

Ausschlaggebend für die Berechnung aussagekräftiger Ranglisten ist die geeignete Wahl der Kriteriengewichte, um die angestrebten Ziele widerspiegeln zu können [Ols04; SBLW22b]. Die *Simple Multi-Attribute Rating Technique (SMART)* [Edw77] ist ein Verfahren, um Nutzende bei der Wahl quantitativer Gewichte zu unterstützen [GL14; SBLW22a]: Jedem Kriterium wird eine Punktezahl zwischen 0 und 100 zugewiesen. Das wichtigste Kriterium erhält 100 Punkte, die anderen entsprechend weniger. Kriterien mit 0 Punkten werden ausgeschlossen. Das Gewicht eines Kriteriums ergibt sich aus der zugewiesenen Punktzahl geteilt durch die Summer aller vergebenen Punkte.

Dennoch ist das Bestimmen quantitativer Gewichte für Nutzende meist schwer umsetzbar und führt zu Ungenauigkeiten [WJZZ09]. Eine Möglichkeit, Gewichte zu bestimmen, sodass ein gesetztes Ziel erreicht werden kann, ist, deren Einfluss anhand gegebener Datensätze mittels ML zu extrahieren [LvM19; OAS18; Ols04; SBLW22b]. Initial unterstützt das erweiterbare MarieQurie-Framework die drei Optimierungsalgorithmen *Genetischer Algorithmus* [Hol73], *Evolutionäre Strategie* [Fog94] sowie *Constrained Opti-*

mization BY Linear Approximations (COBYLA) [Pow94] zur automatisierten Wahl geeigneter Gewichte [SBLW22b]. Als Voraussetzung benötigen die drei Algorithmen von den MCDA-Methoden ausschließlich die berechnete Rangpunktzahl pro Alternative [Bös06; CJK+16; HH18; SBLW22b].

2.3.3.1 Genetischer Algorithmus

Der für seine Robustheit bekannte genetische Algorithmus ist ein stochastischer Ansatz, der den Evolutionsprozess nachstellen soll [CJK+16; Fog94; HH18]. Als initiale *Population* zufällige Eltern zu wählen, deren Eignung anschließend bewertet wird, ist eine mögliche Variante des genetischen Algorithmus [HH18; Pat21]. Durch das *Kreuzen* der tauglichsten Eltern und dem Mitwirken von *Mutationen* werden Kinder erzeugt, welche eine neue *Generation* darstellen. Dieser Prozess wird wiederholt, bis eine geeignete Generation besteht, das heißt, das Optimum erreicht wurde oder ein Abbruchkriterium eingetreten ist. Im Vergleich zu anderen Vorgehen, wie beispielsweise gradientenbasierten Algorithmen, ist der genetische Algorithmus resistenter gegenüber lokalen Minima und erfordert weniger Voraussetzungen [CJK+16; Hau95]. Dafür läuft der genetische Algorithmus jedoch Gefahr, keine optimalen Ergebnisse zu liefern und ineffizient zu sein.

2.3.3.2 Evolutionsstrategie

Wie der genetische Algorithmus ist auch die Evolutionsstrategie ein evolutionärer Optimierungsalgorithmus [CJK+16; Fog94]. Die erste Population besteht aus zufällig generierten Eltern, zu deren Eigenschaften Gaußsches Rauschen hinzugefügt wird, um Nachkommen zu erzeugen [Fog94]. Anschließend wird geprüft, ob Elternteil oder Kind geeigneter ist und entsprechend in die nächste Population aufgenommen. Dies wird wiederholt, bis eine ausreichende Lösung gefunden wurde. Im Vergleich zum genetischen Algorithmus ist die heuristische Evolutionsstrategie oft schneller, bleibt jedoch mit erhöhter Wahrscheinlichkeit in lokalen Minima hängen [CJK+16].

2.3.3.3 COBYLA

Der deterministische, gradientenfreie Optimierungsalgorithmus COBYLA ist ursprünglich entwickelt worden, um ein lokales Optimum zu finden, kann jedoch wesentlich effizienter sein als der genetische Algorithmus [Bös06; Pow94; PSPP21]. COBYLA schätzt mittels Interpolation auf Funktionspunkten linearpolynomiell die Zielfunktion, deren Gradienten anschließend bestimmt werden [Bös06; Pow07]. Die zu optimierende Zielfunktion wird durch die Berechnung eines neuen Funktionspunkts ausgewertet. Das Vorgehen wiederholt sich, bis eine ausreichende Lösung ermittelt ist.

2.3.4 Sensitivitätsanalyse

Um die Stabilität von Ranglisten zu bewerten, werden Sensitivitätsanalysen durchgeführt [GL14; LQWC13; SBLW22b]. Sie ermöglichen es, den Einfluss der einzelnen Kriterien zu analysieren und Rangordnungswechsel bei Änderung der Eingabeparameter zu erkennen. Eine Art der Sensitivitätsanalyse ist die wiederholte Ausführung der zuvor verwendeten MCDA-Methode mit schrittweise veränderten Gewichten [LQWC13]. Die MCDA-Methoden TOPSIS [HY81] und PROMETHEE II [BM05] (Abschnitt 2.3.2) basieren auf normalisierten Gewichten [BACA14; GL14]. Eine Methode zur Anpassung der Gewichte von Li et al. [LQWC13] lautet wie folgt:

$$\omega_k' = \frac{\gamma_k \omega_k}{\omega_0 + \omega_1 + ... + \gamma_k \omega_k + ... + \omega_{n-1}} = \frac{\gamma_k \omega_k}{1 + (\gamma_k - 1)\omega_k} \tag{2.7}$$

$$\omega_j' = \frac{\omega_j}{\omega_0 + \omega_1 + ... + \gamma_k \omega_k + ... + \omega_{n-1}} = \frac{\omega_j}{1 + (\gamma_k - 1)\omega_k}$$
$$\text{mit } j = 0, ..., k-1, k+1, ..., n-1 \tag{2.8}$$

Hierbei ist ω_k in Gleichung (2.7) das um den Faktor γ_k anzupassende Gewicht eines der n Kriterien mit $0 \leq k \leq n-1$ [LQWC13]. ω_k' ist das durch

die Summer aller Gewichte normalisierte, neue Gewicht für Kriterium k. Entsprechend sind ω'_j in Gleichung (2.8) die für die Normalisierung neu berechneten Gewichte aller anderen Kriterien. Die MCDA-Methode wird auf den neuen Gewichten ausgeführt und das Resultat kann mit der ursprünglichen Rangliste verglichen werden [GL14]. Die Methode kann mit variierenden γ wiederholt werden.

2.3.5 Zusammenfassung

Regressionsalgorithmen werden beim maschinellen Lernen angewendet, um mit unabhängigen Variablen kontinuierliche Zielgrößen anhand verfügbarer Datensätze vorhersagen zu können. Zur Berechnung einer Rangliste aus mehreren Alternativen mit unterschiedlichen Kriterien und Zielsetzungen werden häufig MCDA-Methoden eingesetzt. Dabei ist die Bestimmung der Kriteriengewichte ausschlaggebend für den Erhalt einer geeigneten Rangliste. Die Gewichte können beispielsweise von Nutzenden selbstständig bestimmt werden oder mittels Optimierungsalgorithmen, basierend auf vorhandenen Datensätzen, erlernt werden. Resultierende Ranglisten lassen sich bezüglich ihrer Sensitivität untersuchen, sodass Nutzende Ordnungsänderungen bei Veränderung der Gewichte analysieren können.

Die vorliegende Arbeit verwendet die präsentierten Regressionsalgorithmen zur (i) Vorhersage geeigneter Kombinationen aus Quantenschaltkreisen, Quantencompilern und Quantencomputern für gewählte Quantenalgorithmen, basierend auf historischen Ausführungen (Forschungsbeitrag 2). Die vorgestellten MCDA-Methoden in Kombination mit den Optimierungsalgorithmen ermöglichen (ii) die Priorisierung von Kompilationen und (iii) die manuelle als auch die automatisierte Bestimmung von Kriteriengewichten (Forschungsbeitrag 4). Des Weiteren erlaubt die MarieQurie-Methode (iv) die automatisierte Sensitivitätsanalyse der berechneten Ranglisten.

2.4 Muster

Nachfolgend werden die Eigenschaften von Mustern und das in dieser Arbeit verwendete Musterformat präsentiert.

2.4.1 Muster und Mustersprachen

Muster bereiten bewährte Lösungen für wiederkehrende Probleme strukturiert in abstrahierter Form auf, sodass diese interdisziplinär verständlich für Menschen lesbar sind [AIS77; WBLS21b; WBLS21c]. Muster stehen zudem in Verbindung zueinander und bilden eine sogenannte *Mustersprache* [AIS77]. Im Bereich des Quantencomputings existiert eine Mustersprache, die beispielsweise Muster zu Quantenalgorithmen, Unterroutinen, Fehlerkorrektur und Quantensoftwareentwicklung umfasst und durch die Arbeit von Leymann [Ley19] initiiert wurde.

2.4.2 Musterformat dieser Arbeit

Die Struktur eines Musters ist durch ein Format festgelegt, wobei verschiedene Formate existieren [Cop96]. Als Format für die in dieser Arbeit vorgestellten Muster wird das von Fehling et al. [FLR+14], inklusive des Abschnitts *Kräfte* (engl. „*Forces*“), verwendet [WBLS20; WBLS21b; WBLS21c]: Ein Muster zeichnet sich durch einen *Namen* aus. Neben einem *Mustersymbol* (engl. „*Icon*“) ist der *Zweck* (engl. „*Intent*“) des Musters in Form einer kurzen Zusammenfassung beschrieben. Mit dem *Kontext* (engl. „*Context*“) wird die Problemsituation mit möglichen Vorbedingungen geschildert, zu denen das jeweilige Muster als Lösung eingesetzt werden soll. *Kräfte* beschreiben Abwägungen, die beim Einsatz des Musters in Betracht gezogen werden sollten. Die *Lösung* (engl. „*Solution*“) zeigt den abstrahierten Lösungsweg auf, welcher in einer *Lösungsskizze* (engl. „*Solution Sketch*“) dargestellt ist. Daraus resultiert das *Ergebnis* (engl. „*Result*“). Verbindungen zu anderen Mustern werden im Abschnitt *Verwandte Muster* (engl. „*Related Patterns*“) aufgezeigt. Wird das Muster in leicht abgeänderter Form angewendet, wird dies im

optionalen Abschnitt *Varianten* (engl. „*Variations*“) präsentiert. *Anwendungsbeispiele* (engl. „*Known Uses*“) beschreiben Einsatzgebiete wie beispielsweise spezifische Quantenalgorithmen oder Implementierungen, die von den Mustern Gebrauch machen.

Kapitel 3

Methode zur Empfehlung von Quantenressourcen

In diesem Kapitel wird die MarieQurie-Methode zur Empfehlung von Quantenressourcen vorgestellt. Die Methode präsentiert das automatisierte Selektieren geeigneter Quantenressourcen für einen gegebenen Quantenalgorithmus und eine entsprechende Probleminstanz. Zudem ermöglicht sie das Übersetzten von Quantenschaltkreisen in unterschiedliche Formate, deren Kompilierung mit diversen Quantencompilern und die Ausführbarkeitsanalyse der Kompilationen. Ausführbare Kompilationen werden anhand von Anforderungen priorisiert und ausgeführt. Das Kapitel beschreibt Forschungsbeitrag 1, wobei zunächst die Anforderungen an die Methode definiert werden (Abschnitt 3.1). Anschließend wird ein Überblick über die einzelnen Bestandteile und Varianten der Methode gegeben (Abschnitt 3.2). Zuletzt wird die Umsetzung der Anforderungen besprochen (Abschnitt 3.3).

3.1 Anforderungen an das Empfehlen von Quantenressourcen

Basierend auf den in Kapitel 2 präsentierten Grundlagen und identifizierten Herausforderungen werden im Folgenden Anforderungen an die MarieQurie-Methode zur Empfehlung von Quantenressourcen gestellt.

Anforderung 1 (Zugänglichkeit). Aufgrund der Komplexität des Quantencomputings (Abschnitt 2.1.1), der fehleranfälligen NISQ-Rechner (Abschnitt 2.1.3) und des heterogenen Softwareangebots (Abschnitt 2.2.2), muss die Methode über diese Herausforderungen hinweg abstrahieren und auch Personen ohne entsprechende Expertise geeignete Quantenressourcen für gegebene Quantenalgorithmen und Probleminstanzen empfehlen.

Anforderung 2 (Effizienz). Um den Nutzenden einen Mehrwert bei der Verwendung der MarieQurie-Methode zu bieten, muss das Vorgehen den manuellen Aufwand für die Selektion geeigneter Quantenressourcen (Abschnitt 2.2.3) reduzieren.

Anforderung 3 (Technologieunabhängigkeit). Um die Methode flexibel erweitern zu können, muss die Unabhängigkeit von individuellen Technologien aufgrund der Heterogenität der aktuellen Soft- und Hardwarelandschaft im Bereich des Quantencomputings (Abschnitt 2.2.2) sichergestellt werden.

Anforderung 4 (Einflussnahme auf die Selektion). Die Methode muss die individuellen Anforderungen der Nutzenden (Abschnitt 2.3.3) bei der Selektion von Quantenressourcen berücksichtigen, um ihnen geeignete Empfehlungen (Abschnitte 2.3.1 und 2.3.2) anbieten zu können.

Anforderung 5 (Nachvollziehbarkeit). Die Methode muss transparent sein, um die resultierenden Ergebnisse und Empfehlungen nachvollziehen zu können (Abschnitt 2.3).

Anforderung 6 (Automatisierbarkeit). Um die MarieQurie-Methode effektiv für die Empfehlung von Quantenressourcen anwenden zu können, muss diese automatisiert ablaufen.

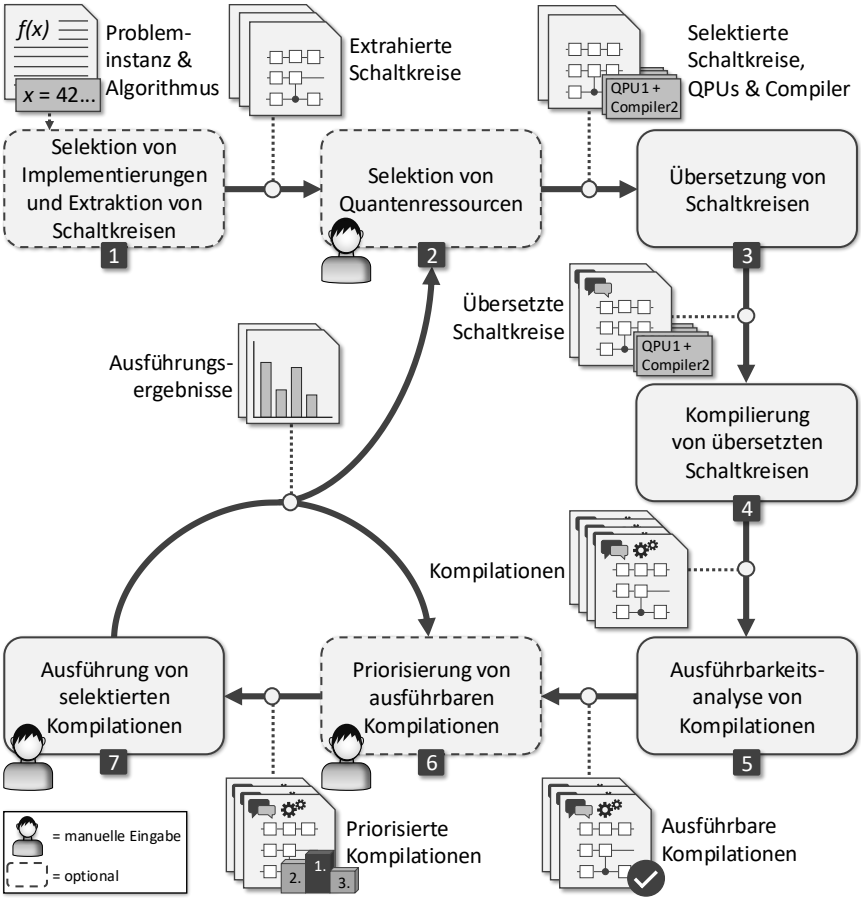

Abbildung 3.1: Schritte und Artefakte der MarieQurie-Methode

3.2 Die MarieQurie-Methode

Die MarieQurie-Methode definiert ein Verfahren zur Empfehlung von Quantenimplementierungen, Quantenschaltkreisen, Quantencompilern und Quantencomputern basierend auf der zu lösenden Probleminstanz und dem gewählten Quantenalgorithmus. Die in den sieben Methodenschritten enthal-

tenen Konzepte wurden im Rahmen dieser Arbeit entwickelt. Ein Überblick der MarieQurie-Methode ist in Abbildung 3.1 dargestellt.

Zu Beginn übergeben Nutzende die zu lösende Probleminstanz und wählen den anzuwendenden Quantenalgorithmus (Schritt 1). Anhand des Quantenalgorithmus selektiert die MarieQurie-Methode verfügbare Implementierungen und extrahiert die zur Probleminstanz korrespondierenden Quantenschaltkreise. Basierend auf den Anforderungen der Nutzenden können Vorhersagen anhand historischer Daten getroffen werden, sodass potenzielle Quantenschaltkreise, Quantencompiler und Quantencomputer gewählt werden, die die gewünschten Ergebnisse liefern sollen (Schritt 2). In Schritt 3 werden die Schaltkreise in die Formate der SDKs übersetzt, deren Compiler die Kompilierung (Schritt 4) auf den in Schritt 2 selektierten Quantencomputern unterstützen. Anschließend wird die Ausführbarkeit der Kompilationen hinsichtlich der Gatterzeiten und Dekohärenzzeiten zugehöriger Quantencomputer bestimmt (Schritt 5). Die ausführbaren Kompilationen können anhand der Anforderungen der Nutzenden sowie vergangener Ausführungen priorisiert (Schritt 6) und zur Ausführung (Schritt 7) selektiert werden. Die Ausführungsergebnisse und Metriken der Schaltkreise, Quantencomputer und Kompilationen werden sowohl für die Quantenressourcenselektion (Schritt 2) als auch für die Priorisierung (Schritt 6) wiederverwendet.

Weitere Varianten der MarieQurie-Methode sind der Einstieg der Nutzenden bei Schritt 2 oder Schritt 3. Die beiden ersten Schritte sind optional, sodass Nutzende zum einen mit einer Quantenimplementierung oder einem konkreten Schaltkreis zur Hand mit der Selektion geeigneter Quantencompiler und -computer (Schritt 2) einsteigen können. Zum anderen können sie mit der Übersetzung des Schaltkreises (Schritt 3) unter Berücksichtigung aller möglichen Quantencomputer- und Compilerkombinationen beginnen. Zudem kann die Priorisierung der Kompilationen (Schritt 6) ausgelassen und empfohlene Kompilationen direkt ausgeführt werden (Schritt 7).

Des Weiteren können Teilkonzepte von Forschungsbeitrag 2 der MarieQurie-Methode losgelöst verwendet werden: Schritt 1 zur Selektion einer Implementierung und Schritt 2 zur Selektion geeigneter Quantencompiler

und -computer für einen gegebenen Schaltkreis sowie Schritt 3 zur Übersetzung eines Schaltkreises können unabhängig voneinander ausgeführt werden. Ebenso kann Schritt 4 zur Kompilierung eines Schaltkreises mit kompatiblen Quantencompilern und -computern losgelöst von der MarieQurie-Methode verwendet werden. Im Folgenden werden die einzelnen Schritte detailliert beschrieben.

3.2.1 Schritt 1: Selektion von Implementierungen und Extraktion von Schaltkreisen

Im ersten Schritt übergeben Nutzende eine Probleminstanz, deren Lösung mittels Quantencomputing berechnet werden soll und selektieren aus einer Auswahl von unterstützten Quantenalgorithmen den darauf anzuwendenden Algorithmus. Ziel des Schritts ist die Selektion geeigneter Implementierungen zur Lösung der gegebenen Probleminstanz. Wie in Kapitel 2 beschrieben, handelt es sich bei Quantencomputing um ein andersartiges Konzept, bei dem die Entwicklung von Quantenalgorithmen und deren Implementierungen für Nutzende eine komplexe Hürde darstellen kann. Zusätzlich besteht eine große Auswahl an SDKs, die sich im Angebot von Programmiersprachen sowie Hardwareanbietern unterscheiden und dadurch zu einer SDK- und Anbieterbindung führen kann (Abschnitt 2.2.2). Der erste Schritt soll Nutzenden den Aufwand und das dafür vorausgesetzte Wissen abnehmen, selbst Implementierungen schreiben zu müssen und stattdessen automatisiert auf bestehende Implementierungen zurückzugreifen.

Für jeden gegebenen Quantenalgorithmus sind ein bis mehrere Implementierungen dessen verfügbar, die durch Entwickelnde implementiert und gespeichert werden. Die Probleminstanz wird den gewählten Implementierungen als Eingabe übergeben, sodass die entsprechenden Schaltkreise erzeugt und an den nächsten Schritt weitergegeben werden können. Die konzeptionellen Details sind in Kapitel 4 dargestellt.

3.2.2 Schritt 2: Selektion von Quantenressourcen

Anhand der im vorherigen Schritt erzeugten Quantenschaltkreise können nun optional Vorhersagen darüber getroffen werden, welche Kombinationen an Quantenschaltkreisen, Quantencompilern und Quantencomputern Ergebnisse liefern könnten, die den Anforderungen der Nutzenden entsprechen. Alternativ werden alle Kombinationen für die Kompilierung betrachtet. Die Kompilierung beeinflusst die Größe eines Schaltkreises und mit den unterschiedlichen Compilern sowie der NP-Schwere des Abbildungsvorgangs auf der Hardware variieren die resultierenden Kompilationen für einen Schaltkreis (Abschnitt 2.2.2). Die Betrachtung mehrerer Compiler ist daher maßgebend, um die Kompilation zu wählen, die zu möglichst präzisen Ausführungsergebnissen auf heutigen NISQ-Rechnern führen (Abschnitt 2.1.3).

Jedoch ist die Kompilierung sowie die durch die SDK-Bindung gegebenenfalls notwendige Übersetzung in Schritt 3 rechen- und zeitintensiv und nimmt mit zunehmender Anzahl an SDKs und Quantencomputern exponentiell zu. Daher wird in Schritt 2 anhand vorheriger Ausführungsergebnisse (Schritt 7), zugehörigen Quantencomputermetriken und initial erzeugten Schaltkreisen (Abschnitte 2.1.2 und 2.1.3) mit *Maschinellem Lernen (ML)* berechnet, welche Kombinationen mit den aktuell vorliegenden Quantenschaltkreisen vielversprechend sind und welche im weiteren Verlauf nicht berücksichtigt werden sollen. Hierfür können Nutzende wählen, ob das anzustrebende Ziel kurze Wartezeiten bis zur Ausführung, präzise Messergebnisse oder ein Verhältnis beider zueinander sein soll. Resultierend gibt Schritt 2 geeignete Kombinationen aus Schaltkreisen, Compilern und Quantencomputern den nächsten Schritt weiter. Die Details sind in Kapitel 4 beschrieben.

3.2.3 Schritt 3: Übersetzung von Schaltkreisen

Anhand der selektierten Kombinationen an Quantenressourcen aus dem vorherigen Schritt werden in diesem Schritt die gewählten Schaltkreise bei Bedarf in die Formate der SDKs zugehöriger Compiler übersetzt.

Wie in Abschnitt 2.2.2 dargestellt, unterscheiden sich die SDKs in den unterstützten Formaten, sodass bestehende Implementierungen häufig nur mit Transformationsaufwand mit anderen SDKs gehandhabt werden können. Daher wird in Schritt 3 verglichen, in welchen Formaten die Schaltkreise vorliegen und welche Formate von den SDKs der selektierten Compiler angeboten werden. Wird das Format eines Schaltkreises nicht unterstützt, wird dieser in die benötigten Formate übersetzt und an den nächsten Schritt weitergegeben (Kapitel 4). Hiermit wird das Kompilieren und Ausführen gegebener Schaltkreise mit diversen SDKs und Hardwareanbietern ermöglicht.

3.2.4 Schritt 4: Kompilierung von übersetzten Schaltkreisen

In Schritt 4 werden die übergebenen Schaltkreise mit den in Schritt 2 selektierten Quantencompilern für die gewählten Quantencomputer kompiliert. Die Kompilierung ist notwendig, um Schaltkreise auf die Hardware der Zielquantencomputer für die Ausführung abzubilden (Abschnitt 2.1.4). Dabei ersetzten die angebundenen Compiler nicht-unterstützte Gatter, bilden logische auf physikalische Qubits ab und optimieren die Schaltkreise, um deren Größe und Struktur weitestgehend zu optimieren. Die erzeugten Kompilationen werden an den nächsten Schritt übermittelt. Die Details des Konzepts sind in Kapitel 5 detailliert dargestellt.

3.2.5 Schritt 5: Ausführbarkeitsanalyse von Kompilationen

In diesem Schritt werden die Kompilationen zunächst analysiert. Identifizierte Schaltkreismetrikwerte sowie aktuelle Quantencomputermetrikwerte werden verwendet, um abzuschätzen, ob die Kompilationen erfolgreich auf den Zielquantencomputern ausgeführt werden können, sodass präzise Ergebnisse erzielt werden können.

Existierenden Compiler erkennen in der Regel, wenn Schaltkreise eine größere Breite als die Quantencomputer verfügbare Qubits verfügen. Weitergehend ist zu beachten, dass die Qubits der Quantencomputer nur für eine

kurze Zeit stabil sind (Abschnitt 2.1.3), sodass nur eine begrenzte Menge an Gattern innerhalb dieses Zeitfensters angewendet werden kann. Daher wird in diesem Schritt die maximal ausführbare Tiefe der betrachteten Quantencomputer abgeschätzt und mit der tatsächlichen Tiefe der Kompilationen abgeglichen. Kompilationen, bei denen die Annahme ist, dass sie innerhalb des jeweiligen Zeitfensters ausführbar sind, werden an den folgenden Schritt übergeben. So wird forciert, dass den Nutzenden der MarieQurie-Methode nur erfolgreich ausführbare Kompilationen empfohlen werden. Eine detaillierte Beschreibung ist in Kapitel 5 zu finden.

3.2.6 Schritt 6: Priorisierung von ausführbaren Kompilationen

Die ausführbaren Kompilationen werden den Nutzenden mit ihren Metrikwerten, den verwendeten Compilern und den aktuellen Metrikwerten der Zielquantencomputer präsentiert. Um den Nutzenden die Selektion einer oder mehrerer Kompilationen für die darauffolgende Ausführung zu erleichtern, werden in Schritt 6 die Kompilationen anhand definierter Präferenzen priorisiert. Initial wird anhand der Präferenzen von Schritt 2 priorisiert. Nutzende können jedoch optional erneut zwischen kurzen Wartezeiten, präzisen Ergebnissen, einem Verhältnis zwischen beiden oder individuellen Präferenzen wählen. Für eine Rangliste geordnet nach erwarteter Präzision dienen die historischen Daten aus Schritt 7 als Fundament, auf dem ein Optimierungsalgorithmus (Abschnitt 2.3.3) die Wichtigkeit der Metriken von Kompilationen und Quantencomputern mittels Gewichten bestimmt. Hierbei ermöglicht die Analyse von vergangenen und aktuellen Kompilationen genauere Präzisionsabschätzungen, da verglichen zu Schritt 2, die kompilierten Schaltkreisgrößen vorliegen. Mit den Gewichten wird anschließend eine MCDA-Methode (Abschnitt 2.3.2) zur Priorisierung angewendet. Die errechnete Rangliste wird den Nutzenden angezeigt, woraufhin eine Sensitivitätsanalyse (Abschnitt 2.3.4) optional durchgeführt werden kann. Die Details des Konzeptes sind in Kapitel 6 dargestellt.

3.2.7 Schritt 7: Ausführung von selektierten Kompilationen

In diesem Schritt können Nutzende wählen, welche der empfohlenen Kompilationen auf den zugehörigen Quantencomputern ausgeführt werden sollen. Die MarieQurie-Methode vereinfacht die individuelle Handhabung der heterogenen SDKs (Abschnitt 2.2.2), indem sie die Kompilationen automatisiert mit den SDKs der Compiler an die Hardwareanbieter sendet und auf den Erhalt der Ausführungsergebnisse wartet. Zugleich werden die Schaltkreise, falls klassisch simulierbar, auch auf einem Simulator ausgeführt, um fehlerfreie Ergebnisse zu erhalten (Abschnitt 2.1.5). Diese werden zur Präzisionsbestimmung mit den Ergebnissen der Quantencomputer verglichen.

Die Ergebnisse, deren durch Simulation berechnete Präzision sowie die Metrikwerte der Quantencomputer, Kompilationen und initialen Schaltkreise werden persistiert und dienen in Schritt 2 und 6 als Basis für die Selektion und Priorisierung. Die Messergebnisse und deren Präzision werden den Nutzenden angezeigt. Weitere Kompilationen können ausgeführt werden. Die Details sind in Kapitel 6 gegeben.

3.3 Erfüllung der Anforderungen

In den Schritten 1 und 2 (Kapitel 4) werden Quantenressourcen für die Ausführung eines Quantenalgorithmus zur Lösung einer gegebenen Probleminstanz selektiert, sodass auch Nutzende ohne die benötigte Expertise in der Programmierung von Quantenschaltkreisen und tiefgreifendem Wissen über Quantencomputer Quantencomputing einsetzen können (Anforderung 1). Des Weiteren reduziert die Selektion anhand der Präferenzen Nutzender den rechnerischen Aufwand, sodass nur ausgewählte Quantenressourcen für die Kompilierung eingesetzt werden (Anforderungen 2 und 4). Hierfür werden unter anderem Methoden des maschinellen Lernens auf die Eigenschaften und Ergebnisse vorheriger Ausführungen angewandt. Die Schritte 3 und 4 (Kapitel 4) stellen die Übersetzung von Quantenschaltkreisen in die benötigten Formate ausgewählter Compiler sowie die Kompilierung

mit diesen dar. Die Übersetzung lässt die bestehende Heterogenität der SDKs und Hardwareanbieter überwinden und gewährt die Ausführung von Schaltkreisen auf Quantencomputern unterschiedlicher Hersteller (Anforderung 3). Mit Schritt 5 (Kapitel 5) werden die erzeugten Kompilationen analysiert und selektiert, sodass den Nutzenden ausschließlich ausführbare Kompilationen empfohlen werden (Anforderung 1). Schritt 6 (Kapitel 6) beschreibt das Konzept, welches empfohlene Kompilationen anhand der Präferenzen von Nutzenden priorisiert und gewählte schließlich ausführt (Anforderung 4). Hierbei werden unter anderem Optimierer verwendet, welche die Informationen aus vorherigen Ausführungen betrachten. Erstellte Ranglisten können zudem zur Nachvollziehbarkeit auf ihre Stabilität hin geprüft und erneut berechnet werden (Anforderung 5). Die Schritte 3 bis 5 der MarieQurie-Methode werden voll-automatisiert ohne Eingabe von Nutzenden ausgeführt (Anforderung 6). In Schritt 1 wählen Nutzende den auszuführenden Quantenalgorithmus und übergeben die Probleminstanz, woraufhin Quantenimplementierungen automatisiert selektiert werden. In den Schritten 2 und 6 definieren Nutzende ihre Anforderungen für eine individuelle Empfehlung von Quantenressourcen. Des Weiteren können sie in Schritt 6 die automatisierte Sensitivitätsanalysen auslösen und deren Untersuchungsfaktoren definieren. Für Schritt 7 (Kapitel 6) wählen die Nutzenden schließlich, welche der empfohlenen Kompilationen automatisiert ausgeführt werden sollen.

Kapitel 4

Selektieren von Quantenressourcen und Übersetzen von Quantenschaltkreisen

Die Programmierung von Quantenalgorithmen bedarf Expertise und die Quantenschaltkreise sind nicht zwangsläufig mit jedem Quantencompiler auf Quantencomputern beliebiger Hersteller kompilier- und ausführbar. Dieses Kapitel stellt die (i) Selektion geeigneter Quantenimplementierungen (Abschnitt 4.2), Quantenschaltkreise, Quantencompiler sowie Quantencomputer für einen Quantenalgorithmus und eine Probleminstanz vor (Abschnitt 4.3). Zudem wird (ii) die Übersetzung der Schaltkreise gezeigt (Abschnitt 4.4). Das Kapitel setzt die Schritte 1 bis 3 der MarieQurie-Methode um und beschreibt Forschungsbeitrag 2. Die Konzepte wurden bereits im Rahmen dieser Arbeit publiziert [SBB+20; SBL+21; SBLW23].

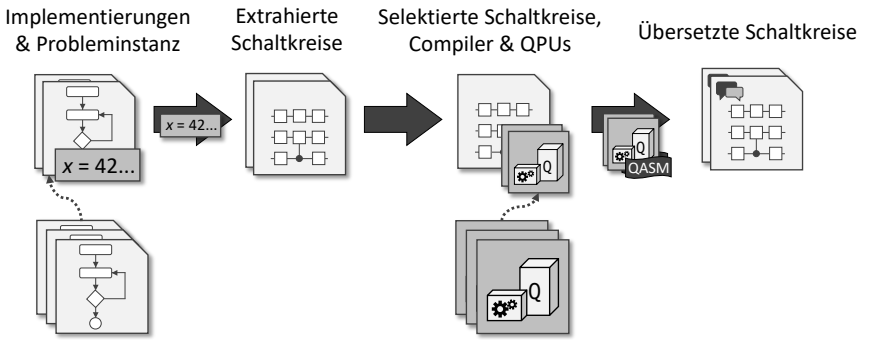

Abbildung 4.1: Vorgehen zur Selektion von Quantenressourcen und Übersetzung von Quantenschaltkreisen

4.1 Idee und Grundkonzept

Die Idee dieses Forschungsbeitrags ist, anhand des von Nutzenden gewählten Quantenalgorithmus und der Probleminstanz geeignete Implementierungen und daraus generierte Quantenschaltkreise sowie Quantencompiler und Quantencomputer automatisiert zu selektieren. Anschließend werden die Schaltkreise in die SDK-Formate der selektierten Compiler übersetzt. Den Nutzenden wird die Programmierung eigener Lösungen beziehungsweise die manuelle Wahl bestehender Implementierungen abgenommen. Außerdem ermöglicht die Vorauswahl geeigneter Quantenschaltkreise, Quantencompiler und Quantencomputer einen Effizienzgewinn für Nutzende. Die Übersetzung überwindet die Heterogenität der SDKs und ermöglicht die Kompilierung mit diversen Quantencompilern und -computern, um präzise Ausführungsergebnisse anstreben zu können. Abbildung 4.1 zeigt das Vorgehen: Für den gewählten Quantenalgorithmus werden gegebene Implementierungen selektiert. Diesen wird die Probleminstanz übergeben, sodass Quantenschaltkreise generiert und extrahiert werden können. Diese dienen mitunter zur Vorhersage, welche Quantenressourcenkombinationen die gewünschten Ergebnisse der Nutzenden erzeugen könnten. Selektierte Schaltkreise werden schließlich in die Zielformate, beispielsweise OpenQASM, übersetzt.

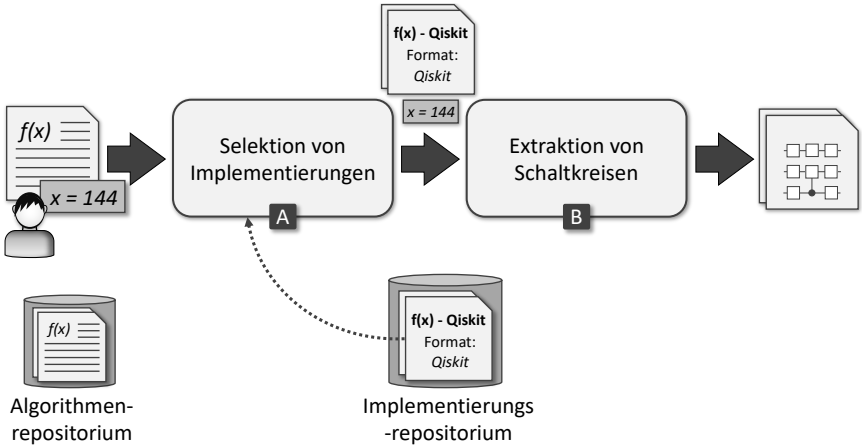

Abbildung 4.2: Selektion von Quantenimplementierungen und Extraktion der Quantenschaltkreise (adaptiert von [SBB+20])

4.2 Selektion von Quantenimplementierungen und Extrahieren von -schaltkreisen

Um Nutzenden Quantenressourcen zur Lösung ihrer Probleminstanz empfehlen zu können, wählen diese zuallererst mithilfe gegebener Algorithmenbeschreibungen [BLF+21; LBF19] einen für die Berechnung geeigneten Quantenalgorithmus und übergeben die zu lösende Probleminstanz. Ein skizzierter Ablauf ist in Abbildung 4.2 zu sehen. Ein mit der MarieQurie-Methode bereitgestelltes Repositorium enthält auswählbare Quantenalgorithmen, für welche eine oder mehrere Implementierungen vorhanden sind. Die Algorithmenbeschreibungen können beispielsweise mit Mustern ergänzt werden, um sie für die Nutzenden zugänglich darzustellen (Abschnitt 2.4) [Ley19; WBLV21]. Die Algorithmenbeschreibungen geben zudem wieder, welche Eingabeparameter zur Berechnung benötigt werden und welche Eigenschaften die zu lösenden Probleminstanzen für den jeweiligen Quantenalgorithmus erfüllen müssen. Ein Beispiel könnte sein, dass die Probleminstanz einer ungeraden natürlichen Zahl entsprechen muss. In Abbildung 4.2 selektiert

die nutzende Person den Quantenalgorithmus $f(x)$ mit der Probleminstanz $x = 144$ als Eingabe an Schritt A der MarieQurie-Methode. Im Folgend wird auf die Schritte A und B eingegangen.

4.2.1 Selektion von Quantenimplementierungen

In Schritt A werden automatisiert alle verfügbaren Implementierungen des gewählten Quantenalgorithmus selektiert. Die Implementierungen sind in einem Repositorium gespeichert. Ihnen ist die Zusatzinformation beigefügt, welchem Format sie entsprechen. Die zu betrachtende Implementierung *f(x) - Qiskit* des Algorithmus $f(x)$ in Abbildung 4.2 wurde beispielsweise mit der Pythonbibliothek *Qiskit* entwickelt. Die selektierten Implementierungen werden an Schritt B übergeben.

4.2.2 Extraktion von Quantenschaltkreisen

In Schritt B von Abbildung 4.2 wird den selektierten Implementierungen die Probleminstanz übergeben und die Implementierungen werden klassisch ausgeführt. Die Implementierungen verarbeiten zunächst die Probleminstanz und generieren jeweils einen entsprechenden Quantenschaltkreis (Abschnitt 2.2.1). Die zu untersuchenden Quantenschaltkreise werden automatisiert an den nächsten Schritt der MarieQurie-Methode übergeben. Sollen in einer Implementierung die später folgenden Ausführungsergebnisse eines Quantencomputers weiterverarbeitet werden, wird die Ausführung der Implementierung selbst nach Schritt 7 der MarieQurie-Methode, siehe Abbildung 3.1, mit Erhalt der Ausführungsergebnisse fortgesetzt.

4.3 Selektion von Quantenressourcen

Die in Abschnitt 4.2 selektierten und extrahierten Quantenschaltkreise müssen für eine Ausführung zunächst per Kompilierung auf die Hardware des jeweiligen Quantencomputers abgebildet werden (Abschnitt 2.1.4). Die

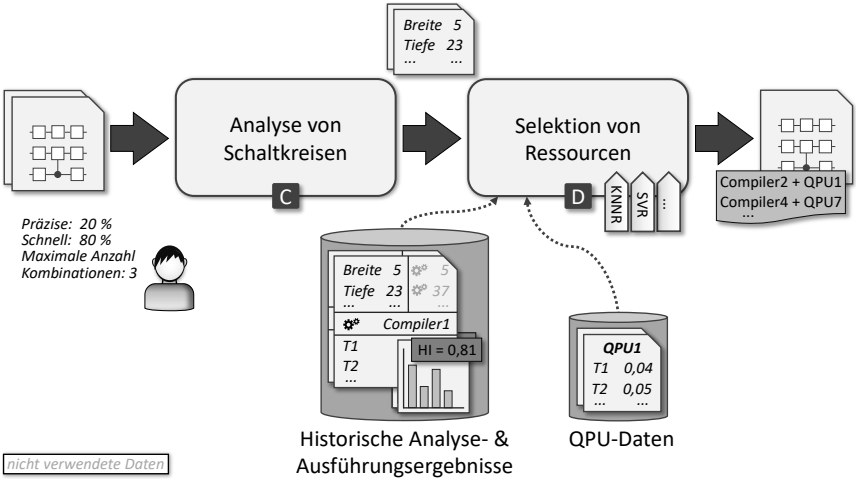

Abbildung 4.3: Analyse gegebener Quantenschaltkreise zur Selektion geeigneter Quantenressourcen (adaptiert von [SBLW23])

zunehmende Vielzahl unterschiedlicher Quantencompiler und -computer machen es Nutzenden schwer, für sie geeignete Kombinationen aus Schaltkreisen, Compilern und Quantencomputern zu wählen [KIMK22; LBF+20; MSSD21; OBB+23]. Soll eine Ausführung zusätzlich schnell durchgeführt werden, müssen Nutzende neben den technischen Aspekten die aktuellen Wartezeiten bis zur Ausführung auf den einzelnen Quantencomputern berücksichtigen [VBL+21]. Damit Nutzende sicherstellen können, die effizientesten Kompilationen für möglichst präzise Ausführungsergebnisse zu erlangen, müssen sie mit exponentiellem Aufwand alle verfügbaren Kombinationen aus Schaltkreisen und Quantencomputern mit den Compilern kompilieren und untersuchen. Hierfür automatisiert die MarieQurie-Methode, wie in Abbildung 4.3 dargestellt, die Analyse gegebener Schaltkreise und die Wahl geeigneter Kombinationen anhand der Anforderungen der Nutzenden.

Nutzende können angeben, ob sie präzise Ausführungsergebnisse, kurze Wartezeiten bis zur Ausführung oder ein Verhältnis beider Anforderungen präferieren [SBLW22a; SBLW22b]. Aufgrund der aktuellen NISQ-Ära ist

das Messen präziser Ergebnisse auf den aktuellen Quantencomputern eine große Herausforderung, um Quantencomputing praktikabel anwenden zu können (Abschnitt 2.1.3) [Pre18]. Durch die Zugriffsbeschränkungen der Hardwareanbieter auf deren Quantencomputer und die zunehmende Anzahl Nutzender nehmen auch die Wartezeiten für das Ausführen von Quantenschaltkreisen zu [LaR19; SBLW22a]. Durch das Bestimmen eines Verhältnisses beider Präferenzen können die Nutzenden festlegen, inwiefern die Wünsche nach kurzen Wartezeiten und präzisen Ergebnissen zueinander stehen sollen. Zusätzlich können sie angeben, wie viele Kombinationen die MarieQurie-Methode höchstens empfehlen soll.

4.3.1 Analyse von Quantenschaltkreisen

Um Kombinationen aus Quantenschaltkreisen, Quantencompilern und Quantencomputern wählen zu können, werden die Schaltkreise aus Schritt B in Abbildung 4.2 an Schritt C in Abbildung 4.3 übergeben sowie die Anforderungen nach Präzision, Wartezeiten und Anzahl Kombinationen der Nutzenden abgefragt. Für die Wahl geeigneter Kombinationen werden die Schaltkreise basierend auf den folgenden Metriken analysiert (Abschnitt 2.1.2):

a) Breite
b) Tiefe
c) Multi-Qubit-Gattertiefe
d) Anzahl Operationen
e) Anzahl Ein-Qubit-Gatter
f) Anzahl Multi-Qubit-Gatter
g) Anzahl Messoperationen

Die Metriken wurden basierend auf existierenden Arbeiten [LaR19; VBLW21; WBL+21] und der Untersuchung verfügbarer Analysefunktionen der SDKs Qiskit [Qis23], Cirq [Cir23], Forest [Rig21] und pytket [SDC+20] bestimmt [SBLW22a]. Die Metrikwerte werden an Schritt D übergeben.

4.3.2 Selektion von Quantenressourcen

Neben den Schaltkreismetrikwerten werden zur Selektion geeigneter Kombinationen die aktuellen Werte der im Folgenden aufgelisteten Hardwaremetriken (Abschnitt 2.1.3) und Wartezeiten der Quantencomputer mithilfe des Provenance-Systems *QProv* [WBL+21] abgerufen:

a) Durchschnittliche Ein-Qubit-Gatterfehlerrate
b) Durchschnittliche Multi-Qubit-Gatterfehlerrate
c) Durchschnittliche Ein-Qubit-Gatterzeit
d) Durchschnittliche Multi-Qubit-Gatterzeit
e) Durchschnittliche Messfehlerrate
f) Durchschnittliches T_1
g) Durchschnittliches T_2
h) Länge der Warteschlange

Die Hardwaremetriken wurden analog zu den Schaltkreismetriken (Abschnitt 4.3.1) bestimmt [SBLW22a]. Zusätzlich wurden für die Sammlung der Hardwaremetriken die Schnittstellen der Hardwareanbieter IBMQ, Google und Rigetti untersucht. QProv ruft die aktuellen Hardwaremetrikwerte sowie Verfügbarkeit und Wartezeiten von Quantencomputern angebundener Hardwareanbieter periodisch ab, persistiert diese und stellt sie anbieterübergreifend vereinheitlicht bereit [WBL+21].

Wird lediglich eine Ausführung mit kurzen Wartezeiten benötigt, werden alle validen Kombinationen aus zur Verfügung stehenden Quantencompilern, -computern sowie den extrahierten Schaltkreisen ermittelt. Kombinationen sind valide, wenn deren Compiler die enthaltenen Quantencomputer zur Kompilierung über die SDK nativ unterstützen (Abschnitt 2.2.2). Dadurch ist die Formatkompatibilität sowie die Bereitstellung aktueller Quantencomputerdaten für die Kompilierung sichergestellt. Die daraus entstandene Liste wird anhand der Wartezeiten für die Quantencomputer aufsteigend sortiert und auf den definierten Grenzwert gekürzt.

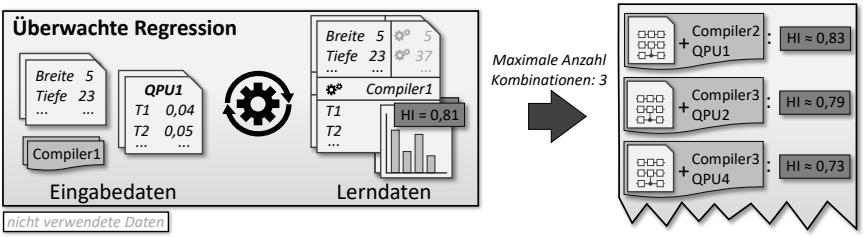

Abbildung 4.4: Verarbeitung verfügbarer Daten mit überwachter Regression zur Vorhersage der Ergebnispräzision aus Kombinationen von Quantenschaltkreisen, Quantencompilern und Quantencomputern

Liegt die Präferenz auf präzisen Ausführungsergebnissen, kann zwischen verschiedenen Implementierungen von überwachten ML-Algorithmen (Abschnitt 2.3.1) gewählt werden, um eine Ergebnispräzisionsvorhersage für die diversen Kombinationen berechnen zu können. Als Grundlage für die Vorhersage greift die gewählte ML-Implementierung auf die Metrikwerte ehemals ausgeführter Schaltkreise, den damals verwendeten Compilern und Quantencomputern sowie den zugehörigen präzisionsbestimmenden HI-Werten (Abschnitt 2.1.5) zu, dargestellt in Abbildung 4.4. Die ML-Implementierung wird bei jedem Aufruf der MarieQurie-Methode ausgeführt, um stets die neusten Ergebnisse für die Selektion zu berücksichtigen. Ziel dieses Lernprozesses ist, den *mittleren absoluten Fehler (engl. „Mean Absolute Error (MAE)“)* [WM05] zwischen den tatsächlichen und den von der ML-Implementierung geschätzten HI-Werten zu minimieren. Die aktuell zu betrachtenden Schaltkreis- und Quantencomputermetrikwerte und die verfügbaren Compiler werden normiert und darauf das Erlernte der ML-Implementierung angewendet (Abschnitt 2.3.1). Die HI-Werte werden für alle validen Kombinationen der aktuellen Eingabedaten durch die ML-Implementierung geschätzt, anhand dessen absteigend sortiert und durch den Grenzwert der Nutzenden gekürzt. Sind keine persistierten Lerndaten mit historischen HI-Werten vorhanden, wird die Selektion geeigneter Kombinationen (Schritt D in Abbildung 4.3) ausgelassen und alle validen Kombinationen werden zurückgegeben.

Wählen Nutzende ein Verhältnis aus kurzen Wartezeiten und präzisen Ergebnissen, wird die jeweilige Präferenzvariante zunächst separat ohne Kürzung berechnet und sortiert. Um beide Ranglisten zu vereinen, wird die *Borda-Wahl* (engl. „*Borda count*“) angewendet [BKS+21; WJZZ09]: Erstplatzierte Kombinationen der jeweiligen Rangliste erhalten $n-1$ Punkte, Zweitplatzierte $n-2$ Punkte et cetera. Letztplatzierte erhalten 0 Punkte. Hierbei entspricht n der Anzahl der Kombinationen aus Schaltkreisen, Compilern und Quantencomputern. Die Punkte werden listenübergreifend für jede Kombination addiert und eine neue Rangliste basierend auf der absteigenden Gesamtpunktzahl wird erstellt und entsprechend dem gesetzten Grenzwert abgeschnitten. Wurde ein Verhältnis definiert, welches von einer 50:50 Verteilung abweicht, werden die Prozentanteile mit den Punkten der jeweiligen Rangliste multipliziert [Rus07]: Ist, wie in Abbildung 4.4 dargestellt, das vorgegebene Verhältnis beispielsweise 20:80, sodass die Präzision 20 % und eine kurze Wartezeit 80 % an Wichtigkeit für die Nutzenden hat, werden die Punktestände der Präzisionsrangliste mit 0,2 und der Wartezeitliste entsprechend mit 0,8 vor Aufsummierung und Kürzung multipliziert. Die auserwählten Kombinationen aus Schaltkreisen, Compilern und Quantencomputern werden schließlich an den nächsten Schritt weitergegeben.

4.4 Übersetzung von Quantenschaltkreisen

Um die Kompilierung eines selektierten Quantenschaltkreises mit dem zugeordneten Compiler sicherzustellen, wird das Format des Schaltkreises mit den unterstützen Formaten des SDKs des zugeordneten Compilers verglichen. Gibt es keine Übereinstimmung, ist eine Übersetzung des Schaltkreises nötig. Ist das Schaltkreisformat ebenfalls ein SDK-Format des Compilers, bedarf es keiner Übersetzung. Stattdessen kann der Schaltkreis direkt an den zugehörigen Compiler übergeben werden.

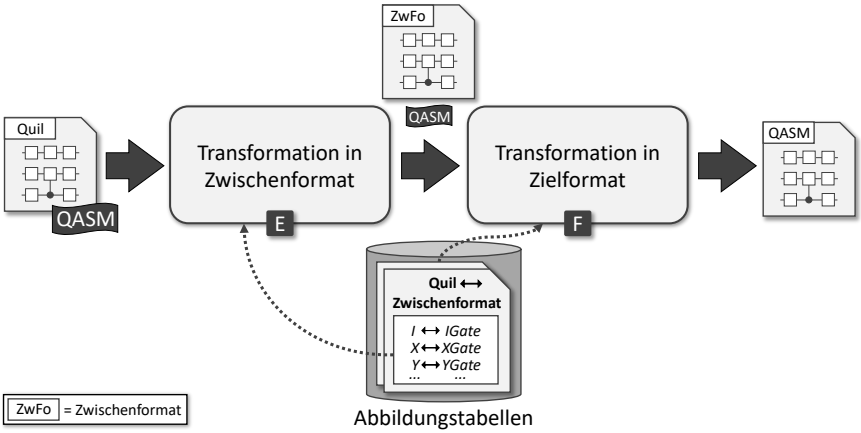

Abbildung 4.5: Übersetzung von Quantenschaltkreisen in Zwischen- und Zielformat

4.4.1 Übersetzung mittels Zwischenformat

Entspricht der Schaltkreis keinem der unterstützten SDK-Formate, wird dieser entsprechend automatisiert übersetzt, siehe Abbildung 4.5. Liegt beispielsweise ein Schaltkreis im Format *Quil* vor und soll mit dem Qiskit Transpiler [Qis23] für einen IBMQ Quantencomputer kompiliert werden, muss dieser in das Format *OpenQASM* oder *Qiskit* übersetzt werden (Abschnitt 2.2.2). Um die Anzahl der verschiedenen Übersetzungsvarianten in die unterschiedlichen Formate gering zu halten, wird ein Zwischenformat, siehe *ZwFo* in Abbildung 4.5, verwendet (Abschnitt 7.2.3) [HW04]. Der zu übersetzende Schaltkreis wird zunächst von seinem Ausgangsformat in das Zwischenformat übersetzt, dargestellt mit Schritt E. Im Zwischenformat befindend, wird der Schaltkreis in das Zielformat des SDKs übersetzt, beispielsweise *OpenQASM*, siehe Schritt F. Bei der jeweiligen Übersetzung werden der Reihenfolge entsprechend Schritt für Schritt die einzelnen enthaltenen Anweisungen des Schaltkreises betrachtet. Im Schaltkreis verwendete Qubits, Bits und Gatter werden mithilfe von Abbildungstabellen in äquivalente Anweisungen in der Syntax des Zwischen- beziehungsweise Zielformats

transformiert. Für jedes unterstützte Format liegt eine Abbildungstabelle für die Übersetzung von und in das Zwischenformat vor.

4.4.2 Ersetzung nicht-unterstützter Quantengatter

Ein Gatter im Ausgangsformat, das nicht nativ im Zwischen- oder Zielformat enthalten ist, wie beispielsweise solche, die den Gattern eines spezifischen Quantencomputers entsprechen, werden durch eine Folge gegebener Gatter (Abschnitt 2.1.3) ersetzt. Alternativ kann die Ersetzung eines Gatters mit der Beschreibung einer unitären Matrix als benutzerdefiniertes Gatter durchgeführt werden, wenn das jeweilige SDK und das zugehörige Format die Definition solcher benutzerdefinierten Gatter bereitstellen.

4.5 Diskussion und Abgrenzung zu verwandten Arbeiten

Im Bereich des Cloud Computings gibt es unter anderem die Ansätze von Sáez, Andrikopoulos und Leymann [SAL16] und Sáez et al. [SALS14] zur Bestimmung geeigneter Rechenressourcen und der Verteilung einer Anwendung auf diesen in der Cloud, abhängig von der aktuellen Auslastung anhand definierbarer Richtlinien. Für die Vorhersage benötigter klassischer Cloudressourcen existieren verschiedene Arbeiten, die die Anwendung von ML auf historische Daten vorschlagen [IKLL12; ON23; VGN+16]. Die Arbeiten beziehen sich jedoch nicht auf den Bereich des Quantencomputings.

Der vorgestellte ML-basierte Selektionsmechanismus benötigt, nebst den Messergebnissen und verwendeten Compilern, die Metrikwerte der Schaltkreise und Quantencomputer historischer Ausführungen, um geeignete Kombinationen aus Quantenschaltkreisen, Quantencompilern und Quantencomputern wählen zu können. Nichtsdestotrotz ermöglicht das Verfahren, die Kompilierung aller möglichen Kombinationen zu ersparen, um mit wachsender Zahl an SDKs und Quantencomputern den Aufwand zu reduzieren und den Nutzenden effizienter geeignete Kombinationen vorschlagen zu können. Proctor et al. [PRY+22] bestärken die Berücksichtigung vielfältiger Metriken, indem

sie mit ihrer Arbeit zeigen, dass für das Benchmarking von Quantencomputern zusätzlich neben der Breite und Tiefe auch die Struktur der Schaltkreise sowie die Fehlerstruktur maßgebend für eine erfolgreiche Ausführung sind.

In der Arbeit von Quetschlich, Burgholzer und Wille [QBW23d] wird für einen gegebenen Quantenschaltkreis automatisiert die geeignetste Kombination bestehend aus Quantencomputerarchitektur, Quantencomputer, Compiler und Compilereinstellungen mittels ML gewählt. Compilereinstellungen beschreiben die verschiedenen Optimierungsstufen und Strategien der verwendeten Compiler. Für die Evaluation, wie präzise die Ausführungsergebnisse einer Kompilation in Zukunft zu sein verspricht, definieren sie eine Metrik, die die Gatterfidelitäten sowie die Messfidelitäten der verfügbaren Qubits berücksichtigt. In einer weiteren Arbeit von Quetschlich, Burgholzer und Wille [QBW23a] wird bestärkendes Lernen (engl. „*Reinforcement Learning*“) verwendet, um die geeignetste Kombination zu wählen. Hierbei werden zusätzlich noch die verschiedenen Kompilierungsstrategien unterschiedlicher Compiler zusammengestellt, um die Fehleranfälligkeit der Kompilationen verbessern zu können. Mit ihren beiden Arbeiten [QBW23a; QBW23d] zeigen sie den Einfluss der verschiedenen Kompilierungskombinationen auf die geschätzte Qualität der Ausführung auf. Die Kompilationen werden jedoch nicht ausgeführt, wodurch kein Abgleich mit tatsächlichen Messergebnisse stattfindet. Zudem berücksichtigen sie keine Wartezeiten für den Zugriff auf die Quantencomputer und fragen entsprechend nicht die Präferenzen der Nutzenden ab. Hingegen unterstützt die MarieQurie-Methode die Selektion geeigneter Kombinationen nach den Präferenzen kurzer Wartezeiten und präziser Ergebnisse (Abschnitt 4.3). Weitere Präferenzen können hinzugefügt werden, indem die Bedingungen beispielsweise als mathematische Formeln für die Selektion definiert werden.

In der Arbeit von Wang et al. [WLC+22] wird die Fidelität eines Schaltkreises für NISQ-Rechner mittels ML vorhergesagt. Hierbei verwenden sie *PST* (Abschnitt 2.1.6.3) als Metrik, um die Ergebnispräzision zu erfassen. Sie betrachten jedoch keine existierenden Compiler für die Kompilierung, sodass keine Empfehlung geeigneter Kombinationen ausgegeben wird.

Die Arbeit von Suchara et al. [SKF+13] ermittelt die Anzahl physischer Gatter und Qubits sowie die Ausführungszeit und notwendige Ressourcen zur Fehlerkorrektur für gegebene Schaltkreise und verschiedene Quantentechnologien. Hierfür betrachten sie jedoch alle möglichen Kombinationen und nehmen keine Reduktion dieser Anzahl vor. Beverland et al. [BMT+22] schätzen basierend auf einer Quantenalgorithmusimplementierung und physikalischen Qubiteigenschaften wie Gatterfehlerraten und -zeiten, notwendige Quantenressourcen wie Anzahl der Qubits, Rechenzeit und Ressourcenbedarf für Fehlerkorrektur ab. Für die Abschätzung kompilieren sie die Implementierung auf eine Menge vorgegebener Operationen. Ihr Fokus liegt darauf, abzuschätzen, welche Voraussetzungen Quantencomputer erfüllen müssen, um praktische Anwendungen ausführen zu können. Verfügbare Quantencomputer für die Ausführung werden nicht berücksichtigt.

Die automatisierte Übersetzung ermöglicht aufgrund der heterogenen SDKs die Kompilierung eines Quantenschaltkreises mit unterschiedlichen Compilern. Durch die schrittweise Übersetzung der Schaltkreise in andere Formate kann der Fall auftreten, dass diese an Größe zunehmen, wodurch die Kompilationen weniger effizient ausfallen und die Messergebnisse schlechter werden könnten als für den originalen Schaltkreis. Zudem enthält die vorliegende Arbeit kein Verfahren, dass die Äquivalenz der übersetzten Schaltkreise mit den originalen Schaltkreisen verifiziert. Verschiedene Arbeiten [CCD+23; SDC+20; TGO+22] können Quantenschaltkreise in diverse Formate übersetzen, um diese schließlich für unterschiedliche Hardwarehersteller zu kompilieren und auszuführen. Sie geben jedoch keine Empfehlung zu geeigneten Kompilierungskombinationen.

Kapitel 5

Kompilieren und Analysieren der Ausführbarkeit von Quantenschaltkreisen

Die Ausführung eines Quantenschaltkreises auf einem NISQ-Rechner kann Fehler verursachen. Dieses Kapitel präsentiert Forschungsbeitrag 3 und diskutiert (i) den Einfluss der Schaltkreisgröße auf die Ausführbarkeit (Abschnitt 5.2). Die Größe wird durch die Dateninitialisierung beeinflusst, weshalb (ii) Kodierungsmuster präsentiert werden (Abschnitt 5.3). Auch die Kompilierung beeinflusst die Größe, sodass mit Schritt 4 der MarieQurie-Methode (iii) die Kompilierung auf Quantencomputern durch diverse Compiler vorgestellt wird (Abschnitt 5.4). Mit Schritt 5 werden (iv) die Kompilationen analysiert und selektiert (Abschnitt 5.5). Die Konzepte wurden bereits publiziert [SBB+20; SBL+21; SBLW20; WBLS20; WBLS21b; WBLS21c].

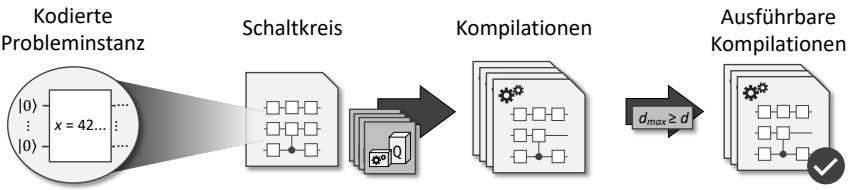

Abbildung 5.1: Vorgehen zur Kompilierung von Quantenschaltkreisen und Analyse der Ausführbarkeit von Quantenkompilationen

5.1 Idee und Grundkonzept

In diesem Forschungsbeitrag wird die Ausführbarkeit von Quantenschaltkreisen auf NISQ-Rechnern analysiert. Dabei wird die Größe eines Schaltkreises, welche einen wesentlichen Faktor in Bezug auf die Ausführbarkeit darstellt, unter anderem von der Probleminstanz und der Kodierung dieser beeinflusst (Abschnitte 2.1.2 und 2.2.1). Außerdem ändert sich die Schaltkreisgröße durch die Kompilierung auf den Quantencomputer. Um die Anzahl an Ausführungen und somit Kosten zu reduzieren, sollte die Ausführbarkeit der Schaltkreise beziehungsweise ihrer Kompilationen zuvor untersucht werden, wie in Abbildung 5.1 gezeigt. Die Untersuchung erfordert Aufwand und Expertise. SDKs stellen häufig Funktionen zur Untersuchung der Kompilationen bereit, unterscheiden sich jedoch in ihrer Handhabung (Abschnitt 2.2.2). Dabei bedarf die Ausführbarkeitsanalyse zusätzlich der Berücksichtigung aktueller Hardwaremetrikwerte. Die MarieQurie-Methode automatisiert die Kompilierung von Schaltkreisen mit diversen Compilern und Quantencomputern. Die Methode ermöglicht die automatisierte Analyse der Kompilationen und Quantencomputer, schätzt anhand dessen die Ausführbarkeit der Kompilationen und empfiehlt nur solche, die als ausführbar eingestuft werden. Dadurch müssen Nutzende sich nicht mit verschiedenen Technologien auseinandersetzen, mögliche Fehler bei manuellen Analysen und Berechnungen können vermieden werden und die Automatisierung reduziert den Aufwand.

5.2 Ausführbarkeit von Quantenschaltkreisen

Es existieren bereits verschiedene Metriken, mit denen die Performanz von Quantencomputern gemessen und verglichen werden kann (Abschnitt 2.1.6). Jedoch liegt deren Fokus nicht auf der Bestimmung der Ausführbarkeit eines beliebigen Schaltkreises. Ein Quantenschaltkreis gilt als erfolgreich ausführbar, wenn die Ergebnisse einer zukünftigen Ausführung nicht fundamental durch aufgetretene Fehler gestört und eindeutig identifiziert werden können [LB20; SBL+21; SBLW20].

Verschiedene Arbeiten [BBC+17; CBS+19; MBB+18; Pre18] nehmen an, dass mit der Größe eines Quantenschaltkreises bestimmt werden kann, ob dieser Schaltkreis erfolgreich ausführbar ist [CBS+19; LB20]:

$$wd \ll \frac{1}{\epsilon_{\text{eff}}} \tag{5.1}$$

Hierbei muss die Breite w multipliziert mit der Tiefe d viel kleiner sein als das multiplikative Inverse der effektiven Fehlerrate ϵ_{eff} des verwendeten Quantencomputers (Abschnitt 2.1.6.2) [CBS+19; LB20; MBB+18]. Entspricht die Größe des Schaltkreises ungefähr dem Wert von $\frac{1}{\epsilon_{\text{eff}}}$ oder übersteigt diesen, ist die Annahme, dass die Ausführung des Schaltkreises höchstwahrscheinlich fehlschlägt [CBS+19]. Eine genaue Zusammensetzung von ϵ_{eff} ist bisher nicht bekannt [BBC+17; MBB+18]. Es wird jedoch angenommen, dass verschiedene Faktoren, wie beispielsweise hardwareseitig die Gatterfehler, verfügbare Gattertypen und Qubitkonnektivität (Abschnitt 2.1.3), aber auch die Parallelisierung und somit der Abbildungsalgorithmus des Quantencompilers (Abschnitt 2.1.4) bei den vielen Ausführungen von unterschiedlichen Schaltkreisen mit einer ursprünglichen Tiefe von eins zur Bestimmung von ϵ_{eff} (Abschnitt 2.1.6.2) einfließen [BBC+17; MBB+18].

Entsprechend Gleichung (5.1) muss ein Bereich beziehungsweise Intervall kleiner $\frac{1}{\epsilon_{\text{eff}}}$ existieren, für das gilt, dass wenn die Größe des Schaltkreises innerhalb des Intervalls liegt, dieser nicht ausführbar ist. Liegt die Schalt-

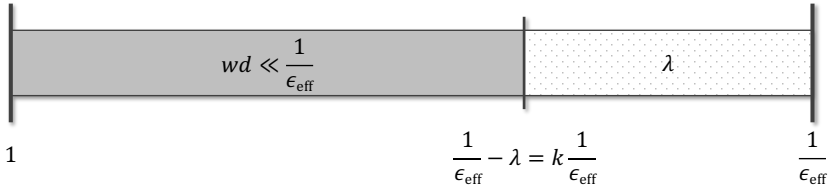

Abbildung 5.2: Grenze der Größe ausführbarer Quantenschaltkreise (adaptiert von [SBLW20])

kreisgröße darunter, ist dieser erfolgreich ausführbar. Dabei muss dieses noch unbekannte, in Abbildung 5.2 skizzierte, Intervall $]\frac{1}{\epsilon_{\text{eff}}} - \lambda, \frac{1}{\epsilon_{\text{eff}}}[$ bestimmt werden, um die Unschärfe der Gleichung (5.1) aufzulösen. Die untere Grenze des Intervalls und somit die obere Grenze des ausführbaren Größenbereichs lässt sich durch einen Faktor k mit $k \cdot \frac{1}{\epsilon_{\text{eff}}}$ ermitteln, wodurch sich die folgende verfeinerte Metrik ergibt [SBLW20]:

$$wd < k\frac{1}{\epsilon_{\text{eff}}} \tag{5.2}$$

Für den praktischen Einsatz der Metrik müssen k beziehungsweise λ sowie ϵ_{eff} zukünftig durch Benchmarking [BY20; LJV+21; QBW23b] auf diversen Quantencomputern individuell ermittelt werden, um präzise Vorhersagen bezüglich der Ausführbarkeit von Quantenschaltkreisen treffen zu können.

5.3 Muster zur Kodierung klassischer Daten

Die Breite und Tiefe von Quantenschaltkreisen wird mitunter durch die Kodierung der klassischen Probleminstanz in einen Quantenzustand beeinflusst [LB20; WBLS20]. Dabei werden abhängig von der Kodierung die Daten klassisch aufbereitet und entsprechende Quantengattersequenzen erzeugt, um diese als Zustandspräparationsroutine auf dem Quantencomputer auszuführen [LB20]. Das Kodierungsverfahren kann dabei die Laufzeit des ge-

samten Quantenalgorithmus beeinträchtigen [Aar15; WBLS20; WBLS21b]: Erfordert das Verfahren exponentielle Laufzeit, wird der ansonsten beispielsweise logarithmisch schnelle Quantenalgorithmus ebenfalls exponentiell, sodass der theoretische Quantenvorteil ausbleibt [WBLS20].

Im Folgenden werden Muster (Abschnitt 2.4) zu Kodierungen von Eingabedaten für Quantenalgorithmen präsentiert, um einen Überblick über die verschiedenen Arten dieser zu geben. Sie beschreiben den Einsatz und die Lösungsansätze der Kodierungen, sodass diese für Personen mit Expertise aus verschiedenen Disziplinen lesbar sind [WBLS20]. Die Anwendung der Kodierungsmuster zur Implementierung von Quantenalgorithmen ist jedoch nicht Teil der vorliegenden Arbeit, da insbesondere die Analyse der in den Implementierungen beschriebenen Schaltkreise im Fokus steht. Eine Lösung ist als Muster formalisiert, wenn sie nach dem Entstehungsprozess von Fehling et al. [FBBL14] bei der Untersuchung von wissenschaftlichen Arbeiten, technischen Dokumentationen und Büchern im Kontext der Zustandspräparation klassischer Daten für Quantenalgorithmen nach der Regel von Coplien [Cop96] mindestens dreimal beschrieben wurde [WBLS20; WBLS21b; WBLS21c]. Eine Implementierung der Lösung ist aufgrund der Einschränkungen von NISQ-Rechner in dieser Arbeit keine Voraussetzung [WBLS20].

Das Muster Initialization aus der Arbeit von Leymann [Ley19] beschreibt im Allgemeinen die Initialisierung des Quantenzustands zu Beginn eines Quantenalgorithmus. Für die Umsetzung von Quantenalgorithmen und der Empfehlung geeigneter Quantenressourcen sind jedoch nähere Informationen für die Kodierung klassischer Daten notwendig. Daher wurden weitere Muster identifiziert, die Initialization verfeinern [WBLS20]: Basis Encoding [WBLS20], QRAM Encoding [WBLS21b; WBLS21c], Amplitude Encoding [WBLS20] und Angle Encoding [WBLS21b; WBLS21c] als Kodierungen und QuAM [WBLS20] und Schmidt Decomposition [WBLS21b] als Kodierungsverfahren. Für jede der Kodierungen und Verfahren müssen die folgenden *Kräfte* bei der Verwendung für einen Quantenalgorithmus abgewogen werden [WBLS20; WBLS21b; WBLS21c]: Aufgrund der wenigen verfügbaren Qubits heutiger Quantencomputer sollte die benötigte Breite

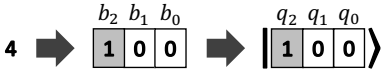

Abbildung 5.3: Lösungsskizze Basis Encoding. Binärapproximation einer Zahl und Darstellung dieser als Basisvektor (adaptiert von [WBLS20])

für die Kodierung möglichst klein sein. Des Weiteren sollte die Kodierung aufgrund von Fehlerraten und Dekohärenzzeiten eine möglichst geringe Tiefe des Quantenschaltkreises in Anspruch nehmen, sodass die Laufzeit für die Zustandserzeugung bestenfalls höchstens logarithmisch oder linear ist. Um die Daten weiterverarbeiten zu können, müssen sie außerdem das entsprechende Format haben, sodass beispielsweise arithmetische Berechnungen durchgeführt werden können. Die folgenden Muster sowie deren zugehörige Mustersymbole[1] wurden bereits in vorherigen Arbeiten veröffentlicht [WBLS20; WBLS21b; WBLS21c].

5.3.1 Basis Encoding

Abbilden eines numerischen Datenelements auf ein Quantenregister.

Kontext: Ein Quantenalgorithmus benötigt zur Berechnung eine reelle Zahl x.

Lösung: Bei der Basiskodierung (engl. „*Basis Encoding*“) werden zur Kodierung von Eingabedaten die Basiszustände $\{|0...00\rangle, |0...01\rangle, ..., |1...11\rangle\}$ verwendet. Eine Zahl x wird binär approximiert, sodass $x := b_{n-1}...b_1 b_0$ gilt und als Basisvektor $|x\rangle := |b_{n-1}...b_1 b_0\rangle$ dargestellt werden kann. In Abbildung 5.3 entspricht beispielsweise die Zahl 4 binär 100 und als Basisvektor $|100\rangle$. Somit lautet die Kodierung $x \approx \sum_{i=-k}^{m} b_i 2^i \mapsto |b_m..., b_0, b_{-1}, ..., b_{-k}\rangle$, sodass x mit $k+m+1$ signifikanten Stellen angenähert wird.

[1] CC-BY: `https://quantumcomputingpatterns.org`

Ergebnis: Basis Encoding entspricht einer digitalen Kodierung und eignet sich für arithmetische Berechnungen [LB20]. Bei der Approximation einer Zahl wird pro Stelle ein Qubit benötigt. Für die Darstellung einer 1 muss der initiale Zustand $|0\rangle$ durch ein Gatter in den Zustand $|1\rangle$ überführt werden. Die Kodierung erfolgt mit einer Tiefe von 1 in konstanter Zeit.

Verwandte Muster: Das Muster verfeinert Initialization [Ley19]. Benötigt ein Quantenalgorithmus als Eingabe mehrere Zahlen, können diese mittels Basis Encoding kodiert und mit dem QuAM-Muster verarbeitet werden. Quantum Phase Estimation (QPE) [WBLS21b], welches die Eigenwerte unitärer Matrizen bestimmt, und Alternating Operator Ansatz (AOA) [WBLV21] sowie Quantum Approximate Optimization Algorithm (QAOA) [WBLV21] zur Lösungsannäherung von Optimierungsproblemen erzeugen Ausgaben in Basis Encoding. Readout Error Mitigation [BBL+22b] nutzt das Muster zur Erzeugung von Kalibrierungsmatrizen bei der Fehlerminderung.

Anwendungsbeispiele: Vedral, Barenco und Ekert [VBE96] präsentieren verschiedene Quantenalgorithmen, die arithmetische Operationen auf basiskodierten Zahlen ausführen. Formal ist die gezeigte Lösung in den Arbeiten von Cortese und Braje [CB18] und Leymann und Barzen [LB20] zu finden.

5.3.2 Quantum Associative Memory (QuAM)

Abbilden einer Menge numerischer Datenelemente auf ein Quantenregister.

Kontext: Ein Quantenalgorithmus benötigt für Berechnungen eine Menge X an numerischen Datenelementen.

Lösung: Um die Superposition mehrerer basiskodierter Zahlen in einem Quantenregister zu erzeugen, wird ein assoziativer Quantenspeicher (engl. „*Quantum Associative Memory (QuAM)*“) [VM00] verwendet [LB20]. Hierbei werden die basiskodierten Zahlen als Basisvektoren mit jeweils einer Amplitude von $\frac{1}{\sqrt{n}}$ in eine gleichgewichtete Superposition gebracht, wobei n die Anzahl der Datenelemente ist. In Abbildung 5.4 sind die basiskodierten

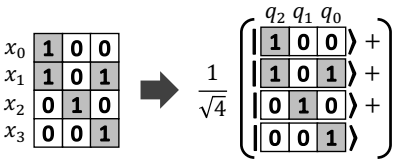

Abbildung 5.4: Lösungsskizze QuAM-Kodierung. Binärzahlen werden basiskodiert und in eine gleichgewichteten Superposition gebracht (adaptiert von [WBLS20])

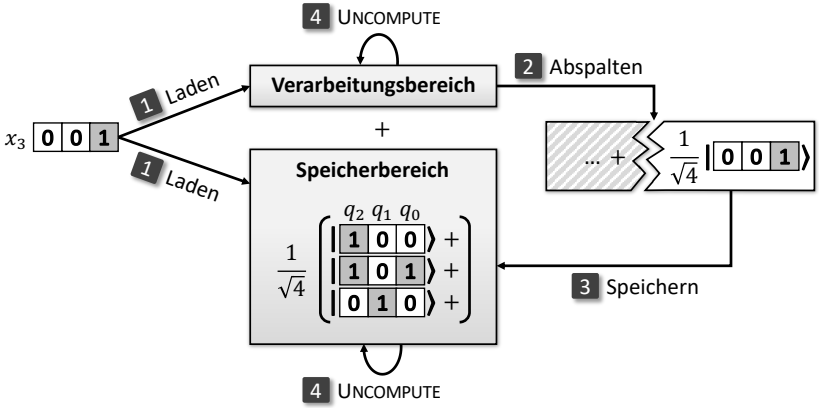

Abbildung 5.5: Darstellung des QuAM-Verfahrens von Ventura und Martinez [VM00] zur Initialisierung von mehreren Datenelementen (adaptiert von [WBLS20])

Zahlen x_0, x_1, x_2 und x_3 mit Amplituden der Größe $\frac{1}{\sqrt{4}}$ als gleichgewichtete Superposition zu sehen.

Ventura und Martinez [VM00] präsentieren ein Verfahren, um die Daten mittels QuAM in ein Quantenregister zu laden, dargestellt in Abbildung 5.5. Hierbei befindet sich das Quantenregister des QuAM in Superposition zweier Bereiche: dem *Verarbeitungsbereich* und dem *Speicherbereich*. Ein neues Datenelement x_3 wird im ersten Schritt in beide Bereiche geladen. Im zweiten Schritt wird das basiskodierte neuen Element von den bisher hinzugefügten Datenelementen im Verarbeitungsbereich abgespalten und ihm wird eine

Amplitude zugefügt. Dadurch befindet sich das neue Element in Superposition mit den bisherigen Elementen und wird im dritten Schritt gespeichert. Zuletzt wird ein UNCOMPUTE [Ley19] durchgeführt, wodurch sich beide Bereiche für den nächsten Durchgang wieder im Zustand $|0\rangle$ befinden.

Ergebnis: Bei der Kodierung handelt es sich um eine digitale Kodierung und sie ist für arithmetische Berechnungen geeignet [LB20]. Für n Zahlen, die durch l Ziffern angenähert werden, sind l Qubits nötig, wobei jede Zahl durch einen Basisvektor mit einer Amplitude von $\frac{1}{\sqrt{n}}$ dargestellt wird. Die restlichen $2^l - n$ Amplituden haben den Wert 0, wodurch die Amplitudenvektoren meist dünn besetzt sind [SP18].

Verwandte Muster: Das Muster ist einer Verfeinerung von INITIALIZATION [Ley19] und verwendet die Muster UNCOMPUTE und UNIFORM SUPERPOSITION aus der Arbeit von Leymann [Ley19] sowie BASIS ENCODING.

Anwendungsbeispiele: Zur Kodierung mehrerer Datenelemente in BASIS ENCODING kann das Verfahren von Ventura und Martinez [VM00] eingesetzt werden. Die Kodierung wird in den Algorithmen von Shor [Sho97] und Grover [Gro96] sowie der Quantenfouriertransformation [Cop02] angewendet.

5.3.3 Quantum Random Access Memory (QRAM) Encoding

Verwenden eines Quantenarbeitsspeichers, um auf eine Superposition von Datenelementen zuzugreifen.

Kontext: Ein Quantenalgorithmus benötigt Direktzugriff auf einzelne Elemente der Eingabedaten.

Lösung: Wie beim klassischen Arbeitsspeicher (engl. „*RANDOM ACCESS MEMORY (RAM)*“), besteht der Quanten-RAM (QRAM) aus einem Speicher, einem Adress- und einem Ausgaberegister, wobei es sich bei QRAM um Quantenregister handelt [GLM08]. Jeder Speicherzelle, in der jeweils ein Datenelement in BASIS ENCODING gespeichert ist, ist ein Index des Adressregisters zugeordnet. Wird im Adressregister ein Zustand erzeugt, wird das Datenelement der zugehörigen Speicherzelle im Ausgaberegister ausgegeben. Wird im

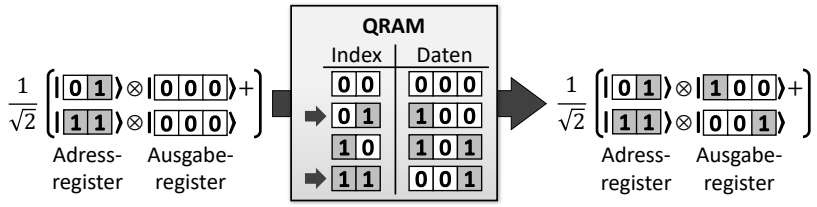

Abbildung 5.6: Lösungsskizze QRAM Encoding. Das Adressregister wird mit zwei Indizes in Superposition initialisiert. Daraufhin werden die Daten der zugehörigen Speicherzellen in das Ausgaberegister geladen (adaptiert von [WBLS21b; WBLS21c])

Adressregister eine Superposition an Indizes erzeugt, wird eine Superposition an gespeicherten Datenelementen im Ausgaberegister ausgegeben, sodass sich beide Register in Superposition befinden. In Abbildung 5.6 wird die Superposition zweier Indizes geladen: $\frac{1}{\sqrt{2}}(|01\rangle + |11\rangle)$. Dies führt mit einem zuvor leeren Ausgaberegister durch QRAM Encoding in den Zustand $|\psi\rangle = \frac{1}{\sqrt{2}}(|01\rangle \otimes |100\rangle + |11\rangle \otimes |001\rangle)$. Im Allgemeinen lautet die Kodierung von m Datenelementen $\frac{1}{\sqrt{m}} \sum_{i=0}^{m-1} |a\rangle_i |0\rangle \xrightarrow{\text{QRAM}} \frac{1}{\sqrt{m}} \sum_{i=0}^{m-1} |a\rangle_i |x_a\rangle$. Hierbei entspricht $|a\rangle_i$ dem Adressregisterindex des i-ten angefragten Datenelements und $|x_a\rangle$ dem Ausgaberegister mit dem zum Index zugehörigen Datenelement x_a [GLM08; SP18].

Ergebnis: QRAM Encoding ist eine digitale Kodierung [LB20]. Die Kodierung mittels Basis Encoding benötigt für Datenelemente mit l Binärstellen l Qubits [LB20]. Weitere n Qubits werden für 2^n Indizes im Adressregister benötigt [GLM08]. Aufgrund der Kodierung der Datenelemente in Superposition ist eine parallele Verarbeitung dieser wie bei QuAM möglich. Eine Zustandspräparation mittels QRAM Encoding ist theoretisch in logarithmischer Zeit möglich, allerdings existiert hierfür noch keine praktische Umsetzung, sodass Quantenalgorithmen, die diese Kodierung verwenden, nicht den theoretischen Quantenvorteil erzielen können [SP18].

Verwandte Muster: QRAM Encoding verfeinert von Initialization [Ley19] und verwendet Basis Encoding. Außerdem kann es Creating Entangle-

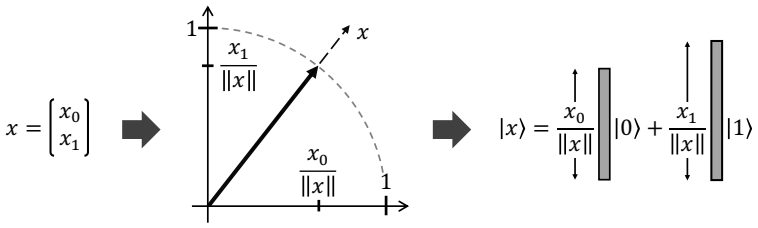

Abbildung 5.7: Lösungsskizze AMPLITUDE ENCODING. Der Datenvektor wird zunächst normiert und die zwei Datenpunkte werden anschließend in die Amplituden kodiert. (adaptiert von [WBLS20])

MENT [Ley19] für spezifische Zustände wie beispielsweise dem Greenberger-Horne-Zeilinger-Zustand ($\frac{1}{\sqrt{2}}|00\rangle \otimes |000\rangle + \frac{1}{\sqrt{2}}|11\rangle \otimes |111\rangle$) verwendet. Aufgrund der aktuell uneffizienten Realisierbarkeit von QRAM ENCODING können andere Verfahren zur Zustandspräparation eingesetzt werden.

Anwendungsbeispiele: QRAM ENCODING wird in verschiedenen Quantenalgorithmen verwendet [GLM08; MKF19]. Ein Beispiel ist der Algorithmus von Harrow, Hassidim und Lloyd (HHL) [HHL09] zur Lösung linearer Gleichungssysteme. Park, Petruccione und Rhee [PPR19] präsentieren eine Implementierung eines Flip-Flop-QRAMs für unsortiert gespeicherte Daten, die beispielhaft zur Kodierung von Daten für die Quantum Support Vector Machine eingesetzt wird.

5.3.4 Amplitude Encoding

Kodierung numerischer Vektoren in kompakter Form.

Kontext: Ein Quantenalgorithmus benötigt als Eingabe einen numerischen Datenvektor $x = (x_0, ..., x_{n-1})$.

Lösung: Für die Kodierung eines numerischen Datenvektors werden die Amplituden eines Quantenzustands verwendet [LB20]. Da die Summe der quadrierten Amplitudenwerte stets 1 ergibt, muss der Datenvektor normiert

werden. Zudem muss die Dimension des Vektors einer Zweierpotenz entsprechen, um auf die Amplituden abgebildet zu werden, da ein Quantenregister mit m Qubits 2^m Dimensionen besitzt. Alternativ kann der Vektor durch Nullen mit weiteren Dimensionen aufgefüllt werden (engl. „*padding*“) [LB20; SBSW20]. Die Kodierung eines zu normierenden Vektors x sieht wie folgt aus: $|x\rangle = \sum_{i=0}^{n-1} \frac{x_i}{\|x\|} |i\rangle$, wobei x_i der i-ten unabhängigen Variable von x entspricht [LC20]. Abbildung 5.7 zeigt einen zweidimensionalen Datenvektor, der normiert und dessen Variablenwerte kodiert werden.

Ergebnis: Die Amplitudenkodierung (engl. „*Amplitude Encoding*“) ist eine analoge Kodierung, da die Daten mit den Amplituden abgebildet werden [LB20]. Um einen n-dimensionalen Datenvektor abzubilden, werden $\lceil \log_2(n) \rceil$ Qubits benötigt. Bei beliebigen Daten sind 4^n Gatter nötig [LB20; SM05]. Wird von dem Basiszustand $|0\rangle$ ausgegangen, sind 2^n Gatter nötig. Durch weitere Anforderungen an den Datenvektor oder dessen dünne Besetzung können weitere Gattereinsparungen gemacht werden [LB20; SP18].

Verwandte Muster: Das Muster verfeinert Initialization [Ley19] und benötigt weniger Qubits als Basis, Angle und QRAM Encoding. Das Muster wird von QPE [WBLS21b] verwendet.

Variationen: Hur, Kim und Park [HKP22] präsentieren hybrides Amplitude Encoding, welches mehrere unabhängige Quantenregistern parallel verwendet, wodurch sich die Größe der Kodierungsschaltkreises reduziert.

Anwendungsbeispiele: Amplitude Encoding wird von verschiedenen Quanten-ML-Algorithmen [LC20] und dem HHL-Algorithmus [HHL09] verwendet. Dabei gilt für viele ML-Algorithmen die Annahme, dass die zu kodierenden Daten normierbar sind [DS19]. Für die Umsetzung der Amplitudenkodierung existieren verschiedene Verfahren: Das SDK PennyLane bietet ein Verfahren für Amplitude Encoding[1] und weitere zur Präparation beliebiger Zustände (engl. „*arbitrary state preparation*“)[2] an, wobei

[1] `https://docs.pennylane.ai/en/stable/code/api/pennylane.AmplitudeEmbedding.html`

[2] `https://docs.pennylane.ai/en/stable/code/api/pennylane.ArbitraryStatePreparation.html`

eines dieser auf der Arbeit von Mottonen et al. [MVBS04][1] basiert. Ein weiteres Verfahren zur beliebigen Zustandspräparation ist die SCHMIDT DECOMPOSITION [PB11]. Qiskit stellt eine Implementierung[2] für die beliebige Zustandspräparation basierend auf der Arbeit von Shende, Bullock und Markov [SBM05] bereit.

5.3.5 Angle Encoding

Abbilden eines Datenelements pro Qubit.

Kontext: Um Berechnungen innerhalb der Dekohärenzzeit auszuführen, benötigt ein Quantenalgorithmus eine effiziente Kodierung.

Lösung: Für die Winkelkodierung (engl. „*ANGLE ENCODING*“) werden die Elemente des zu kodierenden Datenvektors auf das Intervall $[0, \frac{\pi}{2}]$ normiert [CWV+20]. Jedes Element x_i wird durch ein Qubit dargestellt, auf welchem eine unitäre Rotationsoperation R_y ausgeführt wird, zu sehen in Abbildung 5.8 [LB20; LC20]:

$$R_y(2x) = \begin{pmatrix} \cos x & -\sin x \\ \sin x & \cos x \end{pmatrix} \tag{5.3}$$

Dadurch gilt $R_y(2x_i)\,|0\rangle = |x_i\rangle = \cos x_i \cdot |0\rangle + \sin x_i \cdot |1\rangle$, sodass der kodierte Datenvektor x als Tensorprodukt wie folgt aussieht [LB20]:

$$\left(\bigotimes_{i=0}^{n-1} R_y(2x_i)\right)|0...0\rangle = |x\rangle = \begin{pmatrix} \cos x_0 \\ \sin x_0 \end{pmatrix} \otimes ... \otimes \begin{pmatrix} \cos x_{n-1} \\ \sin x_{n-1} \end{pmatrix} \tag{5.4}$$

[1] `https://docs.pennylane.ai/en/stable/code/api/pennylane.MottonenStatePreparation.html`

[2] `hhttps://github.com/Qiskit/qiskit/blob/stable/1.0/qiskit/circuit/library/data_preparation/state_preparation.py`

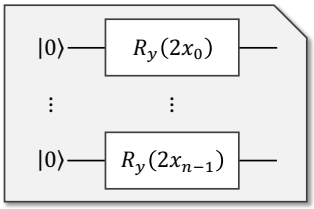

Abbildung 5.8: Lösungsskizze Angle Encoding. Quantenschaltkreis zur Kodierung der Daten mittels Rotationsgattern (adaptiert von [LB20; WBLS20])

Ergebnis: Angle Encoding ist eine analoge Kodierung [LB20]. Die Kodierung benötigt für n Dimensionen des Datenvektors n Qubits sowie ein Ein-Qubit-Gatter pro Qubit [LB20; LC20].

Verwandte Muster: Angle Encoding verfeinert Initialization [Ley19].

Variationen: LaRose und Coyle [LC20] präsentieren *Dense Angle Encoding*, durch das mithilfe der relativen Phase zwei Elemente pro Qubit dargestellt werden können. Hur, Kim und Park [HKP22] stellen eine hybride Variante von Angle Encoding vor, bei der die Kodierung auf mehreren unabhängigen Quantenregistern parallel ausgeführt wird, sodass die Größe der Kodierungsschaltkreises reduziert werden kann.

Anwendungsbeispiele: Angle Encoding wird unter anderem als Datenkodierung für Quanten-neuronale Netze verwendet [LC20; SP18]. Yan, Iliyasu und Venegas-Andraca [YIV16] verwenden die Kodierung, um die Farbe von Pixeln im Bereich der Quantenbildverarbeitung in einen Quantenzustand zu laden. Des Weiteren bietet PennyLane eine Implementierung zur Präparation von Daten in Angle Encoding an.[1]

[1] `https://docs.pennylane.ai/en/stable/code/api/pennylane.AngleEmbedding.html`

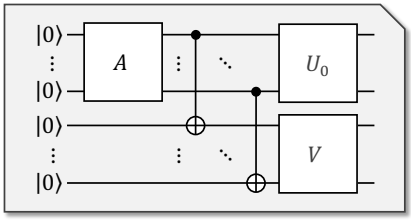

Abbildung 5.9: Lösungsskizze Schmidt Decomposition. Quantenschaltkreis zur Erzeugung eines beliebigen Zustands (adaptiert von [LB20; WBLS21b])

5.3.6 Schmidt Decomposition

Erzeugen eines beliebigen Quantenzustands.

Kontext: Auf einem mit $|0\rangle$ initialisierten Quantenregister soll ein beliebiger Zustand $|x\rangle$ erzeugt werden. Ist kein effizientes Verfahren zur Erzeugung dieses Zustands bekannt, kann eines für die beliebige Zustandspräparation angewendet werden.

Lösung: In den zwei Hilberträumen S und W kann $|x\rangle$ mithilfe der Orthonormalbasen $\{e_i\} \in S$ und $\{f_j\} \in W$ durch eine Linearkombination dieser dargestellt werden: $|x\rangle = \sum_{i,j} \beta_{ij} \cdot e_i \otimes f_j$ [LB20]. Daraufhin wird die Singulärwertzerlegung [GK65] der Matrix $M = \{\beta_{ij}\}$ berechnet:

$$M = \left(U_0 U_1\right) \begin{pmatrix} A \\ 0 \end{pmatrix} V^* \tag{5.5}$$

$U = (U_0 U_1)$ und V sind unitäre Matrizen [LB20]. Die Diagonalmatrix A enthält die Diagonalelemente $\{\alpha_0, ..., \alpha_{m-1}\}$, die sogenannten *Schmidt-Koeffizienten*, wobei $\sum_{i=0}^{m-1} \alpha_i = 1$ und $\alpha_i \in \mathbb{R} \geq 0$. Die Schmidt-Dekomposition von $|x\rangle$ lautet $|x\rangle = \sum_{i=0}^{m-1} \alpha_i \cdot u_i \otimes v_i$. Dabei stellen $\{u_0, ..., u_{m-1}\}$ und $\{v_0, ..., v_{m-1}\}$ die Spaltenvektoren von U_0 und V dar und bilden die *Schmidt-Basis* [DGK14;

LB20]. Abbildung 5.9 stellt den zugehörigen Schaltkreis für die Schmidt Decomposition dar. Für die Ausführung müssen die Gatter für A, U_0 und V noch in weitere Ein- und Multi-Qubit-Gatter aufgespalten werden [LB20].

Ergebnis: Das Quantenregister befindet sich im Zustand $|x\rangle$, wobei die Koeffizienten α_i bekannt sind, sodass das Maß an Verschränkung bestimmt werden kann [DGK14; NC11]. Abhängig von der Anzahl Qubits wächst die Tiefe des Schaltkreises im schlechtesten Fall exponentiell [PB11].

Verwandte Muster: Schmidt Decomposition ist eine Verfeinerung von Initialization [Ley19] und kann bei QRAM und Amplitude Encoding als Verfahren zur Zustandspräparation eingesetzt werden.

Anwendungsbeispiele: Iten et al. [IRM+21] präsentieren eine Implementierung für Schmidt Decomposition. In einer weiteren Arbeit wird das Kodierungsverfahren verwendet, um zufällige Quantenzustände zu erzeugen und diese kontrolliert zu verschränken [DGK14].

5.4 Kompilierung von Quantenschaltkreisen

Unter Berücksichtigung der Breite und Tiefe eines Quantenschaltkreises kann dessen Ausführbarkeit auf einem Quantencomputer bestimmt werden (Abschnitt 5.2) [CBS+19; LB20]. Dabei wird die Größe neben der zu lösenden Probleminstanz und dessen Kodierung (Abschnitt 5.3) [LB20; WBLS20] unter anderem auch von der Kompilierung auf den Quantencomputer bestimmt (Abschnitt 2.1.4) [ZPW19]. So sollten aufgrund der variierenden Kompilierungsmethoden [KIMK22] verschiedene Compiler für die Abbildung auf die Hardware in Betracht gezogen werden, um eine weitestgehend effiziente Kompilation für die Ausführung wählen zu können.

Hierfür werden mit Schritt G der MarieQurie-Methode in Abbildung 5.10 gegebene Schaltkreise mit den jeweiligen Quantencomputern als Kompilierungsziel an die zugeordneten Quantencompiler (Abschnitt 4.3) übergeben. Dabei befinden sich die Schaltkreise bereits in den Zielformaten der SDKs gewählter Quantencompiler (Abschnitt 4.4) und die Quantencomputer wer-

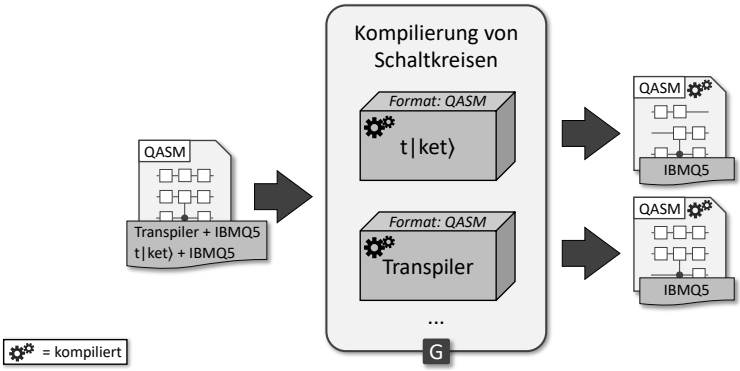

Abbildung 5.10: Kompilierung von Quantenschaltkreisen mit verschiedenen Quantencompilern (adaptiert von [SBL+21])

den nativ von dem jeweiligen Compiler unterstützt. Beispielhaft wird in Abbildung 5.10 ein Schaltkreis in *OpenQASM* und der Name eines IBMQ Quantencomputers sowohl an den *t|ket⟩ Compiler* [SDC+20] als auch an den *Qiskit Transpiler* [Qis23] übermittelt. Die Compiler bilden die Schaltkreise auf die Hardware der Quantencomputer ab (Abschnitt 2.1.4). Die resultierenden Kompilationen befinden sich für die Ausführung aufgrund der nativen Unterstützung bereits im nötigen Format mit den hardwarespezifischen Quantengattern der Quantencomputer (Abschnitt 4.3.2). Die beiden Compiler geben die Kompilationen im Format *OpenQASM* zurück, sodass diese auf dem Quantencomputer von IBMQ ausgeführt werden können.

Ist eine klassische Simulation der Schaltkreise möglich (Abschnitt 2.1.5), werden sie zusätzlich für einen Simulator kompiliert [SBLW22a; SBLW22b]. Die Simulatorergebnisse werden später verwendet, um die Präzision der Ausführungsergebnisse mittels HI für Vorhersagen und Priorisierungen in Abschnitt 4.3 und Kapitel 6 bestimmen zu können.

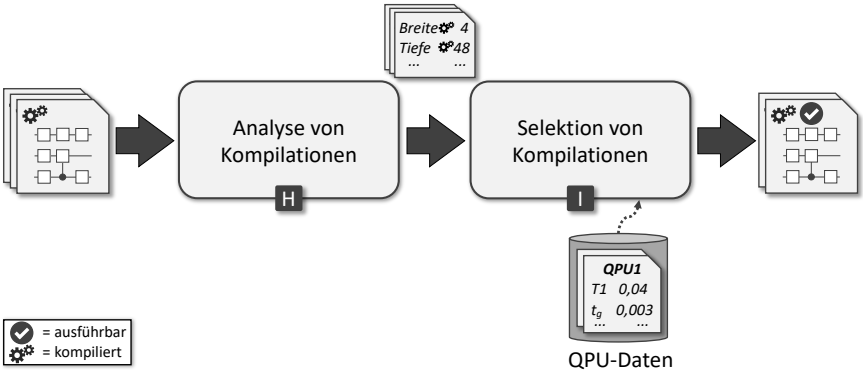

Abbildung 5.11: Selektion von ausführbaren Quantenkompilationen

5.5 Entscheiden der Ausführbarkeit von Quantenkompilationen

Um die Ausführbarkeit von kompilierten Schaltkreisen zu bestimmen, analysiert die MarieQurie-Methode mit Schritt H in Abbildung 5.11 die Metrikwerte (Abschnitt 4.3.1) dieser. Die Breite und Tiefe der Kompilationen werden schließlich an Schritt I übergeben. In diesem Schritt werden die Anzahl verfügbarer Qubits, die maximale Gatterzeit t_g sowie der Durchschnittswert von T_1 (Abschnitt 2.1.3) aller Qubits als T_q des jeweils zugehörigen Quantencomputers mittels QProv [WBL+21] abgerufen. Kompilationen werden verworfen, wenn deren Qubitzahl, die der Quantencomputer übertrifft, falls diese nicht bereits von den Compilern selbst abgelehnt wurden. Des Weiteren wird basierend auf der von Sete, Zeng und Rigetti [SZR16] definierten Formel (Abschnitt 2.1.6.1) die maximal ausführbare Tiefe eines Quantencomputers mithilfe von T_q und t_g geschätzt und mit der Tiefe d der jeweiligen Kompilation verglichen [SZR16]:

$$\frac{T_q}{t_g} \geq d \tag{5.6}$$

Erfüllen die Kompilationen zusätzlich die Bedingung von Gleichung (5.6), übertreffen sie weder die verfügbare Qubitzahl noch die geschätzte maximal ausführbare Tiefe des jeweiligen Quantencomputers und gelten somit als ausführbar. Diese Kompilationen werden an die Nutzenden zurückgegeben, sodass sie anhand der ermittelten Metrikwerte verglichen werden können.

5.6 Diskussion und Abgrenzung zu verwandten Arbeiten

Die vorgestellten Muster (Abschnitt 5.3) sind Teil der stetig größer werdenden Quantencomputingmustersprache, die mit der Arbeit von Leymann [Ley19] initiiert wurde (Abschnitt 2.4.1). Unter anderem beschreiben Leymann und Barzen [LB20] die vorgestellten Kodierungen sowie deren Realisierbarkeit auf NISQ-Rechner. Ihre Inhalte beruhen jedoch nicht auf dem Konzept des Musters von Alexander, Ishikawa und Silverstein [AIS77].

Abgesehen von den vorgestellten gibt es weitere Eingabedatenkodierungen für Quantenalgorithmen, unter anderem von Schuld und Petruccione [SP18] vorgestellt, die beispielsweise im Kontext von QML eingesetzt werden. Dabei kann die Kodierung klassischer Daten in einen Quantenzustand im QML auch als *Quantum Feature Map* verstanden werden [SP18; WBLS21b]. Für den HHL-Algorithmus präsentieren Duan et al. [DYY+20] Kodierungen, welche jedoch nicht allgemein als Muster beschrieben werden. Die Identifizierung weiterer Muster im Quantencomputing wird auch in Zukunft stattfinden.

Die Größe und Struktur einer Kompilation beeinflusst maßgeblich die Präzision späterer Ausführungen auf NISQ-Rechnern (Abschnitt 2.1.4). In der Arbeit von Kremer et al. [KVP+24] wird beispielsweise Reinforcement Learning verwendet, um die Kompilierung von Quantenschaltkreisen weitgehend zu optimieren. Umso wichtiger ist es, Kompilationen verschiedenster Compiler vergleichen zu können. Campbell et al. [CCD+23] und Sivarajah et al. [SDC+20] wie auch andere Arbeiten [AG20; MLM+19] vergleichen die Leistungsfähigkeit ihrer Compiler, mit der anderer anhand bestimmter Benchmarks. Auch in der Arbeit von Mills et al. [MSSD21] werden die Kompilierungsmethoden zweier Compiler miteinander verglichen. Diese

Arbeiten bieten jedoch kein automatisiertes Plug-in-basiertes Framework für den Vergleich von Kompilationen zur Lösung beliebiger Probleminstanzen. McCaskey et al. [MLD+20] ermöglichen die Kompilierung mit unterschiedlichen Compilern in ihrem Framework. Die Arbeit von Kharkov et al. [KIMK22] präsentiert ein automatisiertes Benchmarking-Framework für Compiler. Das Framework ermöglicht den Vergleich von Compilern verschiedener SDKs und wie in der Arbeit von Quetschlich, Burgholzer und Wille [QBW23a; QBW23c] erlaubt es die Verkettung von Kompilierungs- und Optimierungsmethoden unterschiedlicher Compiler. Sowohl McCaskey et al. [MLD+20] als auch Kharkov et al. [KIMK22] prüfen jedoch nicht die Ausführbarkeit der Kompilationen. Quetschlich, Burgholzer und Wille [QBW23b] stellen ein Benchmarking-Framework vor, das algorithmische Schaltkreise auf verschiedenen Abstraktionsebenen zurückgibt. Dabei können Nutzende unter anderem den Compiler und den Zielquantencomputer wählen. Jedoch werden die Schaltkreise nur in den nativen Formaten des Qiskit SDKs beschrieben.

In den Arbeiten von Quetschlich, Burgholzer und Wille [QBW23c; QBW23d] wird die Wahrscheinlichkeit für eine erfolgreiche Ausführung eines Schaltkreises durch die Multiplikation der Fidelitäten verwendeter Operationen geschätzt. *Estimated Probability of Successful Trial (EPST)* [LD21] schätzt den PST-Wert (Abschnitt 2.1.6.3) und somit die Zuverlässigkeit einer Schaltkreisausführung ebenfalls anhand der Fidelitäten der Operationen auf den Qubits des Zielquantencomputers. Kharkov et al. [KIMK22] schlagen eine Metrik vor, die einen Kostenfaktor anhand der Tiefe und der Gatterfidelitäten berechnet, wobei ein Bestrafungsfaktor für tiefe Schaltkreise hinzumultipliziert wird [BGT23; KIMK22]. Diese Metriken ermöglichen einen Vergleich zwischen den Kompilationen und könnten ebenfalls für die Empfehlung von Quantenressourcen in die MarieQurie-Methode integriert werden. Jedoch müssen für diese Grenzwerte bestimmt werden, um Kompilationen als ausführbar zu deklarieren und diese zu selektieren. Die in der vorliegenden Arbeit verwendete Gleichung (5.6) stellt aufgrund der hohen Fehlerkomplexität der NISQ-Rechner [BBC+17; MBB+18] nur eine grobe Schätzung dar, bietet allerdings einen solchen Ausführbarkeitsgrenzwert.

In der Arbeit von Ravi et al. [RSMC21] werden die Fidelitäten von Kompilationen für alle verfügbaren Quantencomputer durch ML basierend auf deren Tiefe, der Anzahl Zwei-Qubit-Gatter auf dem kritischen Pfad, der Zwei-Qubit-Gatterfehlerraten und Messfehlerraten erlernt. Gelernt wird anhand von Ausführungen auf simulierten Quantencomputern. In gleichem Verhältnis wird neben der Fidelität die geschätzte Wartezeit für die Ausführung auf dem Zielquantencomputer berücksichtigt und die unter diesen Voraussetzungen beste Kompilation automatisch selektiert und ausgeführt. Ob die Kompilation anhand eines Fidelitätsgrenzwertes erfolgreich ausführbar ist, wird nicht geprüft. Ruan et al. [RWJ+23] stellen die Eigenschaften von Kompilationen eines einzelnen Compilers und verschiedenen Quantencomputern visuell dar und ermöglichen diese miteinander zu vergleichen. Zusätzlich schätzen sie die Gesamtfehlerrate der Kompilationen. Sie sprechen jedoch keine Empfehlung von Kompilationen aus und prüfen diese nicht auf ihre Ausführbarkeit. Brandhofer, Devitt und Polian [BDP21] präsentieren ein Framework, welches die maximal zu akzeptierende Fehlerrate und die Erfolgswahrscheinlichkeit der Ausführung eines beliebigen Schaltkreises mittels Simulation berechnet und so Nutzende bei der Wahl geeigneter Quantenressourcen unterstützt. Aktuelle Metrikwerte diverser Quantencomputer werden jedoch nicht regelmäßig automatisiert abgerufen.

Kapitel 6

Präferenzbasiertes Priorisieren von Quantenkompilationen

Stehen für die Lösung einer Probleminstanz mehrere Quantenkompilationen zur Verfügung, müssen Nutzende entscheiden, welche dieser für eine Ausführung herangezogen werden soll. Hierbei müssen sie die Metrikwerte der Kompilationen und Quantencomputer sowie die Servicequalitäten der Hardwareanbieter wie Wartezeiten bis zur Ausführung berücksichtigen. Aufgrund der verschiedenen Metriken wird (i) deren Gewichtung anhand der Anforderungen Nutzender bestimmt (Abschnitt 6.2), (ii) die Kompilationen mithilfe der Gewichte priorisiert (Abschnitt 6.3) und (iii) die Ranglistenstabilität analysiert (Abschnitt 6.4). Gewählte Kompilationen werden schließlich automatisiert ausgeführt (Abschnitt 6.5). Es werden die Schritte 6 und 7 der MarieQurie-Methode und Forschungsbeitrag 4 präsentiert, welche im Rahmen dieser Arbeit veröffentlicht wurden [SBLW22a; SBLW22b].

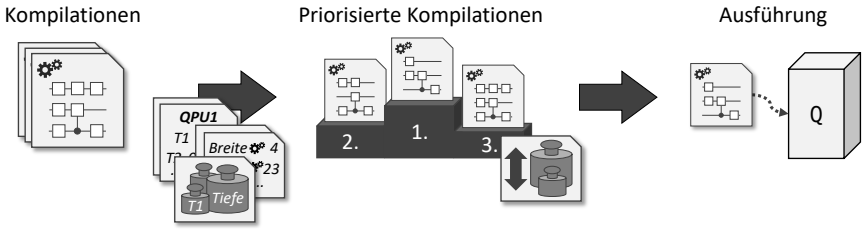

Abbildung 6.1: Vorgehen zur anforderungsorientierten Priorisierung von Quantenkompilationen

6.1 Idee und Grundkonzept

Bei Vorliegen mehrerer Quantenkompilationen priorisiert dieser Forschungsbeitrag diese automatisiert, wie in Abbildung 6.1 dargestellt. Priorisiert wird anhand der Anforderungen der Nutzenden. So wurde bereits in Abschnitt 4.3 die Ergebnispräzision der Eingangsschaltkreise prognostiziert. Mit der Untersuchung der Kompilationen ist durch die Hardwareabbildung jedoch ein noch präziseres Vergleichen dieser möglich. Für die Erstellung einer Rangliste betrachtet die MarieQurie-Methode neben den Schaltkreismetrikwerten der Kompilationen (Abschnitt 4.3.1) ebenso die Quantencomputermetrikwerte (Abschnitt 4.3.2), die ansonsten von den Nutzenden selbst gesammelt und untersucht werden müssen. Neben der Anforderung an präzise Ausführungsergebnisse können außerdem kurze Wartezeiten bis zur Ausführung bei der Priorisierung angestrebt werden. Entsprechend wird die Wichtigkeit der individuellen Metriken für die jeweiligen Anforderungen der Nutzenden anhand von Gewichten automatisiert bestimmt. Die Gewichte werden zur Berechnung der Rangliste der Kompilationen verwendet. Zusätzlich können Nutzende die Stabilität der Rangliste prüfen lassen, um die Sensibilität der einzelnen Platzierungen zu untersuchen. Die für die Nutzenden geeigneten Kompilationen können zudem nach Wahl automatisiert ausgeführt werden.

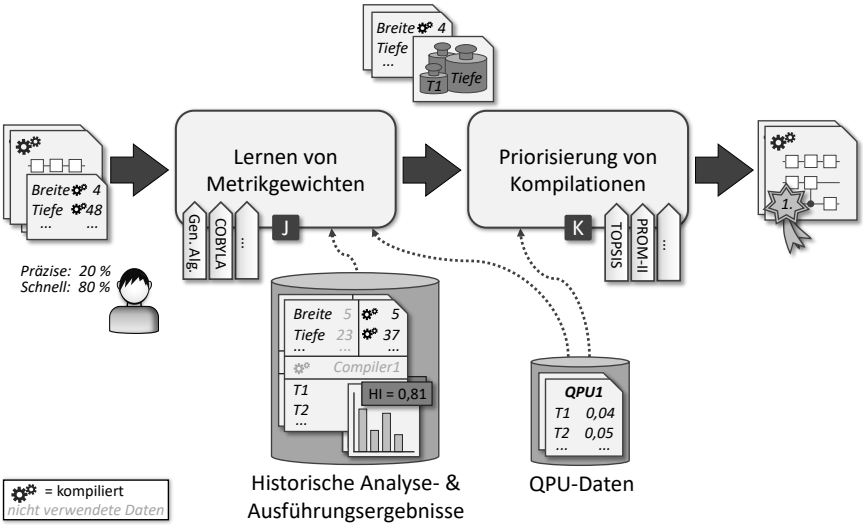

Abbildung 6.2: Berechnung von Metrikgewichten und Priorisierung von Quantenkompilationen (adaptiert von [SBLW22a; SBLW22b])

6.2 Gewichtung der Metriken von Quantenkompilationen und -computern

Die Metrikwerte der zu priorisierenden Kompilationen sind Eingabe für Schritt J, dargestellt in Abbildung 6.2. Nutzende geben ihre Anforderungen an präzise Ergebnisse und kurze Wartezeiten bis zur Ausführung in einem Verhältnis an. Zusätzlich werden die Metrikwerte der zu den Kompilationen zugehörigen Quantencomputer mit QProv [WBL+21] abgerufen. In Schritt J werden die Gewichte für die Metriken zur nachfolgenden Priorisierung der Kompilationen in Abhängigkeit zu den Anforderungen berechnet. Die Gewichte entsprechen dabei der Wichtigkeit der individuellen Metriken.

Haben Nutzende die Anforderung an kurze Wartezeiten bis zur Ausführung einer Kompilation, wird der Wartezeit das Maximalgewicht von 1 und den anderen Metriken das Minimalgewicht von 0 zugewiesen. Für die An-

forderung an präzisen Ausführungsergebnissen werden in Schritt J von Abbildung 6.2 die Gewichte der Metriken durch einen auswählbaren, mit der MarieQurie-Methode verfügbaren Optimierungsalgorithmus erlernt. Der Optimierungsalgorithmus verwendet die Metrikwerte von Kompilationen und Quantencomputern sowie zugehörige Werte des Histogrammschnitts (engl. „*Histogram Intersection (HI)*“) vergangener Ausführungen als Trainingsdaten. Sind keine Trainingsdaten vorhanden, wird keine automatisierte Erlernung der Gewichte sowie Priorisierung bezüglich präziser Ergebnisse angeboten. Für das Training wird eine auswählbare multikriteriellen Entscheidungsanalysemethode (MCDA-Methode) (engl. „*Multi-Criteria Decision Analyses method*“) verwendet, die später ebenfalls für die Priorisierung der aktuellen Kompilationen in Schritt K eingesetzt wird. Während des Trainings werden mittels der MCDA-Methode (Abschnitt 2.3.2) Ranglisten der historischen Kompilationen mit verschiedenen Gewichten berechnet [Alp16]. Die errechneten Ranglistenwerte der historischen Kompilationen werden mit den HI-Werten der damaligen Ausführungen verglichen. Das Ziel des Optimierungsalgorithmus ist, den häufig verwendeten *mittleren quadratischen Fehler (engl. „Mean Squared Error (MSE)“)* zwischen beiden zu minimieren, um geeignete Gewichte zu finden [GZL+21; JWHT21]. Die erlernten Gewichte und Metrikwerte der Kompilationen werden an Schritt K übergeben.

Ist die Anforderung ein Verhältnis zwischen präzisen Ergebnissen und kurzen Wartezeiten, werden zum einen die Gewichte mittels Optimierungsalgorithmus berechnet und zum anderen die Gewichte für kurze Wartezeiten gewählt. Beide Gewichtsvarianten werden an Schritt K übergeben. Alternativ können Nutzende in Schritt J selbst Gewichte nach der Methode SMART [Edw77] (Abschnitt 2.3.3) definieren.

6.3 Priorisierung von Quantenkompilationen

Auf die von Schritt J übergebenen Metrikwerte der Kompilationen, den ermittelten Gewichten und den Metrikwerten der Quantencomputer wird die zuvor ausgewählte MCDA-Methode angewendet. Bei der Anforderung

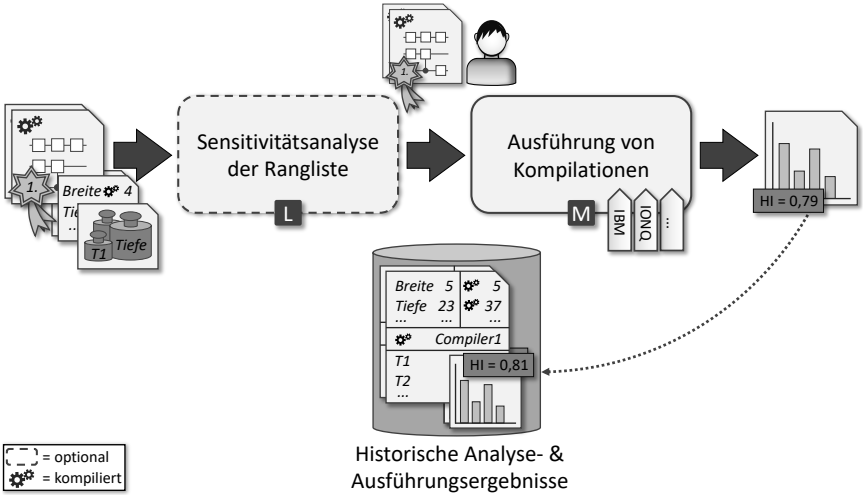

Abbildung 6.3: Sensitivitätsanalyse der Rangliste von Quantenkompilationen und deren Ausführung (adaptiert von [SBLW22a; SBLW22b])

nach kurzen Wartezeiten und präzisen Ergebnissen wird für beide Sätze an Gewichten jeweils einmal die MCDA-Methode appliziert. Die zwei resultierenden Ranglisten werden mittels Borda-Wahl [BKS+21; WJZZ09] unter Berücksichtigung des angegebenen Verhältnisses beider Anforderungen [Rus07; SBLW23] zu einer einzelnen vereint (Abschnitt 4.3.2). Schritt K gibt die Kompilationen in einer Rangliste an die Nutzenden zurück.

6.4 Sensitivitätsanalyse priorisierter Quantenkompilationen

Die dargestellte Rangliste der Kompilationen kann optional von den Nutzenden mit einer Sensitivitätsanalyse bezüglich der Stabilität der einzelnen Platzierungen mit den in Abschnitt 6.2 ermittelten Gewichten untersucht werden [GL14; LQWC13], siehe Schritt L in Abbildung 6.3. Bei der Sensitivitätsanalyse der MarieQurie-Methode wird jedes einzelne Gewicht individuell analysiert. Dabei wird das jeweilige Gewicht zunächst mit ei-

nem schrittweise zunehmenden γ multipliziert (Abschnitt 2.3.4) [LQWC13]. Damit die Summe aller Gewichte weiterhin den Wert 1 ergibt, werden die Gewichte der anderen Metriken entsprechend angepasst. Nach jedem Schritt wird untersucht, ob sich die Rangliste verändert hat oder eine obere Schranke von γ erreicht wurde. Trifft eines der beiden Bedingungen zu, wird der laufende Prozess angehalten und erneut umgekehrt mit einem schrittweise abnehmenden γ und einer unteren Schranke von γ durchgeführt. Resultierende Änderungen der Rangliste werden den Nutzenden präsentiert. Nutzende können für die Analyse nebst initial gegebener Werte sowohl die Schrittgröße als auch die obere und untere Schranke für γ anpassen.

Ist die zu untersuchende Rangliste eine mittels Borda-Wahl vereinte Rangliste, werden zunächst die durch den Optimierungsprozess meist empfindlichen Gewichte für die Präferenz präziser Ausführungsergebnisse mit γ multipliziert. Anschließend wird mit der Borda-Wahl die neu berechnete Rangliste mit der Rangliste für kurze Wartezeiten vereint (Abschnitt 4.3.2). Die Rangliste für kurze Wartezeiten bleibt aufgrund des unempfindlichen Maximalgewichts von 1 für die Metrik Wartezeit unverändert. Nutzenden werden Änderungen der vereinten Rangliste gezeigt.

6.5 Ausführung von Quantenkompilationen

Im letzten Schritt M können Nutzende wählen, welche der priorisierten Kompilationen auf den zugehörigen Quantencomputern automatisiert ausgeführt werden sollen, siehe Abbildung 6.3. Mit der Ausführung einer Kompilation auf einem Quantencomputer wird parallel die zugehörige Kompilation auf einem verfügbaren Simulator ausgeführt, wenn genügend Rechenressourcen vorhanden sind (Abschnitt 5.4). Ist eine Simulation nicht möglich, wird nur die Ausführung auf dem Quantencomputer durchgeführt. Liegen die Wahrscheinlichkeitsverteilungen der Messergebnisse des Quantencomputers und des Simulators vor, wird der HI-Wert berechnet (Abschnitt 2.1.5). Der HI-Wert, die Ergebnisse und die ermittelten Metrikwerte der Schaltkreise, Kompilationen und Quantencomputer (Abschnitte 4.3 und 5.5) werden

gespeichert und den Nutzenden zurückgegeben. Diese Daten dienen wiederum zur Verfeinerung der Vorhersagen geeigneter Quantenressourcen (Abschnitt 4.3.2) und der Gewichtsbestimmung der Metriken (Abschnitt 6.2) zukünftiger MarieQurie-Methodenaufrufe. Wurde der HI-Wert nicht berechnet, werden diese Ausführungsergebnisse und zugehörige Daten zukünftig ignoriert. Die Implementierungen, aus denen die Schaltkreise der ausgeführten Kompilationen extrahiert wurden (Abschnitt 4.2.2), werden erneut aufgerufen und mit der Übergabe der Ausführungsergebnisse fortgesetzt, sodass beispielsweise Messfehlerminderungstechniken durchgeführt werden können [BBL+22a; BBL+22b; BSK+21; WBLV22].

6.6 Diskussion und Abgrenzung zu verwandten Arbeiten

Die in diesem Kapitel vorgestellten Konzepte ermöglichen die Priorisierung von Kompilationen anhand der Präferenzen der Nutzenden. Durch die Erweiterbarkeit der MarieQurie-Methode können diverse Optimierungsalgorithmen und MCDA-Methoden angebunden werden. Des Weiteren bedarf die in Abschnitt 6.5 angewendete Präzisionsbestimmung mittels HI die Simulation der Schaltkreise, um fehlerfreie Ausführungsergebnisse dieser zu erhalten und mit denen der Quantencomputer vergleichen zu können. Dieses Verfahren zur Bestimmung von korrekten Ergebnissen und Fehlereinflüssen ist anwendbar, solange die Anzahl an benötigten Qubits und Quantengattern klassisch simuliert werden kann [CZX+18]. Eine Alternative, um den Einfluss von Fehlern ohne Simulation untersuchen und Lerndaten bereitstellen zu können, ist, das Inverse eines Schaltkreises an diesen anzuhängen, sodass für den adaptierten Schaltkreis der initiale Quantenzustand das Soll-Ergebnis darstellt [MHP+23]. Abweichende Messungen zeigen schließlich die einfließenden Fehler und können zur Präzisionsbestimmung und -prognose herangezogen werden. Durch die Erweiterbarkeit der MarieQurie-Methode ist die Anbindung alternativer Verfahren zur Verifikation von Rechenergebnissen ohne Simulation, wie beispielsweise von Barz et al. [BFKW13] und Gheorghiu, Kapourniotis und Kashefi [GKK19], möglich.

Abschnitt 5.6 diskutiert bereits die Arbeit von Ravi et al. [RSMC21]. Sie selektieren zwar automatisiert die geeignetste Kompilation zur Ausführung auf dem Zielquantencomputer anhand vorhergesagter Fidelitäten und Wartezeiten, berücksichtigen jedoch nicht die Präferenzen von Nutzenden und ermöglicht den Nutzenden nicht, die Selektion des Quantencomputers beispielsweise durch eine Sensitivitätsanalyse hinterfragen zu können.

Garcia-Alonso et al. [GRV+22] präsentieren ein Konzept, welches den geeignetsten Quantencomputer automatisiert entweder anhand der geschätzten Wartezeiten oder der geschätzten Kosten selektiert. Um die Ausführbarkeit zu prüfen, wird ausschließlich die Anzahl benötigter Qubits der verfügbaren Quantencomputer verglichen. Sie ermöglichen Nutzenden nicht, ein Verhältnis zwischen den Prioritäten zu definieren. Grossi et al. [GCA+21] wählen den Quantencomputer zur Ausführung eines Schaltkreises automatisiert anhand der Anzahl verfügbarer Qubits und der geringsten anzunehmenden Wartezeit. Sie berücksichtigen keine weiteren Schaltkreis- und Quantencomputermetriken und beziehen keine Anforderungen von Nutzenden ein.

Die Arbeit von Bhoumik et al. [BMSS23] schneidet Quantenschaltkreise in Unterschaltkreise und verteilt diese automatisiert auf Quantencomputer, sodass die Fidelität möglichst maximal ist und die Rechenzeit unter einem pro Quantencomputer geschätzten Grenzwert liegt. Für die Entscheidung, welcher Unterschaltkreis auf welchem Quantencomputer ausgeführt wird, prüfen sie, ob der Quantencomputer genügend Qubits bereitstellt. Zudem schätzen sie auftretende Fehler mit einem Werkzeug [NT23] anhand der Größe des Schaltkreises auf dem Zielquantencomputer und der Hardwareeigenschaften wie Konnektivität, Dekohärenzzeiten, Mess- und Gatterfehler, dargestellt durch einen Punktestand. Anhand der Punktestände werden die Kombinationen aller Unterschaltkreise mit Quantencomputern absteigend sortiert und dem Optimierungsalgorithmus mit den maximalen Ausführungszeiten zur Selektion übergeben [BMSS23]. Servicequalitäten der Hardwareanbieter wie Wartezeiten werden dabei nicht berücksichtigt.

Kapitel 7

Architektur und Prototyp des MarieQurie-Frameworks

Dieses Kapitel beschreibt mit Forschungsbeitrag 5 die Architektur und prototypische Implementierung der gezeigten Konzepte vorheriger Kapitel zur automatisierten Realisierung der MarieQurie-Methode. Mit diesem Beitrag wird zunächst die Gesamtarchitektur des MarieQurie-Frameworks präsentiert und anschließend dessen einzelne Softwarekomponenten im Detail beschrieben (Abschnitt 7.1). Außerdem wird der Prototyp als Umsetzung der Methode vorgestellt (Abschnitt 7.2). Zuletzt werden beispielhaft anhand eines Anwendungsszenarios geeignete Quantenressourcen für einen Quantenalgorithmus mithilfe des Prototyps empfohlen (Abschnitt 7.3). Die Architektur und der Prototyp wurden bereits im Rahmen der vorliegenden Arbeit publiziert [SBB+20; SBL+21; SBLW22a; SBLW22b; SBLW23].

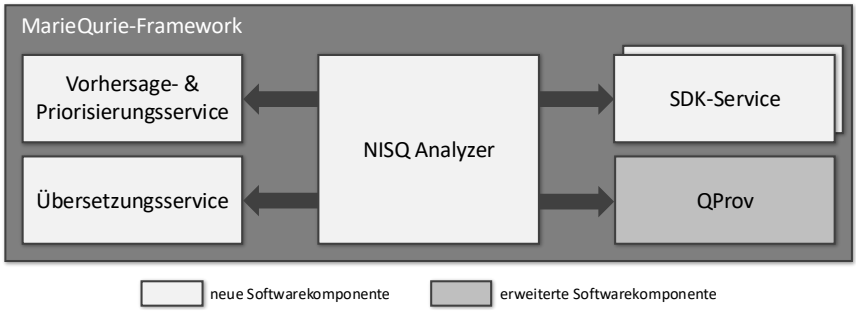

Abbildung 7.1: Gesamtarchitektur des MarieQurie-Frameworks

7.1 Architektur des MarieQurie-Frameworks

Die Gesamtarchitektur des MarieQurie-Frameworks, das die Konzepte der MarieQurie-Methode umsetzt, ist in Abbildung 7.1 dargestellt. Oben links wird der *Vorhersage- & Priorisierungsservice* gezeigt. Dieser setzt die Konzepte zur Vorhersage geeigneter Quantenressourcenkombinationen (Abschnitt 4.3) und zur Priorisierung und Analyse der Ranglisten ausführbarer Kompilationen (Kapitel 6) basierend auf den Präferenzen der Nutzenden um. Darunter befindet sich der *Übersetzungsservice*. Dieser übersetzt Schaltkreise in die geforderten Formate der Zielcompiler (Abschnitt 4.4). Rechts oben befinden sich die *SDK-Services*, welche die Funktionalitäten der integrierten SDKs zur Ausführung der Implementierungen, inklusive ihrer Vor- und Nachbearbeitungsprogramme, Analyse von Schaltkreisen, Kompilierung mit eingebundenen Compilern und Ausführung auf den Zielquantencomputern uniform bereitstellen. Darunter ist die bereits existierende Softwarekomponente *QProv* [WBL+21]. QProv stellt uniform aktuelle Hardware- und Servicequalitätsmetrikwerten wie Wartezeiten verfügbarer Quantencomputer diverser Anbieter über eine REST API[1] bereit. Die Softwarekomponente wurde im Kontext dieser Arbeit um die Quantencomputermetriken

[1] `https://github.com/UST-QuAntiL/qprov/blob/master/docs/api/api-docs.json`

durchschnittliche Ein-Qubit-Gatterfehlerrate und Ein-Qubit-Gatterzeit (Abschnitt 4.3.1) und den Cloudanbieter Amazon Braket ergänzt. QProv ist nicht Fokus dieser Arbeit, Details können bei Weder et al. [WBL+21] und Weder [Wed24] nachgeschlagen werden. Alle aufgeführten Softwarekomponenten werden über den *NISQ Analyzer* aufgerufen, siehe Mitte von Abbildung 7.1.

Der NISQ Analyzer ist das Zentrum des Frameworks. Die Softwarekomponente nimmt die Anfragen der Nutzenden entgegen und selektiert Algorithmusimplementierungen. Sie ruft den Vorhersage- & Priorisierungsservice auf, um geeignete Quantenressourcenkombinationen zu selektieren und die resultierenden Kompilationen zu priorisieren. Für die Übersetzung von Schaltkreisen ruft der NISQ Analyzer den Übersetzungsservice auf. Für die Ausführung der Implementierungen, die Kompilierung der Schaltkreise und die Ausführung der Kompilationen verwendet der NISQ Analyzer die entsprechenden SDK-Services. Außerdem analysiert der NISQ Analyzer die Ausführbarkeit der Kompilationen (Kapitel 5) und ruft zur Verwendung aktueller Quantencomputermetrikwerte QProv auf.

7.1.1 NISQ Analyzer

Abbildung 7.2 zeigt die Architektur des NISQ Analyzers. Über die grafische Nutzeroberfläche (engl. „*Graphical User Interface (GUI)*“) können Nutzende die MarieQurie-Methode aufrufen. Mit der GUI können sie zu lösende Probleminstanz und Präferenzen zu präzisen Ergebnissen und kurzen Wartezeiten bis zur Ausführung übergeben sowie aus Quantenalgorithmen und ausführbaren Kompilationen wählen. Außerdem zeigt die GUI analysierte Metrikwerte, berechnete Gewichte und Ranglisten sowie Sensitivitätsanalyse- und Ausführungsergebnisse an. Die Eingaben der Nutzenden werden von der GUI über eine REST API[1] an den NISQ Analyzer überliefert.

Für die Aufrufe durch den *Selektor* und den *Priorisierer* auf externe Softwarekomponenten abstrahiert der *Konnektor* des NISQ Analyzers über die

[1] `https://github.com/UST-QuAntiL/nisq-analyzer/blob/master/docs/api/openapi.json`

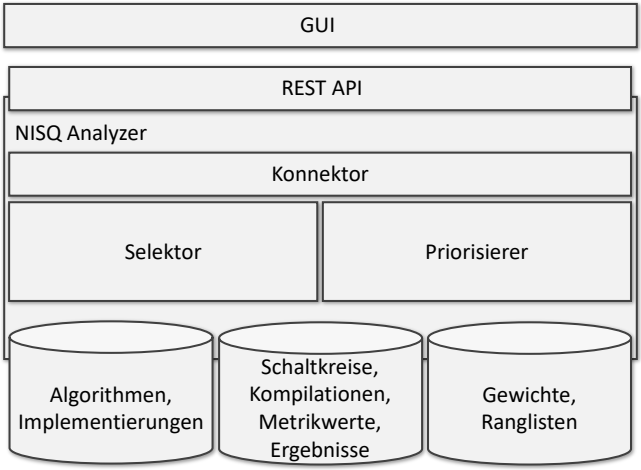

Abbildung 7.2: Architektur des NISQ Analyzers (adaptiert von [SBLW23])

verschiedenen REST APIs dieser Softwarekomponenten hinweg. Der *Selektor* lädt für den gewählten Quantenalgorithmus alle zugehörigen Implementierungen (Abschnitt 4.2) und übergibt die Implementierungen per Konnektor an die SDK-Services, die die vorliegenden Formate unterstützen. Als Antwort erhält der Selektor die extrahierten Schaltkreise und die Metrikwerte dieser. Zudem ruft der Selektor aktuell verfügbare Quantencomputer und deren Metrikwerte via QProv ab. Die Metrikwerte werden mit denen zuvor ausgeführter Schaltkreise, deren zugehörigen Compilern, Quantencomputern und Hardwaremetriken sowie HI-Werten inklusive Präferenzen und der bei Anforderung von präzisen Ausführungsergebnissen zu verwendende ML-Algorithmus an den Vorhersage- & Priorisierungsservice zur Quantenressourcenselektion übergeben. Der Selektor überprüft, ob die Schaltkreise der selektierten Kombinationen den Formaten entsprechen, die mit den SDKs der zugehörigen Compiler kompatibel sind. Bei Inkompatibilität wird dem Übersetzungsservice der jeweilige Schaltkreis, dessen Format sowie das geforderte Format übermittelt (Abschnitt 4.4). Die gegebenenfalls übersetzten Schaltkreise der selektierten Kombinationen werden mit ihren zugeordneten

Compilern und Quantencomputern jeweils an den SDK-Service des Compilers zur Kompilierung und Analyse transferiert (Abschnitt 5.4). Der Selektor selektiert ausführbare Kompilationen anhand deren Metrikwerte und der der zugehörigen Quantencomputer (Abschnitt 5.5). Auszuführende Kompilationen werden jeweils unter Angabe der Zielquantencomputer an die zugehörigen SDK-Services übergeben. Außerdem wird für die Berechnung des HI-Werts jeweils eine Kompilation desselben initialen Schaltkreises für den Simulator zur Ausführung übergeben (Abschnitt 6.5). Mit Erhalt der Ausführungsergebnisse wird der HI-Wert durch den Selektor für zueinandergehörende Quantencomputer-Simulator-Paare berechnet.

Der *Priorisierer* extrahiert die Metrikwerte in der Vergangenheit ausgeführter Kompilationen und Quantencomputer sowie HI-Werte. Ebenso werden die Metrikwerte der aktuellen Kompilationen und der zugehörigen Quantencomputer abgerufen. Die extrahierten Daten werden mit dem zu verwendenden Optimierungsalgorithmus, der MCDA-Methode und den definierten Anforderungen der nutzenden Person an den Vorhersage- & Priorisierungsservice zur Erlernung von Gewichten und Berechnung von Ranglisten überliefert (Kapitel 6). Nutzende können für eine erneute Priorisierung selbst Gewichte definieren oder bestehende anpassen, wodurch bisherige überschrieben werden und mittels Priorisierer und Vorhersage- & Priorisierungsservice erneut eine Rangliste berechnet wird. Für die Sensitivitätsanalyse ruft der Priorisierer Gewichte, aktuelle Metrikwerte und die Rangliste ab. Der Vorhersage- & Priorisierungsservice wird mit den bereitgestellten Daten und dem definierten Wertebereich für γ aufgerufen (Abschnitt 6.4).

Die Quantenalgorithmen sowie verfügbare Implementierungen dieser sind in einem Repositorium gespeichert. Extrahierte und gegebenenfalls übersetzte Quantenschaltkreise, selektierte Kompilationen, deren Metrikwerte sowie Quantencomputermetrikwerte und Ausführungsergebnisse sind ebenfalls in einem Repositorium persistiert, wie in Abbildung 7.2 dargestellt. Zudem sind sowohl die von den ML-Algorithmen prognostizierten als auch die berechneten HI-Werte in diesem Repositorium gespeichert. Zuletzt berechnete Gewichte und Ranglisten sind ebenfalls in einem Repositorium abgelegt.

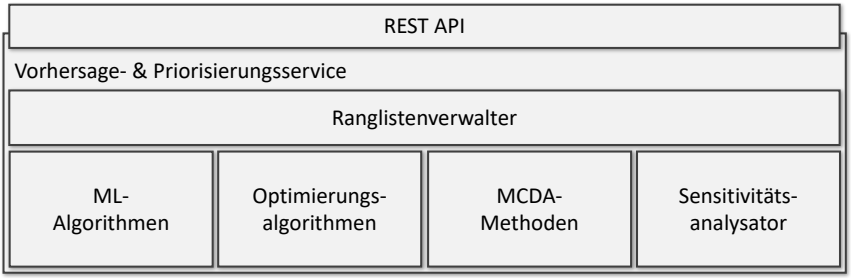

Abbildung 7.3: Architektur des Vorhersage- & Priorisierungsservices (adaptiert von [SBLW22b; SBLW23])

7.1.2 Vorhersage- & Priorisierungsservice

Die Architektur des Vorhersage- & Priorisierungsservices ist in Abbildung 7.3 abgebildet. Dieser wird für die Selektion geeigneter Quantenressourcen (Abschnitt 4.3) über die REST API[1] aufgerufen. Der Service empfängt den Namen des auszuführenden *ML-Algorithmus*, die Metrikwerte der aktuell betrachteten Schaltkreise und Quantencomputer, die verfügbaren Compiler, die Metrikwerte vergangener Ausführungen sowie die Präferenzen der Nutzenden. Die Präferenzen enthalten zum einen die Anforderung an kurzen Wartezeiten, präzisen Ausführungsergebnissen oder einem Verhältnis aus beiden. Zum anderen enthalten die Präferenzen den gesetzten Grenzwert, wie viele Kombinationen aus aktuellen Schaltkreisen, Compilern und Quantencomputern maximal von der MarieQurie-Methode zurückgegeben werden sollen. Werden ausschließlich kurze Wartezeiten bis zur Ausführung von den Nutzenden angestrebt, erzeugt der *Ranglistenverwalter* zunächst alle validen Kombinationen (Abschnitt 4.3). Die Liste der Kombinationen wird durch den Ranglistenverwalter nach Wartezeiten sortiert und nach übergebenen Grenzwert gekürzt. Für den Fall, dass die alleinige Anforderung präzise Ausführungsergebnisse sind, werden dem gewählten ML-Algorithmus die historischen und die aktuellen Metrikwerte sowie die HI-Werte vergange-

[1] `https://github.com/UST-QuAntiL/nisq-analyzer-prio-service/blob/master/docs/api/openapi.json`

ner Ausführungen übergeben, um HI-Werte der gewählten Kombinationen zukünftiger Ausführungen zu schätzen. Der Ranglistenverwalter sortiert anhand der HI-Werte und kürzt ebenfalls anhand des Grenzwerts. Für das Szenario, dass sowohl kurze Wartezeiten als auch präzise Ergebnisse im Verhältnis angestrebt werden sollen, werden die jeweiligen Listen unabhängig voneinander berechnet und schließlich vom Ranglistenverwalter mittels gewichteter Borda-Wahl [Rus07] (Abschnitt 4.3.2) vereint, gekürzt und an die Aufrufenden zurückgegeben.

Wird der Vorhersage- & Priorisierungsservice zur Priorisierung ausführbarer Kompilationen aufgerufen, werden ihm in der Vergangenheit ausgeführte Kompilationen, zugehörige Metrikwerte und HI-Werte sowie der Name der auszuführenden *MCDA-Methode* übermittelt. Für eine Priorisierung basierend auf der Präferenz von präzisen Ergebnissen wird zusätzlich noch der auszuführenden *Optimierungsalgorithmus* transferiert. Im Fall von präzisen Ergebnissen berechnet der Optimierungsalgorithmus die Gewichte, indem dieser abwechselnd wiederholt die MCDA-Methode auf den historischen Daten aufruft und die Gewichte entsprechend anpasst (Abschnitt 6.2). Für die Anforderung nach kurzen Wartezeiten bis zur Ausführung setzt ein weiterer Optimierungsalgorithmus das Maximalgewicht für die entsprechende Metrik, alle anderen Hardware- und Schaltkreismetriken erhalten das Minimalgewicht. Die berechneten Gewichte werden inklusive der aktuellen Metrikwerte an die MCDA-Methode zur Berechnung der Rangliste gegeben (Abschnitt 6.3). Ist ein Verhältnis beider Präferenzen gewünscht, werden beide Ranglisten unabhängig voneinander berechnet, durch den Ranglistenverwalter vereint und diese inklusive der verwendeten Gewichte an die Aufrufenden zurückgeschickt. Möchten Nutzende die Priorisierung erneut mit eigenen oder adaptierten Gewichten durchführen, werden diese zusätzlich übergeben und die selektierte MCDA-Methode wird ausgeführt.

Zur Sensitivitätsanalyse (Abschnitt 6.4) einer Rangliste werden die aktuellen Metrikwerte, Gewichte und MCDA-Methode sowie der Wertebereich von γ an den *Sensitivitätsanalysator* übergeben. Dieser ruft die MCDA-Methode wiederholt mit den durch γ schrittweise veränderten Gewichte auf und

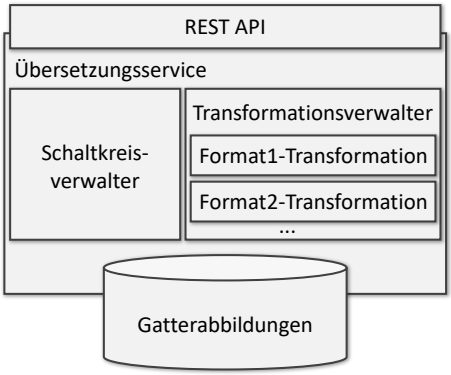

Abbildung 7.4: Architektur des Übersetzungsservices (adaptiert von [SBL+21])

stoppt, sobald eine Änderung in der ursprünglichen Rangliste auftritt. Die Analyseergebnisse werden an die Aufrufenden zurückgegeben.

7.1.3 Übersetzungsservice

Der Übersetzungsservice erhält den zu übersetzenden Quantenschaltkreis, dessen Format sowie das Zielformat über seine REST API.[1] Die Architektur des Services ist in Abbildung 7.4 dargestellt. Der *Schaltkreisverwalter* der den Import, die Transformation und den Export von Schaltkreisen in verschiedene Formate steuert, ruft den *Transformationsverwalter* für die Transformation auf. Bei der Transformation wird die Syntax des Eingangsformats in das Zielformat mittels Zwischenformat umgewandelt (Abschnitt 4.4). Dabei werden nicht-unterstützte Gatter durch verfügbare ersetzt. Für jedes unterstützte Format gibt es ein Formattransformations-Plug-In, welches von diesem Format in das Zwischenformat und umgekehrt Schaltkreise transformieren kann. Der Transformationsverwalter ruft beispielsweise zur Übersetzung eines Schaltkreises von *Format1* in *Format2* die *Format1-Transformation* auf,

[1] `https://github.com/UST-QuAntiL/QuantumTranspiler/blob/master/docs/openapi.json`

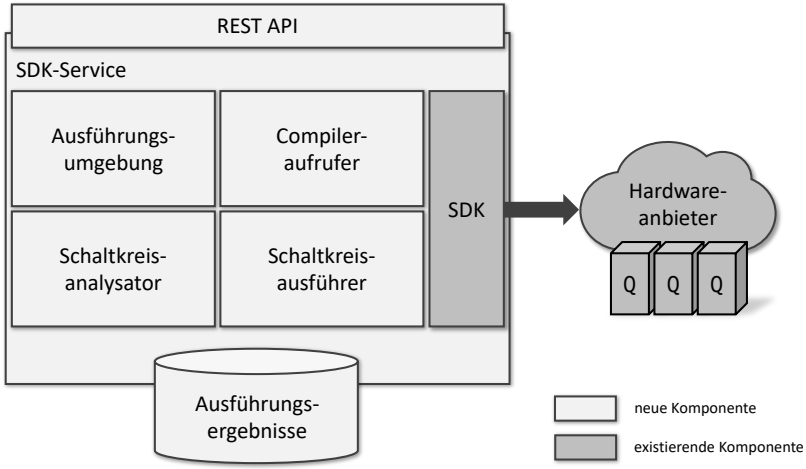

Abbildung 7.5: Architektur der SDK-Services (adaptiert von [SBB+20; SBL+21])

welche zunächst in das Zwischenformat übersetzt. Anschließend wird der sich im Zwischenformat befindliche Schaltkreis zur Übersetzung in das *Format2* der *Format2-Transformation* übergeben. Der übersetzte Schaltkreis wird zurückgegeben. Die Gatterabbildungen sind persistiert.

7.1.4 SDK-Services

Für die Analyse und Extraktion des Quantenschaltkreises einer Quantenalgorithmusimplementierung mit der zu lösenden Probleminstanz wird der zur Implementierung zugehörige SDK-Service über dessen REST API[1,2,3] aufgerufen. Die Architektur der SDK-Services ist in Abbildung 7.5 zu sehen. Der SDK-Service übergibt der Implementierung die Probleminstanz

[1] `https://github.com/UST-QuAntiL/qiskit-service/blob/master/docs/api/openapi.json`

[2] `https://github.com/UST-QuAntiL/forest-service/blob/main/docs/openapi.json`

[3] `https://github.com/UST-QuAntiL/pytket-service/blob/main/docs/openapi.json`

und führt sie in der *Ausführungsumgebung* mithilfe des ummantelten SDKs aus. Die Implementierung generiert und extrahiert den Quantenschaltkreis (Abschnitt 4.2.2). Der *Schaltkreisanalysator* analysiert den Schaltkreis anhand der Metriken aus Abschnitt 4.3.1. Die Metrikwerte und der Schaltkreis werden an die Aufrufenden geschickt. Wird zur Kompilierung aufgerufen, empfängt der *Compileraufrufer* einen Schaltkreis sowie den Namen des Zielquantencomputers. Die Informationen übergibt der Compileraufrufer dem *SDK*-internen Compiler, der eine Kompilation zurückgibt (Abschnitt 5.4). Die Kompilation wird mit dem Schaltkreisanalysator anhand selber Metriken analysiert (Abschnitt 5.5). Die Kompilation und dessen Metrikwerte werden an die Aufrufenden transferiert. Für die Ausführung einer Kompilation übergeben Aufrufende die Kompilation mit Zielquantencomputer an den entsprechenden SDK-Service (Abschnitt 6.5). Der *Schaltkreisausführer* ruft die bereitgestellte Ausführungsfunktion des SDKs auf, sodass die Kompilation an den Hardwareanbieter des Zielquantencomputers übergeben wird. Der Ausführer wartet auf die Messergebnisse und übergibt sie an die Implementierung in der Ausführungsumgebung, sodass diese weitere Berechnungen wie beispielsweise Messfehlerminderungstechniken [BBL+22a; BBL+22b; BSK+21; WBLV22] durchführen kann. Die Ausführungsergebnisse werden gespeichert, sodass die Aufrufenden sie abrufen können.

7.2 Prototypische Implementierung des MarieQurie-Frameworks

Im Folgenden wird die prototypische Umsetzung der in Abschnitt 7.1 dargestellten Softwarekomponenten des MarieQurie-Frameworks[1] behandelt. Alle Softwarekomponenten sind Open-Source unter der Apache 2.0 Lizenz veröffentlicht und kommunizieren via HTTP miteinander.

[1] `https://github.com/UST-QuAntiL/nisq-analyzer-content/tree/master/Docker-latest`

Softwarekomponente	GitHub-Repositorium
NISQ Analyzer	`https://github.com/UST-QuAntiL/nisq-analyzer`
QProv	`https://github.com/UST-QuAntiL/qprov`
GUI	`https://github.com/UST-QuAntiL/qc-atlas-ui`
Vorhersage- & Priorisierungsservice	`https://github.com/UST-QuAntiL/nisq-analyzer-prio-service`
Übersetzungsservice	`https://github.com/UST-QuAntiL/QuantumTranspiler`
Qiskit-Service	`https://github.com/UST-QuAntiL/qiskit-service`
Forest-Service	`https://github.com/UST-QuAntiL/forest-service`
Pytket-Service	`https://github.com/UST-QuAntiL/pytket-service`

Tabelle 7.1: Links zu Dokumentationen und Implementierungen der Softwarekomponenten des MARIEQURIE-Frameworks

7.2.1 NISQ Analyzer

Der NISQ Analyzer wurde, wie auch QProv in der Programmiersprache Java mit dem Spring Boot Framework entwickelt (Tabelle 7.1). Die GUI wurde in TypeScript mit dem Framework Angular programmiert. Gespeicherte Daten wie Algorithmen, Implementierungen und Ergebnisse sind in einer relationalen Datenbank von PostgreSQL gespeichert.

7.2.2 Vorhersage- & Priorisierungsservice

Der Vorhersage- & Priorisierungsservice (Tabelle 7.1) ist in Python mit Flask implementiert. Für die Umsetzung der ML-Algorithmen (Abschnitt 7.1.2)

wird die Pythonbibliothek *scikit-learn*[1] verwendet [PVG+11]. Der Prototyp unterstützt die basierend auf Kapitel 8 geeignetsten Implementierungen der ML-Algorithmen *Multiple Lineare Regression*, *SVR*, *KNNR* und *Entscheidungsbäume* (Abschnitt 2.3.1). Diese sind *DecisionTreeRegressor (DTR)* als Implementierung einfacher Entscheidungsbäume und als Variationen mehrerer kombinierter Entscheidungsbäume *GradientBoostingRegressor (GBR)*, *ExtraTreesRegressor (ETR)*, *HistGradientBoostingRegressor (HGBR)* und *RandomForestRegressor (RFR)*. *KNeighborsRegressor (KNNR)* implementiert KNNR und *NuSVR* setzt SVR um. Außerdem wird als Umsetzung für Multiple Lineare Regression *TheilSenRegressor (TSR)* unterstützt [SBLW23]. Die Implementierungen können einzeln oder jeweils in Kombination mit den Meta-Schätzern *AdaBoostRegressor (ABR)* [FS97] und *BaggingRegressor (BR)* [Bre96] verwendet werden. AdaBoostRegressor führt den jeweiligen ML-Algorithmus wiederholt auf den vorliegenden Daten aus und adaptiert die Parameter anhand der berechneten Schätzungen [FS97]. BaggingRegressor wiederholt ebenfalls die Ausführung des ML-Algorithmus, führt diesen jedoch auf Untermengen der gegebenen Datenmenge aus [Bre96]. Die finale Schätzung entsteht durch Zusammenführen der berechneten Schätzungen. Um die Compiler bei der Vorhersage präziser Ergebnisse zu berücksichtigen, werden sie mittels *OneHotEncoder* von scikit-learn kodiert (Abschnitt 2.3.1). Die Berechnung mit Entscheidungsbäumen ist aufgrund der nachgewiesenen Performanz in Kapitel 8 voreingestellt.

Für die Priorisierung von Kompilationen (Kapitel 6) wird die Pythonbibliothek *pymcdm*[2] verwendet. Die Bibliothek stellt die MCDA-Methoden TOPSIS [HY81] und PROMETHEE-II [BM05] zur Verfügung. Als Optimierungsalgorithmen werden der Genetische Algorithmus [Hol73] beruhend auf der Arbeit von Hassan und Hamada [HH18], Evolutionsstrategie basierend auf der Arbeit von Fogel [Fog94] und COBLYA [Pow94], umgesetzt mit der Pythonbibliothek *SciPy*,[3] unterstützt. Standardmäßig wird PROME-

[1] `https://scikit-learn.org/stable/`
[2] `https://pypi.org/project/pymcdm/`
[3] `https://docs.scipy.org/doc/scipy/reference/optimize.minimize-cobyla.html`

THEE II in Kombination mit dem Genetischen Algorithmus zur Berechnung der Gewichte und für die darauffolgende Priorisierung verwendet, da diese in Kapitel 8 gute Ergebnisse bei der Berechnung von Rangordnungen zeigen. Basierend auf der Arbeit von Li et al. [LQWC13] ist das Vorgehen der Sensitivitätsanalyse programmiert.

7.2.3 Übersetzungsservice

Für die Übersetzung der Schaltkreise (Abschnitt 4.4) verwendet der in Python mit Flask geschriebene Übersetzungsservice (Tabelle 7.1) Qiskits *QuantumCircuit*[1] als Zwischenformat. Mit diesem Zwischenformat ist Import als auch Export in das Format OpenQASM sowie der Import in Python für Qiskit gegeben. Des Weiteren bietet Qiskit nativ eine Vielzahl an Quantengattern und ermöglicht die Definition eigener Gatter. Da der Export von QuantumCircuits in Python für Qiskit nicht nativ unterstützt wird, ist zusätzliche Funktionalität im Übersetzungsservice dafür programmiert. Zusätzlich unterstützt der Übersetzungsservice den Import und Export der Formate Quil, PyQuil, Python für Amazon Braket sowie Python und JSON für Cirq.

Wird dem Übersetzungsservice beispielsweise ein Schaltkreis in PyQuil übergeben, wird zunächst die PyQuil-Transformation aufgerufen. Die PyQuil-Transformation verwendet das zugehörige SDK Forest [Rig21], um den Schaltkreis in das Forest-spezifische *Program*[2] Format zu transformieren. Für die Erzeugung eines äquivalenten QuantumCircuits betrachtet die PyQuil-Transformation nacheinander schrittweise die Instruktionen des Programs und bildet diese in der Syntax des QuantumCircuits nach. Entsprechend entgegengesetzt wird der Schaltkreis vom Zwischenformat in ein anderes Format transformiert. Die Abbildungen der Quantengatter auf äquivalente Gatter definierte Matrizen oder Mengen an ersetzenden Gattern für die verschiedenen Formate sind in einer Pythondatei persistiert.

[1] `https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.html`

[2] `https://pyquil-docs.rigetti.com/en/v2.25.0/apidocs/program.html`

7.2.4 SDK-Service

Umsetzungen der SDK-Services (Abschnitt 7.1.4) existieren für die SDKs Qiskit [Qis23], Forest [Rig21] und pytket [SDC+20] (Tabelle 7.1) und sind jeweils in Python mit dem Framework Flask implementiert. Jedes SDK stellt seine eigene Pythonbibliothek zur Implementierungsausführung, Schaltkreisanalyse, -kompilierung und -ausführung bereit. Deren unterstützte Hardwareanbieter sind in Abschnitt 2.2.2 zu sehen. Hardwareanbieter wie beispielsweise IBMQ reihen auszuführende Kompilationen, die in Jobs gepackt werden, in der Regel in Warteschlangen ein [LaR19; VBL+21], wodurch die Messergebnisse asynchron zurückerhalten werden. Um auch das MarieQurie-Framework bei Ausführung nicht blockieren zu lassen und stattdessen von der Ausführung eines einzelnen Schaltkreises zu entkoppeln, verwenden die SDK-Services selbst ebenfalls Warteschlangensysteme, umgesetzt mit Redis Queue.[1] Auszuführende Jobs werden in einem SDK-Service in eine Warteschlange abgelegt, wobei mehrere *Worker* diese abhören, bei Verfügbarkeit entgegennehmen und über die SDK-Funktionalitäten an den Hardwareanbieter liefern, um die parallele Ausführung mehrerer Jobs zu ermöglichen. Eine weitere Warteschlange mit Workern nach demselben Prinzip ist für die parallele Ausführung von Implementierungen gegeben. Für die Asynchronität werden die Messergebnisse und Implementierungsergebnisse nach Erhalt abrufbar in einer relationalen Datenbank bereitgestellt.

7.3 Anwendungsszenario zur Empfehlung von Quantenressourcen

Für den Quantenalgorithmus von *Grover* [Gro96] werden mithilfe des MarieQurie-Frameworks automatisiert geeignete Quantenressourcen empfohlen. Der Grover-Algorithmus wird verwendet, um Elemente in unsortierten Listen zu suchen [Gro96; JAA+22]. Verglichen zu klassischen Ansätzen löst Grover diese Aufgabe mit quadratischer Beschleunigung. Beispielsweise kann Gro-

[1] `https://python-rq.org`

ver zur Lösung des *Erfüllbarkeitsproblems der Aussagenlogik* (engl. „*Boolean Satisfiability Problem (SAT)*“) angewendet werden, wobei die Belegung der boolschen Variablen der boolschen Formel, welche diese zu 1 auswertet, dem zu findenden Element entspricht [RP11].

Für den Grover-Algorithmus liegt für das Szenario eine Implementierung namens *grover-general-sat-qiskit*[1] vor, die anhand einer gegebenen Probleminstanz in Form einer boolschen Formel einen Quantenschaltkreis erzeugt, welcher die Belegung der boolschen Variablen berechnet. Die Implementierung basiert auf der Pythonbibliothek Qiskit. Für das Szenario werden der Qiskit Transpiler [Qis23] und der t|ket⟩ Compiler [SDC+20] sowie fünf 27-Qubit- und ein 127-Qubit-Quantencomputer von IBMQ betrachtet.

Die zu lösende Probleminstanz, die dem MarieQurie-Framework als Eingabe übergeben wird, lautet $(\neg A \vee B) \wedge (A \vee \neg B)$. Als Präferenz werden präzise Ergebnisse gewählt. Zur Prognose der HI-Werte werden *ExtraTreesRegressor* mit dem Meta-Schätzer *AdaBoostRegressor* (Abschnitt 7.2.2) verwendet. Als Datensatz für Vorhersage und Priorisierung dienen 454 analysierte und ausgeführte Kombinationen.[2] Für die Erlernung der Metrikgewichte und zur Priorisierung wird TOPSIS mit COBYLA eingesetzt.

Die resultierende Empfehlung des Frameworks ist in Abbildung 7.6 mit einem Bildschirmfoto der GUI dargestellt. Zu sehen sind die fünf höchstplatzierten Kombinationen mit den Kompilationen, identifizierbar durch den Namen des Quantencomputers (*Backend Name*) und der SDK. Als Ausschnitt wird außerdem die aktuelle Länge der jeweiligen Warteschlange (*Queue Size*) pro Quantencomputer und der vom ML-Algorithmus prognostizierte HI-Wert gezeigt. Links in Abbildung 7.6 ist der errechnete Rang und die zugehörige Rangpunktzahl (*Score*) (Abschnitt 2.3.3) pro Kombination abgebildet, anhand dessen die Kombinationen absteigend sortiert sind. Die zwei Kombinationen mit den höchsten vorhergesagten HI-Werten sind auch die

[1] `https://raw.githubusercontent.com/UST-QuAntiL/nisq-analyzer-content/master/example-implementations/Grover-SAT/grover-general-sat-qiskit.py`

[2] `https://github.com/UST-QuAntiL/nisq-analyzer-content/blob/master/evaluation/evaluation-scenario/eval-scenario-sample-data.csv`

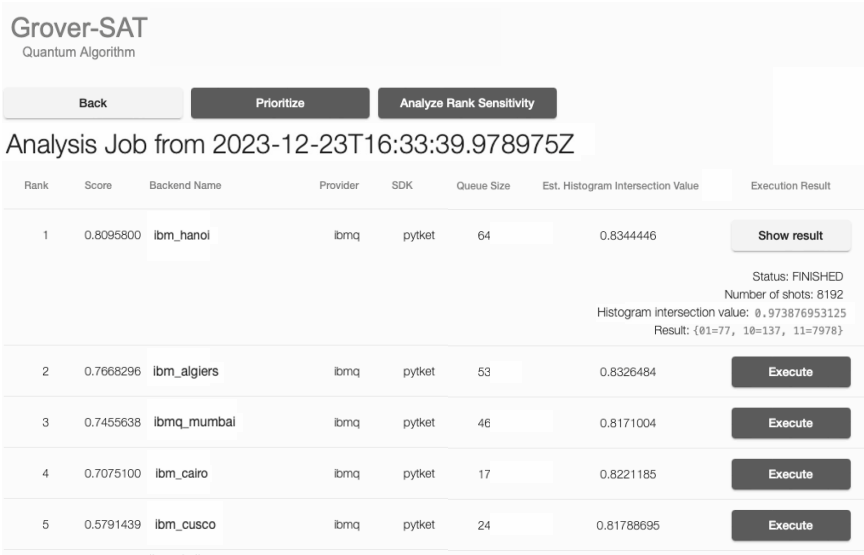

Abbildung 7.6: Empfehlung von Quantenressourcen für Grover

mit den besten Rangplatzierungen. Die fünf nach Präzision platzierten Kombinationen sind ausschließlich Kompilationen des t|ket⟩ Compilers. Auf Platz fünf ist die Kombination mit dem 127-Qubit-Quantencomputer aufgrund seiner vergleichsweise hohen Fehlerraten.

Die erstplatzierte Kompilation wurde auf dem Zielquantencomputer ausgeführt. Die am häufigsten gemessene und korrekte Belegung für A und B lautet 11, wodurch die boolsche Formel zu 1 ausgewertet wird. Die Ergebnispräzision liegt bei einem HI-Wert von circa 0,97 und ist damit um circa 0,14 besser als der prognostizierte HI-Wert. Es zeigt sich, dass der ML-Algorithmus in diesem Fall eine pessimistischere Prognose errechnet hat.

Kapitel 8

Evaluation des MarieQurie-Frameworks

In diesem Kapitel wird die in Forschungsbeitrag 5 enthaltene Evaluation des MarieQurie-Frameworks aus Kapitel 7 präsentiert. Zunächst wird die Anwendung unterstützter ML-Algorithmen zur Prognose geeigneter Kombinationen mit Quantencompilern und -computern evaluiert (Abschnitt 8.1). Zudem werden die im MarieQurie-Framework bereitgestellten Priorisierungsmethoden evaluiert (Abschnitt 8.2). Daraufhin werden die Ergebnisse der Prognose- und Priorisierungsmethoden diskutiert (Abschnitt 8.3). Außerdem werden die Resultate aus Interviews mit unterschiedlichen Nutzenden präsentiert (Abschnitt 8.4). Zuletzt wird die Verwendung des Frameworks in anderen wissenschaftlichen Arbeiten gezeigt (Abschnitt 8.5). Die Evaluationsverfahren wurden basierend auf kleineren Datenmengen, bereits im Rahmen der vorliegenden Arbeit verwendet [SBL+21; SBLW22b; SBLW23].

8.1 Evaluation zur Selektion von Quantenressourcen

Im Folgenden wird die Performanz unterstützter Implementierungen der ML-Algorithmen zur Selektion von Quantencompilern und -computern für Schaltkreise bezüglich der Anforderung präziser Ausführungsergebnisse verglichen (Abschnitt 8.1.1). Anschließend wird der Einfluss der individuellen Schaltkreis- und Quantencomputermetriken sowie Compiler bei der Selektion untersucht (Abschnitt 8.1.2). Außerdem wird die Präzision der Selektion (Abschnitt 8.1.3) und die Laufzeit gemessen (Abschnitt 8.1.4).

Für die Evaluation wurden 85 Schaltkreise mit dem *t|ket⟩ Compiler* [SDC+20] und dem *Qiskit Transpiler* [Qis23] mit höchster Optimierungsstufe auf 13 Quantencomputern von IBMQ sowie einem Quantencomputer von IonQ kompiliert. Während die Quantencomputer von IBMQ mit supraleitenden Qubits rechnen [BKM+14], basieren die Quantencomputer von IonQ auf gefangenen Ionen [BCMS19]. Ionenbasierte Quantencomputer zeichnen sich im Vergleich zu supraleitenden durch längere Dekohärenz- und Gatterzeiten sowie höhere Gatter- und Auslesefidelitäten aus [BCMS19]. Sie ermöglichen eine hohe Qubitkonnektivität, bieten allerdings aktuell weniger Qubits als verfügbare supraleitende [WYW+22]. Supraleitende Quantencomputer ermöglichen schnelle Ausführungszeiten, zeigen jedoch kurze Dekohärenzzeiten und eine begrenzte Konnektivität [BCMS19; Pre18; TQ19b].

Die Schaltkreise haben eine Breite zwischen zwei und 20 Qubits sowie eine Tiefe zwischen neun und 664 Schichten. Es handelt sich zum einen um randomisierte Schaltkreise, bestehend aus Clifford-Gattern, sodass deren Ausführungsergebnisse ohne auftretende Fehler den initialen Quantenzuständen entsprechen [MGJ+12; SBLW22b]. Zum anderen ist jeweils ein Schaltkreis der Quantenalgorithmen von Shor [Sho97] und Bernstein-Vazirani [BV97] sowie zwei des Grover-Algorithmus [Gro96] enthalten [SBLW22a]. Insgesamt konnten 499 Kompilationen ausgeführt werden.[1] Dabei stammt circa die Hälfte der Ausführungsergebnisse von kostenlos zugänglichen 5-Qubit-

[1] `https://github.com/UST-QuAntiL/nisq-analyzer-content/blob/master/evaluation/Dissertation-Sample-Data.csv`

Quantencomputern von IBMQ. Circa ein Viertel der Ergebnisse stammt von 27- und 127-Qubit-Rechnern von IBMQ und das weitere Viertel von dem Quantencomputer von IonQ.

8.1.1 Performanz der ML-Algorithmenimplementierungen

Für den Vergleich der Vorhersageperformanz zwischen gegebenen Implementierungen der ML-Algorithmen wird die *k-fache Kreuzvalidierung (engl. „k-fold-cross-validation")* mit $k = 5$ ähnlich großen, zufälligen Aufteilungen der Datenmenge durchgeführt [JWHT21]. Bei der Aufteilung werden Schaltkreise und die zugehörigen Ausführungsergebnisse in dieselbe Untermenge gelegt, sodass das Testen für noch nicht betrachtete Schaltkreise nachgestellt werden kann [SBLW23]. Dabei wird pro Iteration eine der fünf Untermengen für das Testen, die anderen vier für das Trainieren verwendet. Jede Untermenge dient einmal als Testdatenmenge. Bei jeder i-ten Iteration wird der *mittlere absolute Fehler (engl. „Mean Absolute Error (MAE)") MAE_{ij}* berechnet, welcher schließlich über alle Iterationen hinweg pro j-te Wiederholung gemittelt $\bar{E}_j$ ergibt [SBLW23; WM05; YBS17]. Das Verfahren wird $r = 10$ mal wiederholt, resultierend in $\bar{E}$, sodass sich pro ML-Implementierung insgesamt 50 Trainingsdurchläufe ergeben [SBLW23]:

$$MAE_{ij} = \frac{1}{N} \sum_{n=0}^{N-1} |y_{ijn} - \hat{y}_{ijn}| \tag{8.1}$$

$$\bar{E}_j = \frac{1}{k} \sum_{i=0}^{k-1} MAE_{ij} \tag{8.2}$$

$$\bar{E} = \frac{1}{r} \sum_{j=0}^{r-1} \bar{E}_j \tag{8.3}$$

Hierbei entspricht y_{ijn} dem tatsächlichen HI-Wert und $\hat{y}_{ijn}$ dem zugehörigen vorhergesagten HI-Wert für einen der N Schaltkreise in der betrachteten Datenmenge. Die Abweichung der 50 Ergebnisse wird durch die Wurzel von

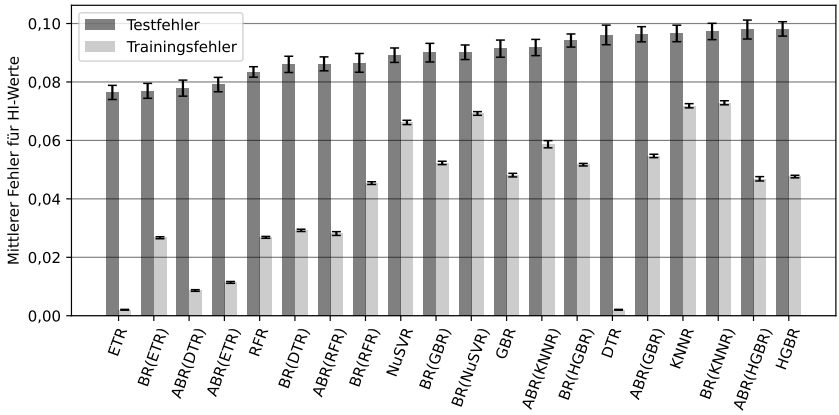

Abbildung 8.1: Mittlerer Fehler der 20 leistungsstärksten gegebenen Implementierungen der ML-Algorithmen

50 geteilt, um den *Standardfehler des Mittelwerts (engl. „Standard Error of the Mean (SEM)“)* zu erhalten [AB05; SBLW23]:

$$SEM = \frac{\sqrt{\frac{1}{k \cdot r - 1} \sum_{j=0}^{r-1} \left(\bar{E}_j - \bar{E}\right)^2}}{\sqrt{k \cdot r}} \tag{8.4}$$

Der über alle 50 Durchläufe gemittelte durchschnittliche absolute Trainings- und Testfehler der 20 am besten abschneidenden Implementierungen der ML-Algorithmen (Abschnitt 7.2.2) ist in Abbildung 8.1 dargestellt.[1] Zudem zeigt die Abbildung durch Fehlerbalken den jeweiligen SEM [SBLW23]. Der Testfehler nimmt zwischen den einzelnen Implementierungen zu, wobei sich alle 20 unter 10 % befinden. Größere Unterschiede sind beim Trainingsfehler zu sehen. Ist der Unterschied zwischen Test- und Trainingsfehler bei einer Implementierung wie ExtraTreesRegressor (ETR) und DecisionTree-

[1] `https://github.com/UST-QuAntiL/nisq-analyzer-content/blob/master/evaluation/ML-evaluation/ml_algo_performance.csv`

Regressor (DTR) sehr hoch, handelt es sich um eine *Überanpassung (engl. „Overfitting")* der Implementierung an die Trainingsdatenmenge [JWHT21]. Bei einer Überanpassung lernen die Implementierungen zwar Muster in den Trainingsdaten, sodass der Fehler in diesem Fall gering ist. Sie lernen jedoch nicht die zugrundeliegenden Zusammenhänge, wodurch der Fehler auf den Testdaten wesentlich größer ist. Nichtsdestotrotz ist der Testfehler der ausschlaggebendere Indikator für die Performanz der Implementierungen. Die geringsten Testfehler und somit die besten Vorhersagen sind bei den diversen Entscheidungsbaumimplementierungen auch in Kombination mit den Meta-Schätzern (Abschnitt 7.2.2) wie ETR, ETR mit dem BaggingRegressor (BR) und DTR mit dem AdaBoostRegressor (ABR) zu erkennen. Diese drei Implementierungen weichen im Schnitt mit ihren geschätzten HI-Werten zwischen 7 % und 8 % von den HI-Werten der tatsächlichen Ausführungsergebnisse ab. Die höchsten Testfehler der 20 besten Implementierungen gehen von HistGradientBoostingRegressor (HGBR) und HGBR mit ABR aus, mit HI-Werten abweichend zwischen 9 % und 10 %. Der SEM ist zwischen den Implementierungen ähnlich groß.

8.1.2 Einfluss der individuellen Metriken bei der Quantenressourcenselektion

In Abbildung 8.2 sind die gewählten Schaltkreis- und Quantencomputermetriken sowie der t|ket⟩ Compiler und Qiskit Transpiler als unabhängige Variablen zu sehen. Anhand dieser unabhängigen Variablen haben die ML-Implementierungen die HI-Werte der verschiedenen Kombinationen aus verfügbaren Quantencompilern und -computern geschätzt. Die Abbildung stellt den durchschnittlichen Wert und die Verteilung der Wichtigkeit bezüglich des Einflusses der einzelnen Metriken und Compiler bei den Vorhersageberechnungen der 20 leistungsstärksten ML-Implementierungen (Abschnitt 8.1.1) dar.[1] Hierfür wurde die Methode von scikit-learn[2] zur *Permutation der Va-*

[1] `https://github.com/UST-QuAntiL/nisq-analyzer-content/blob/master/evaluation/ML-evaluation/feature_importance-of-best-algos.csv`

[2] `https://scikit-learn.org/stable/modules/permutation_importance.html`

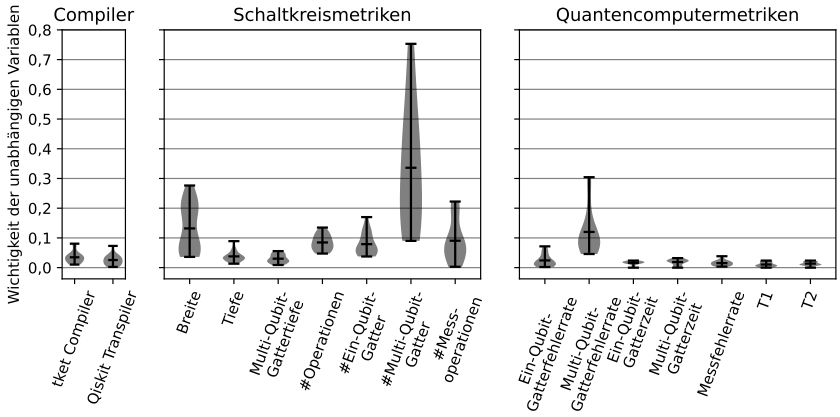

Abbildung 8.2: Mittlere Wichtigkeit der unabhängigen Variablen mit den 20 leistungsstärksten ML-Algorithmenimplementierungen

riablenrelevanz (engl. „Permutation Feature Importance“) verwendet [Bre01]. Hierbei wird eine einzelne Variable betrachtet, dessen in den Trainingsdaten gegebenen Werte zufällig durchmischt werden. Die Durchmischung hebt die Abhängigkeit dieser Variablen zur Zielgröße auf. Durch die Aufhebung kann beobachtet werden, wie stark die Schätzung der Implementierung durch diese Variable beeinflusst wird. Um den Einfluss der verschiedenen Compiler zu untersuchen, werden diese durch die One-Hot-Kodierung in quantitative Variablen überführt (Abschnitt 2.3.1). Die Permutation der Variablenrelevanz wird zehnmal für jede Variable wiederholt [SBLW23]. Dieses Verfahren wird wiederum insgesamt zehnmal ausgeführt, sodass die durchschnittliche Abweichung zur herkömmlichen Zielgröße gemessen werden kann.

Abbildung 8.2 zeigt, dass die zwei Compiler im Mittel einen ähnlichen Einfluss haben, wobei der t|ket⟩ Compiler einen etwas größeren Einfluss bei der Vorhersage von HI-Werten zeigt. Beide haben in der Varianz einen größeren Einfluss als viele Quantencomputermetriken, wie beispielsweise T_1 und T_2. Damit zeigt sich, dass sich die Selektion des Compilers auf die Ergebnispräzision auswirkt, sodass die durch das MarieQurie-Framework automatisierte

Betrachtung mehrerer Compiler für den Nutzenden von Vorteil ist. Des Weiteren zeigt sich, dass insbesondere die gewählten Schaltkreismetriken Einfluss auf die Vorhersage zu präzisen Ergebnissen haben. Den größten Einfluss mit einem Wert von $0,34$ und auch der höchsten Varianz zeigt die Schaltkreismetrik über die Anzahl der Multi-Qubit-Gatter. Darauf folgen bei den Schaltkreismetriken im Mittel die Breite und die Anzahl Operationen, Ein-Qubit-Gatter sowie Messoperationen. Wobei die Breite nicht für jeden Schaltkreis der verwendeten Datenmenge der Anzahl Messoperationen entspricht. Beansprucht die Breite des auszuführenden Schaltkreises einen Großteil der verfügbaren Qubits eines Quantencomputers, kann der Fall eintreten, dass auch fehleranfälligere Qubits für die Berechnung herangezogen werden müssen [TQ19b]. Wissenschaftliche Arbeiten haben gezeigt, dass Multi-Qubit-Gatter mitunter die fehleranfälligsten Operationen auf Quantencomputern sind [SDC+20; TQ19a]. So zeigt sich auch bei den Quantencomputermetriken in Abbildung 8.2, dass die Multi-Qubit-Gatterfehlerrate mit der gegebenen Datenmenge den größten Einfluss hat [SDC+20]. Dadurch zeigt sich, dass die durch das MarieQurie-Framework ermöglichte Berücksichtigung diverser Quantencomputer und deren automatisierte Selektion für Nutzende relevant ist, um für ihre Probleminstanz eine möglichst präzise Berechnung durchführen zu können.

8.1.3 Präzision der Selektion von Quantenressourcen

Mit welcher Wahrscheinlichkeit die ML-Implementierung ETR mit dem geringsten Testfehler (Abschnitt 8.1.1) die geeignetste Kombination aus Quantencompiler und -computer für präzise Ausführungsergebnisse selektiert, ist in Abbildung 8.3 dargestellt.[1] Die geeignetste Kombination für einen initialen Schaltkreis ist diejenige, die den höchsten HI-Wert bei den Ausführungsergebnissen erlangt. Für die Evaluation wurde die 5-fache Kreuzvalidierung mit zehn Wiederholungen angewandt. Der Grenzwert für die maximale Anzahl zurückzugebender Kombinationen (Abschnitt 4.3) wurde schrittwei-

[1] `https://github.com/UST-QuAntiL/nisq-analyzer-content/blob/master/evaluation/ML-evaluation/filter_ratios.csv`

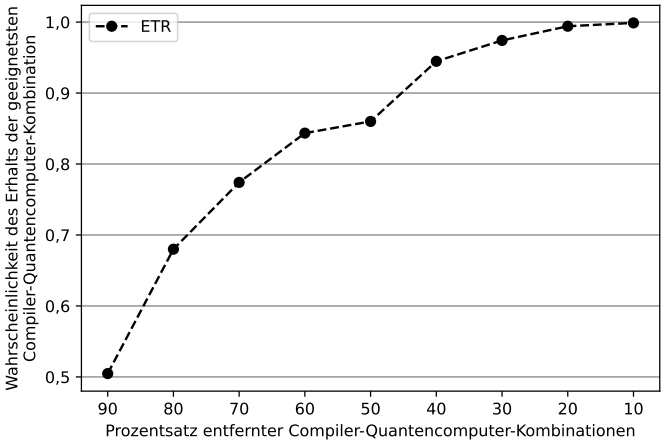

Abbildung 8.3: Selektionspräzision der leistungsstärksten ML-Algorithmenimplementierung ETR

se erhöht. Sollten 90 % der möglichen Kombinationen bei der Selektion verworfen werden, war die geeignetste bei ETR mit über 50 % enthalten. Bei einer Selektion von 50 % der möglichen Kombinationen enthielten die durch das MarieQurie-Framework empfohlenen Kombinationen die geeignetste mit einer durchschnittlichen Wahrscheinlichkeit von über 85 %, wodurch sich zeigt, dass sich die Selektion mittels ML zur Einsparung von Rechenressourcen und Zeit (Abschnitt 8.1.4) eignet.

8.1.4 Laufzeitanalyse zur Selektion von Quantenressourcen

Im Folgenden wird untersucht, inwieweit die Quantenressourcenselektion vor der Kompilierung (Abschnitt 4.3) die Laufzeit des MarieQurie-Frameworks vermindert. Für die Selektion wurde die ML-Algorithmenimplementierung ETR verwendet (Abschnitt 7.2.2). Als Eingabe dienten acht verschiedene randomisierte Clifford-Gatter-Quantenschaltkreise, die in OpenQASM implementiert sind. Die Anforderung von präzisen Ergebnissen und kurzen Wartezeiten ist im Verhältnis 50:50. Die Namen der Schaltkreise entsprechen

Schaltkreis	100 %	50 %	10 %
CGQ-5-13	52	25	16
CGQ-5-357	66	43	29
CGQ-10-47	60	36	22
CGQ-10-93	63	44	24
CGQ-15-17	56	40	23
CGQ-15-101	71	44	25
CGQ-20-23	58	39	23
CGQ-20-730	311	192	98

Tabelle 8.1: Median der Laufzeiten in Sekunden mit schrittweise erhöhten Grenzwerten

dem Format *CGQ-{Breite}-{Tiefe}*. Der Schaltkreis *CGQ-5-13* hat beispielsweise eine Breite von fünf Qubits und eine Tiefe von 13 Schichten. Die Quantencompiler t|ket⟩ [SDC+20] und Qiskit [Qis23] sowie drei Quantencomputer von IBMQ wurden betrachtet, wodurch sich pro Schaltkreis maximal sechs Kombinationen ergaben. Für die Laufzeitanalyse war der Ablauf des MarieQurie-Frameworks wie folgt: Einstiegspunkt ist jeweils die Übergabe einer der Schaltkreise an den zweiten Schritt der gesamtheitlichen MarieQurie-Methode, der Quantenressourcenselektion (Abschnitt 4.3). Da die Schaltkreise bereits in einem Format sind, welches von den SDKs der zwei Compiler unterstützt wird, ist keine Übersetzung nötig und die Schaltkreise werden jeweils direkt kompiliert (Abschnitt 5.4). Abschließend folgt die Ausführbarkeitsanalyse der Kompilationen (Abschnitt 5.5).

Für jeden Schaltkreis wurde der vorgestellte Ablauf mit 100 %, 50 % und mit 10 % aller möglichen Kombinationen an Compilern und Quantencomputern als Grenzwert (Abschnitt 4.3) dreimal durchgeführt und der Median der gemessenen Laufzeiten ermittelt. Für die Laufzeitanalyse wurde das MarieQurie-Framework auf einem MacBook Pro mit Sonoma 14.11, einem Apple M2 Max Chip und 32 GB Speicher ausgeführt.

Tabelle 8.1 zeigt, dass mit einem Grenzwert von 50 % aller möglichen Kombinationen sich die Laufzeit des MarieQurie-Frameworks im Median um

circa 37 % vermindert. Bei einer Reduktion von 50 % auf 10 % des Grenzwerts reduziert sich die Laufzeit ungefähr um 41 %, sodass sich zwischen den Grenzwerten von 100 % und 10 % eine Laufzeitverminderung von circa 63 % ergibt. Eine Abhängigkeit der Laufzeit zur Breite der Schaltkreise ist in diesem Umfang nicht erkenntlich. Jedoch ist zu sehen, dass mit zunehmender Tiefe, vor allem beim Vergleich von *CGQ-20-23* und *CGQ-20-730* zu erkennen, die Laufzeit des MarieQurie-Frameworks aufgrund aufwändigerer Kompilierung zunimmt. Dies zeigt auf, dass die Quantenressourcenselektion vor der Anwendung der Compiler umso fundamentaler für Nutzende im Hinblick auf die Zeiteinsparung ist.

8.2 Evaluation der Priorisierung von Quantenkompilationen

Folgend wird die Performanz der verschiedenen Kombinationen aus MCDA-Methoden und Optimierungsalgorithmen zur Priorisierung von Quantenkompilationen verglichen (Abschnitt 8.2.1). Des Weiteren wird der Einfluss der individuellen Metriken zur Erlernung der Gewichte zur Priorisierung (Abschnitt 6.2) analysiert (Abschnitt 8.2.2). Anschließend wird die Sensitivität der erlernten Metrikgewichte (Abschnitt 6.4) untersucht (Abschnitt 8.2.3). Die Evaluation beruht auf derselben Datenmenge wie in Abschnitt 8.1.

8.2.1 Performanz der Kombinationen aus MCDA-Methoden und Optimierungsalgorithmen

Um zu messen, inwiefern die totalen Ranglisten der verfügbaren Kombinationen aus MCDA-Methoden und Optimierungsalgorithmen (Abschnitt 7.2.2) mit den HI-Werten ausgeführter Kompilationen übereinstimmen, wird der *Spearman-Rangkorrelationskoeffizient (engl. „Spearman rank correlation coefficient“)* [Spe61] eingesetzt [Dod08; Sał20]:

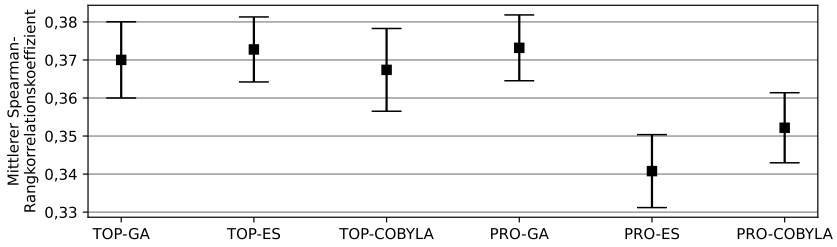

Abbildung 8.4: Durchschnittlicher Spearman-Rangkorrelationskoeffizient mit Standardabweichungen der Kombinationen aus MCDA-Methode und Optimierungsalgorithmus

$$\rho = 1 - \frac{6 \sum_{i=0}^{n-1} (R_{X_i} - R_{Y_i})^2}{n(n^2 - 1)} \tag{8.5}$$

Angewendet auf die Priorisierung von Kompilationen, haben zwei zu vergleichende Ranglisten X und Y in Gleichung (8.5) jeweils n Kompilationen [Dod08; SBLW22b]. Während die Kompilation X_i auf Rang R_{X_i} der Rangliste basierend auf den HI-Werten der Ausführungsergebnisse eines initialen Schaltkreises ist, befindet sich Kompilation Y_i auf Rang R_{Y_i} der errechneten Rangliste einer Kombination aus MCDA-Methode und Optimierungsalgorithmus. Mit $\rho = 1$ sind beide Ranglisten identisch [Dod08]. Gilt $\rho = 0$, zeigt sich keine Übereinstimmung. Exakt gespiegelte Ranglisten ergeben $\rho = -1$. Die Datenmenge wird pro Kombination aus MCDA-Methode und Optimierungsalgorithmus zufällig im Verhältnis 70:30 in Test- und Trainingsdaten aufgeteilt, welches 100-mal wiederholt wird [LvM19; SBLW22b].

Der resultierende durchschnittliche Spearman-Rangkorrelationskoeffizient und dessen mittlere Standardabweichung für alle möglichen Kombinationen unterstützter MCDA-Methoden und Optimierungsalgorithmen ist in Abbildung 8.4 dargestellt. Es zeigt sich, dass TOPSIS (TOP) mit der Evolutionären Strategie (ES) und PROMETHEE II (PRO) zusammen mit dem generischen

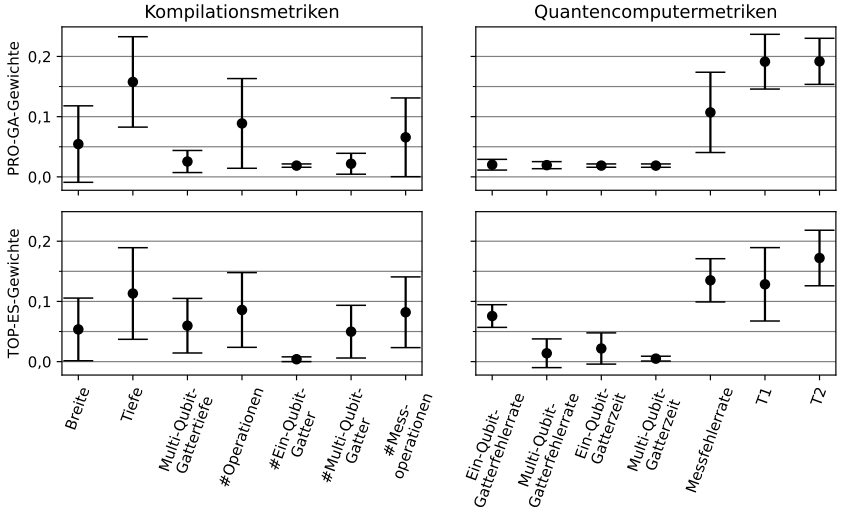

Abbildung 8.5: Durchschnittliche Gewichte der Metriken mit Standardabweichungen von PROMETHEE II mit Genetischem Algorithmus und TOPSIS mit Evolutionärer Strategie

Algorithmus (GA) mit durchschnittlichen Koeffizienten größer 0,37 die besten Kombinationen für die Anwendung des MarieQurie-Frameworks zur Berechnung von Ranglisten in Bezug auf die Anforderung von präzisen Ausführungsergebnissen sind. Darauf folgen die Kombinationen TOP-GA und TOP mit COBYLA. Die im Schnitt am meisten abweichenden Ranglisten berechnet PRO-ES mit der verfügbaren Datenmenge. Die Standardabweichungen der Kombinationen weisen im Vergleich ähnliche Größen auf.

8.2.2 Einfluss der individuellen Metriken bei der Gewichtsbestimmung

Abbildung 8.5 zeigt die Metrikgewichte der leistungsstärksten Kombinationen PRO-GA und TOP-ES auf. Bei den Metrikgewichten handelt es sich um die durchschnittlich berechneten Gewichte aller Prozesswiederholungen von Abschnitt 8.2.1, die für die Priorisierung von Kompilationen eines Schaltkreises

verwendet wurden [SBLW22b]. Bei PRO-GA und TOP-ES ist in Abbildung 8.5 zu sehen, dass die Tiefe der Kompilationen sowie die Messfehlerrate und die Dekohärenzzeiten T_1 und T_2 der Quantencomputer mit Gewichten über $0,1$ die größten Gewichte erhielten. Bis auf die Ein-Qubit-Gatterfehlerrate bei TOP-ES haben die anderen Quantencomputermetriken vergleichsweise geringe Gewichte. Bei den Metriken der Kompilationen zeigt sich, dass zudem die Breite und die Anzahl der Operationen und Messoperationen größere Gewichte bei beiden Kombinationen erhielten. Bei TOP-ES zeigen außerdem die Multi-Qubit-Gattertiefe und die Anzahl der Multi-Qubit-Gatter erhöhte Gewichte. Sowohl die teils einflussreichen Kompilations- als auch Quantencomputermetriken zeigen, dass für möglichst präzise Ergebnisse zum einen ein geeigneter Compiler gewählt werden muss, sodass die Kompilation ressourcensparsam ausfällt. Zum anderen ist auch die Wahl des Quantencomputers abhängig von der vorliegenden Kompilation aufgrund variierender Fehlerraten und Dekohärenzzeiten fundamental für die Ergebnispräzision.

8.2.3 Analyse der Sensitivität erlernter Metrikgewichte

Um die Sensitivität der resultierenden Ranglisten aus Abschnitt 8.2.2 zu untersuchen, werden Sensitivitätsanalysen (Abschnitt 6.4) auf die berechneten Gewichte von PRO-GA und TOP-ES angewendet. Die Ergebnisse für PRO-GA sind in Abbildung 8.6 und für TOP-ES in Abbildung 8.7 zu sehen. Für die Analyse in dieser Evaluation wird als Vorschlag das γ bei 1 beginnend in 1 %-Schritten bis zu einem Wert von $0,99^{500} \approx 0,0066$ reduziert, wodurch das jeweils zu betrachtende normalisierte Metrikgewicht nahe 0 geht [SBLW22b]. Dadurch hat das jeweilige Gewicht verglichen mit den anderen Gewichten zunehmend weniger Einfluss auf die Priorisierung. In selber Schrittgröße werden die Gewichte jeweils bis zu einem Wert von $\gamma = 1,01^{500} \approx 144,77$ erhöht, um normalisiert das höchste Gewicht im Vergleich zu den anderen Metriken zu erlangen. Abbildungen 8.6 und 8.7 stellen das durchschnittliche γ pro Metrik dar, für das pro Satz an berechneten Gewichten der 100 Prozesswiederholungen (Abschnitt 8.2.1) im Schnitt eine Veränderung der ursprünglichen Ranglisten stattfand. Ist das γ in Abbildungen 8.6 und 8.7

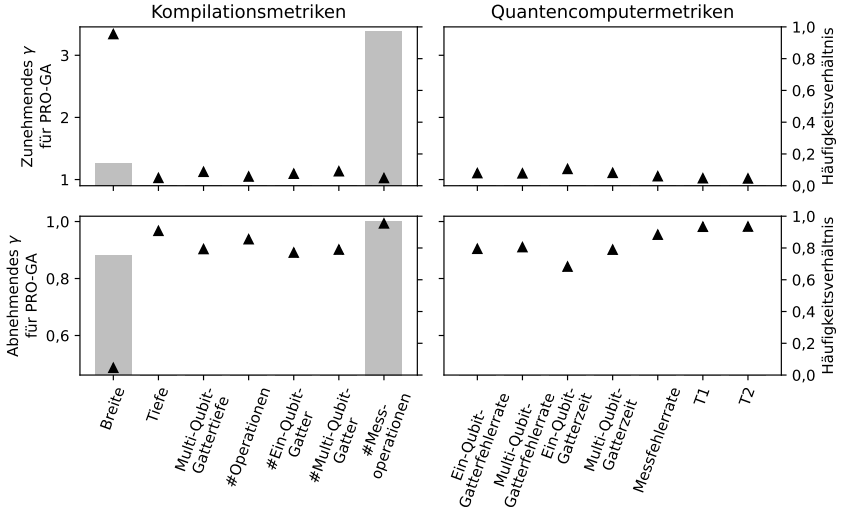

Abbildung 8.6: Sensitivitätsanalyse der durchschnittlichen Metrikgewichte von PROMETHEE II mit Genetischem Algorithmus für zu- (oben) und abnehmendes (unten) γ (Dreiecke) sowie Häufigkeitsverhältnisse bezüglich sporadisch vorkommender Ranglistenveränderungen (Balken)

bei Erhöhung (oben) als auch Verringerung (unten) nahe 1, veränderten sich die Ranglisten bereits bei kleinen Gewichtsanpassungen. Bei PRO-GA in Abbildung 8.6 reagieren die Ranglisten bei Veränderung der Metrikgewichte bis auf bei der Breite im Schnitt sehr sensibel. Bei TOP-ES in Abbildung 8.7 zeigt sich, dass bei zu- als auch abnehmendem γ, die Gewichte für die Anzahl der Ein-Qubit-Gatter sowie die Ein- und Multi-Qubit-Gatterzeiten sich im Verhältnis weniger auf die Ranglisten auswirken.

Die Balken in den Abbildungen stellen dar, wie häufig eine Ranglistenveränderung durch γ pro Metrik für die insgesamt 100 Gewichtssätze der Prozesswiederholungen im Verhältnis stattgefunden hat [SBLW22b]. Ist das Verhältnis bei 1, wie bei der Anzahl Messoperationen in Abbildung 8.7, wurde innerhalb der Schranken für γ in keiner der 100 Gewichtssätze eine

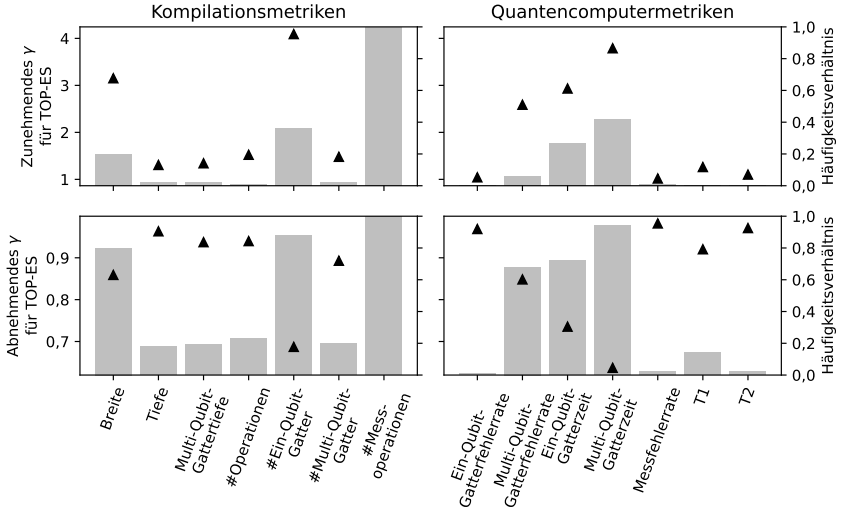

Abbildung 8.7: Sensitivitätsanalyse der durchschnittlichen Metrikgewichte von TOPSIS mit Evolutionärer Strategie für zu- (oben) und abnehmendes (unten) γ (Dreiecke) sowie Häufigkeitsverhältnisse bezüglich sporadisch vorkommender Ranglistenveränderungen (Balken)

Veränderung der Ranglisten gefunden. In diesem Fall zeigt sich, dass die Gewichte dieser Metriken nicht ausschlaggebend für die Rangreihenfolge sind. Ist der Verhältniswert bei 0, fand bei allen Gewichtssätzen pro Gewicht eine Veränderung der Ranglisten statt, wie beispielsweise der Ein-Qubit-Gatterfehlerrate. Diese Metriken haben somit wiederholt starken Einfluss auf die Rangreihenfolge. Es wurde für alle Metriken außer der Breite und der Anzahl Messoperationen in jedem Satz an Gewichten von PRO-GA ein γ gefunden, welches die initialen Ranglisten veränderte. Insgesamt sind die Ranglisten mit den ermittelten Metrikgewichten von PRO-GA sehr sensibel. Somit lohnt es sich hierbei für Nutzende die Sensitivitätsanalyse nach der Priorisierung durchzuführen, um mögliche Änderungen in den Rangplatzierungen bei der Analyse erkennen und bewerten zu können.

8.3 Diskussion der Evaluationsergebnisse

Die in Abschnitt 8.1 und 8.2 verwendete Datenmenge enthält hauptsächlich randomisierte Schaltkreise, die keinen realistischen Anwendungsfällen entsprechen. Des Weiteren stammen die Ergebnisse in unterschiedlichen Anteilen von unterschiedlichen Quantencomputern (Abschnitt 8.1). Anhand der Performanzmessungen der ML-Algorithmenimplementierungen (Abschnitt 8.1.1) und der Kombinationen aus MCDA-Methoden und Optimierungsalgorithmen (Abschnitt 8.2.1) zeigt sich in Abhängigkeit zur gegebenen Datenmenge, inwiefern zuverlässige Vorhersagen und Ranglisten für die Anforderung an präzise Ergebnisse anhand der gewählten Metriken gemacht werden können. Aus den Analysen über die Einflüsse der Metriken in Abschnitt 8.1.2 und 8.2.2 zeigt sich, dass die Größe und Struktur der Schaltkreise und Kompilationen die Präzision von Ausführungsergebnissen beeinflussen und dass es daher wichtig ist, neben der Wahl eines geeigneten Quantencomputers ebenso einen geeigneten Quantencompiler einzusetzen [QBW23d; SBLW22b; TGO+22]. In Abschnitt 8.2.2 ist in Bezug auf die unterschiedlichen Quantencomputer ein großer Einfluss der Dekohärenzzeiten und Messfehlerraten auf die Ergebnispräzision erkenntlich. Da die Ausführung jedoch nur auf einem einzelnen ionenbasierten Quantencomputer möglich war, ist eine Gegenüberstellung der Gewichte pro Hardwaretechnologie mit der verfügbaren Datenmenge nicht aufschlussgebend[1]: Die gleichgebliebenen Quantencomputermetrikwerte des Quantencomputers von IonQ ermöglichen es den Optimierungsalgorithmen nicht, anhand dessen Gewichte zu lernen. Stattdessen werden in diesem Fall die Gewichte gleichmäßig auf die Quantencomputermetriken verteilt. Des Weiteren lassen sich direkte Zusammenhänge im Vergleich zwischen den Einflüssen der Metriken bezüglich Schaltkreisen und Kompilationen in den Abbildungen 8.2 und 8.5 schwer aufweisen: Die Schaltkreise können sich durch die Kompilierung, abhängig von den diversen Compilern und Quantencomputern, erheblich ver-

[1] `https://github.com/UST-QuAntiL/nisq-analyzer-content/tree/master/evaluation/MCDA-optimizer-evaluation/MCDA-optimizer-split-ionq-ibm-evaluation`

ändern. Außerdem handelt es sich bei ML-Algorithmen und MCDA-Methoden in Kombination mit Optimierungsalgorithmen um zwei unterschiedliche Ansätze, um Aussagen anhand gegebener Anforderungen treffen zu können. Diese werden in der MarieQurie-Methode jedoch unabhängig voneinander angewendet, sodass sie sich nicht gegenseitig beeinflussen. Dadurch könnten beispielsweise die Kombinationen aus MCDA-Methoden und Optimierungsalgorithmen zur Priorisierung von Kompilationen ebenfalls durch ML-Algorithmen ersetzt werden. Durch die Plug-in-basierte Architektur kann das MarieQurie-Framework mit weiteren Metriken, Implementierungen von ML-Algorithmen, MCDA-Methoden und Optimierungsalgorithmen sowie SDKs mit ihren Compilern und Quantencomputern ergänzt werden.

8.4 Befragung Nutzender zum MarieQurie-Framework

Um die Nutzbarkeit des MarieQurie-Frameworks zu validieren, wurde dieses von drei Testpersonen der Firmen *Anaqor*, *Deutsche Bahn* und *d-fine* getestet. Jede Testperson übergab einen eigenen, bereits extrahierten Quantenschaltkreis, auf dem die MarieQurie-Methode angewendet werden sollte. Anschließend wurden die Testpersonen unstrukturiert interviewt.

Die Testperson von Anaqor übergab den OpenQASM-Schaltkreis einer Qiskit-Implementierung der *Quantum Boltzmann Maschine*, deren Details in der Arbeit von Paul, Falkenthal und Feld [PFF23] enthalten sind. Die Testperson der Deutschen Bahn verwendete einen extrahierten OpenQASM-Schaltkreis einer Cirq-Implementierung, welcher zufällige Eingabeparameter zur *Anomalieerkennung* mittels künstlicher neuronaler Netze übergeben wurde. Auch die Testperson von d-fine übergab einen OpenQASM-Schaltkreis einer Cirq-Implementierung zur Anomaliedetektion, dessen Details in der Arbeit von Herr, Obert und Rosenkranz [HOR21] nachgelesen werden können.

Die Schaltkreise der Testpersonen wurden mit dem MarieQurie-Framework analysiert. Für den Schaltkreis von Anaqor wurde ausschließlich der Simulator empfohlen, da dieser Schaltkreis für die zugänglichen Quantencomputer zu diesem Zeitpunkt mit einer bereits initialen Tiefe von 204 Schichten

laut der Empfehlung des Frameworks zu tief war. Für die anderen beiden Schaltkreise wurden außerdem verfügbare Quantencomputer empfohlen.

Die Testpersonen wurden befragt, *anhand welcher Eigenschaften sie die Quantencomputer bisher für Ausführungen selektierten*. Hierbei gaben alle an, vor allem die verfügbaren Qubits der Quantencomputer und die Wartezeiten zu berücksichtigen. Eine Testperson gab an, zumeist den Quantencomputer mit der höchsten Qubitzahl auszuwählen. Eine andere Testperson berücksichtigt zusätzlich die Fehlerraten der Quantencomputer. Eine weitere Person betrachtet außerdem die Konnektivität der Qubits.

Zudem wurden die Personen gefragt, *für welche Fälle sie das* MarieQurie-Framework *einsetzen würden*. Eine Testperson würde dieses einsetzen, sobald mehrere Quantencomputer zur Auswahl stünden. Zwei der drei Personen gaben an, das Framework vor allem für die initialen Schaltkreise von *variationellen Quantenalgorithmen (VQAs)* [CAB+21] verwenden zu wollen. Bei einem VQA werden nach Ausführung eines parametrisierten Quantenschaltkreises die Parameter mit einem klassischen Computer anhand der Messergebnisse angepasst [BL23; CAB+21]. Der Optimierungsvorgang wird so lange wiederholt, bis eine gegebene Bedingung erfüllt ist. So würden sie das Framework einsetzen, um die Eignung der verfügbaren Quantencomputer vor der Optimierung der Parameter abschätzen zu können.

Die Testpersonen wurden zudem darüber interviewt, *ob sie die Anwendung des* MarieQurie-Frameworks *und dessen einzelne Konzepte nützlich finden*. Dabei empfanden die Personen die Empfehlung von Quantencomputern für einen Schaltkreis und dessen automatisierte Übersetzung in andere Formate als hilfreich. Außerdem fanden sie hilfreich, dass die Möglichkeit besteht, sich Implementierungen durch das Framework selektieren zu lassen. Auch der Vergleich von mehreren Compilern wurde von ihnen als sinnvoll bewertet, wobei alle angaben, dass sie bisher stets den enthaltenen Standardcompiler des jeweiligen SDKs verwendet und keinen Vergleich aufgestellt haben. Zwei Personen gaben zusätzlich an, dass ihnen die unterschiedlichen Größen und Strukturen der Kompilationen verschiedener Compiler für einen einzelnen Eingangsschaltkreis bisher nicht bewusst waren. Dabei gaben zwei Testper-

sonen an, dass sie vor allem die Tiefe der Kompilationen vergleichen. Die dritte Person gab an, dass sie die Fehlerraten der Multi-Qubit-Gatter und die Multi-Qubit-Gattertiefe der Kompilationen betrachtet. Weiterhin fanden die Testpersonen das Priorisieren der Kompilationen sowie die vordefinierten Präferenzen nach präzisen Ergebnissen und kurzen Wartezeiten sinnvoll. Eine Person fand außerdem vorteilhaft, dass auch eigene Gewichte für die Priorisierung gewählt werden können. Zwei Personen gaben an, dass die Möglichkeit, empfohlene Schaltkreise direkt auf den Zielquantencomputern ausführen zu können, nutzerfreundlich ist. Eine dieser Personen gefiel zudem der Vergleich mit den Simulatorergebnissen.

Nachfolgend wurden die Testpersonen befragt, *welche Eigenschaften des* MarieQurie-Frameworks *sie als unnötig empfinden*. Zum einen empfanden die Personen als überflüssig, dass die maximale Anzahl zu empfehlender Kombinationen bestimmt werden kann. Ihnen war vor allem wichtig, nicht allzu lange auf die Empfehlung warten zu müssen. Hierbei gab eine Person an, nicht länger als zehn Minuten warten zu wollen. Die Sensitivitätsanalyse wurde von einer Person als unnötig empfunden, da sie direkt den Quantencomputer mit der geringsten Wartezeit für eine Ausführung wählen würde.

Eine *Zeitersparnis* sahen zwei Testpersonen nicht, da sie Schaltkreise zuvor nicht analysiert hatten. Sie haben jedoch die Annahme, dass durch das Framework bessere Ausführungsergebnisse erzielt werden können.

Schließlich wurden die Testpersonen nach *Verbesserungsvorschlägen* gefragt. Hierbei schlugen die Personen eine integrierte Unterstützung von variationellen Quantenalgorithmenimplementierungen im Framework vor, sodass das Anpassen von Eingabeparametern und das wiederholte Ausführen entsprechend adaptierter Schaltkreise automatisiert unterstützt werden. Des Weiteren äußerte eine Person den Wunsch, Messfehlerminderungstechniken und -korrekturen in das Framework zu integrieren (Abschnitt 6.5).

Auf die Frage hin, *ob sie das* MarieQurie-Framework *wiederverwenden würden*, bejahten zwei der drei Testpersonen diese. Die dritte Person ist auf das Entwickeln von Algorithmen fokussiert und weniger auf die praktische Umsetzung, weswegen diese keinen Bedarf nach dem Framework sieht.

8.5 Verwendung des MarieQurie-Frameworks in anderen Arbeiten

Während die in der vorliegenden Arbeit vorgestellte Methode zur Empfehlung geeigneter Quantenressourcen grundsätzlich im Kontext der Ausführung von Quantenalgorithmenimplementierungen stattfindet [BGT23; WBL+20; WBLV22], werden im Folgenden andere Arbeiten vorgestellt, die Bestandteile des MarieQurie-Frameworks integriert haben.

8.5.1 QC-Atlas

Das vollständige MarieQurie-Framework ist in die Open-Source-Plattform *QC-Atlas*[1] integriert. Der QC-Atlas ermöglicht das Speichern und Teilen von Wissensartefakten wie klassischen oder Quantenalgorithmen, Implementierungen und Publikationen [BLF+21; LBF19]. Das MarieQurie-Framework greift im QC-Atlas auf gegebene Quantenalgorithmen und Implementierungen zu, um geeignete Quantenressourcen empfehlen zu können (Abschnitt 3.2). Zusätzlich können durch Anmeldung verfügbare Wissensartefakte der *PlanQK*-Plattform [Pla24] für Quantenapplikationen über den QC-Atlas abgerufen werden, sodass auch auf den dort gegebenen Quantenalgorithmen und -implementierungen die MarieQurie-Methode angewendet werden kann. Zudem ist der Pattern Atlas[2] [LB21; WBB+20] mit der darin enthaltenen Quantencomputingmustersprache [Ley19; WBLV21] (Abschnitte 4.2 und 5.3) in den QC-Atlas integriert, sodass dort beschriebene Quantenalgorithmen und deren Implementierungen über den QC-Atlas abgerufen und mit dem MarieQurie-Framework analysiert werden können.

8.5.2 QHAna

Das *Quantum Humanities Analysis Tool (QHAna)* [Bar22] ermöglicht Nutzenden klassische sowie Quanten-ML-Algorithmen, das Laden und Präparieren

[1] `https://github.com/UST-QuAntiL/qc-atlas`
[2] `https://github.com/PatternAtlas`

von Daten sowie das Extrahieren von unabhängigen Variablen auf verfügbare Daten anzuwenden [Bar22; BL21]. Das Plug-in-basierte, erweiterbare Werkzeug[1] erlaubt es Nutzenden klassische und Quantenalgorithmen anhand individueller Anwendungsfälle zu vergleichen [BBH+22; BL19; BL20]. Für die Empfehlung geeigneter Quantencompiler und -computer für verfügbare Schaltkreise (Abschnitt 3.2) von Quanten-ML-Algorithmen in QHAna, wurde ein Plug-in[2] für das MarieQurie-Framework implementiert, sodass die Anwendung der Algorithmen zusätzlich erleichtert wird.

8.5.3 Quantenworkflows

Aufgrund der Hybridität von Quantenalgorithmen bietet es sich an, diese mit *Workflow-Technologien* zu orchestrieren, um den Daten- sowie Kontrollfluss der verschiedenen klassischen und Quantensoftwareanteile modellieren und ausführen zu können [LR00; VBLW22; WBLW20]. Die Arbeit von Weder et al. [WBLW20] erweitert mit der *Quantum Modeling Extension (QuantME)* die aus dem Klassischen kommende Workflow-Technologie, sodass sie auch die Anforderungen der Softwareentwicklung im Quantencomputing erfüllt. Weder et al. [WBLS21a] ermöglichen durch die Integration des MarieQurie-Frameworks die automatisierte Selektion geeigneter Quantencomputer für einen gegebenen Quantenschaltkreis zur Ausführungszeit eines modellierten Workflows. Mit der Arbeit wird das aufwändige Modellieren des Workflows für alle möglichen Quantencomputer zur Designzeit vermieden [WBLS21a]. Zudem erspart die Arbeit das kostenintensive Bereitstellen aller benötigten Services für jeden Quantencomputer zur Ausführung, von denen schlussendlich nur eine Teilmenge tatsächlich benutzt wird.

[1] `https://github.com/UST-QuAntiL/qhana`

[2] `https://github.com/UST-QuAntiL/qhana-plugin-runner/tree/main/stable_plugins/nisq_analyzer`

Kapitel 9

Zusammenfassung und Ausblick

Existierende Quantencomputer unterliegen aufgrund hoher Fehlerraten starken Hardwareeinschränkungen, wodurch ihre Einsatzmöglichkeiten bei relevanten Probleminstanzen beschränkt sind. Nichtsdestotrotz nimmt das Interesse an Quantencomputing stetig zu und die Anwendung dessen wird in verschiedenen Bereichen erprobt. Allerdings ist die Verwendung von Quantencomputern aufgrund des heterogenen Softwareangebots sowie den diversen Schnittstellen der Hardwareanbieter für Nutzende mit vielen komplexen Herausforderungen verbunden. Fehlt den Nutzenden außerdem die nötige Expertise im Bereich des Quantencomputings, fällt nicht nur die praktische Umsetzung von Quantenalgorithmen schwer, sondern ebenso die Wahl eines geeigneten Quantencomputers zur Ausführung der beschriebenen Quantenschaltkreise zur Lösung einer gegebenen Probleminstanz.

Forschungsbeitrag 1 stellt die MarieQurie-Methode vor (Kapitel 3). Diese ermöglicht die automatisierte Empfehlung geeigneter Quantenimplementie-

rungen sowie Quantenschaltkreise, Quantencompiler und Quantencomputer anhand eines selektierten Quantenalgorithmus und einer zu lösenden Probleminstanz. Für die Empfehlung werden mit Forschungsbeitrag 2 zunächst basierend auf dem anzuwendenden Quantenalgorithmus Quantenimplementierungen selektiert und die anhand der Probleminstanz generierten Quantenschaltkreise extrahiert (Kapitel 4).

Neben der Diversität der Hardware heutiger NISQ-Rechner unterscheiden sich ebenfalls die Abbildungs- und Optimierungsmethoden verfügbarer Quantencompiler. So ist die Betrachtung verschiedener Compiler und Quantencomputer essenziell, um möglichst optimale Kompilationen zu erhalten und während der Ausführung auftretende Fehler zu reduzieren. Die MarieQurie-Methode wählt durch die Anwendung von ML auf historischen Daten verschiedene Kombinationen aus Quantenschaltkreisen, Quantencompilern und Quantencomputern, die entsprechend der Anforderungen der Nutzenden die gewünschten Ergebnisse liefern könnten.

Aufgrund unterschiedlicher Formatanforderungen der heutigen heterogenen SDKs kann der Fall eintreten, dass Schaltkreise nicht nativ mit beliebigen Compilern kompiliert und auf beliebigen Quantencomputern ausgeführt werden. Daher übersetzt die MarieQurie-Methode die selektierten Schaltkreise, wenn nötig, in die Formate der SDKs ausgewählter Quantencompiler.

Forschungsbeitrag 3 diskutiert zunächst die Ausführbarkeit von Schaltkreisen in Abhängigkeit zu ihrer Größe (Kapitel 5). Da die Kodierung der klassischen Daten in einen Quantenzustand als Initialisierungsschritt von Quantenalgorithmen die Größe von Schaltkreisen erheblich beeinflussen kann, sind verschiedene Kodierungen und Kodierungsverfahren in Form von Mustern präsentiert. Zudem verändert die Kompilierung die Größe der Schaltkreise. Die Kompilationen dabei direkt auf den Quantencomputern auszuführen kann falsche Ergebnisse und unnötige Kosten verursachen. Daher kompiliert die MarieQurie-Methode die gegebenen Schaltkreise zunächst mit den selektierten Compilern auf den zugehörigen Quantencomputern. Anschließend überprüft sie die Ausführbarkeit der Kompilationen anhand ihrer resultierenden Größe sowie den Gatter- und Dekohärenzzeiten der Quantencomputer.

Als Resultat können gegebenenfalls mehrere Kompilationen als ausführbar eingestuft werden. Um Nutzenden daher die Wahl der Kompilationen für die Ausführung zu vereinfachen, priorisiert die MarieQurie-Methode mit Forschungsbeitrag 4 die Kompilationen anhand der aktuellen Hardware- und Schaltkreismetrikwerte sowie den gestellten Anforderungen (Kapitel 6). Hierfür werden Optimierungsalgorithmen und MCDA-Methoden verwendet.

Forschungsbeitrag 5 beschreibt die Architektur der MarieQurie-Methode und präsentiert die prototypische Implementierung des Frameworks (Kapitel 7). Die Implementierung wurde evaluiert und ermöglicht die Empfehlung geeigneter Quantenressourcen in anderen Arbeiten (Kapitel 8).

Zukünftig soll die in Abschnitt 5.2 vorgestellte Formel evaluiert werden, um eine präzisere Abschätzung bezüglich der Ausführbarkeit von Quantenschaltkreisen und deren Kompilationen zu ermöglichen. Um Nutzenden in Bezug auf Servicequalitäten der Hardwareanbieter weitergehende Empfehlungen aussprechen zu können, sollen außerdem Abschätzungen zu monetären Kosten getroffen werden, die durch Laufzeiten auf den bereitgestellten Quantencomputern anfallen können [SAH+16].

Um große Quantenschaltkreise mit NISQ-Rechnern zu berechnen, können diese aufgeteilt und auf mehreren Quantencomputern verteilt ausgeführt werden [BBL+23; BBLM23; BDG+22; FBB+23]. Das MarieQurie-Framework soll zukünftig an die *Circuit Knitting Toolbox* [BBB+23b] von Qiskit angebunden werden, sodass Schaltkreise, für die keine geeigneten Quantencomputer zur Verfügung stehen, in kleinere aufgeteilt werden.

Um Fehler bei der Berechnung mit Quantencomputern zu vermeiden, existieren verschiedene Fehlerkorrekturverfahren [BBL+22b]. Da diese jedoch ähnlich zur Kodierung klassischer Daten (Abschnitt 5.3) die Schaltkreise stark vergrößern, werden in der NISQ-Ära Fehlerminderungsverfahren angewendet, die vergleichsweise nur wenige bis keine zusätzlichen Qubits und Gatter benötigen [BBL+22b; Pre18]. Die MarieQurie-Methode soll erweitert werden, sodass, entsprechend der Präzisionsvorhersagen (Abschnitt 4.3), die Schaltkreise zur Fehlerminderung vor der Ausführung automatisiert adaptiert werden, um die Ergebnispräzision zu erhöhen [BBL+22a; BBL+22b].

Ein weiterer Ansatz, um die Vorteile von NISQ-Rechnern zu nutzen, ist die Anwendung von VQAs (Abschnitte 8.4 und 8.5) [BBB+23a; BBG+23; BL23; WBLV21]. Zukünftig soll das MarieQurie-Framework erweitert werden, sodass dieses Schaltkreisveränderungen während des Optimierungsvorgangs beobachtet und Empfehlungen gibt, sobald sich ein anderer Quantencomputer für präzisere Ergebnisse eignet [BBB+23a; WBBL22; WBBL23].

Literaturverzeichnis

[Aar15] S. Aaronson. „Read the fine print“. In: *Nature Physics* 11.4 (2015), S. 291–293 (Zitiert auf S. 85).

[AB05] D. G. Altman, J. M. Bland. „Standard deviations and standard errors“. In: *Bmj* 331.7521 (2005), S. 903 (Zitiert auf S. 130).

[AB17] D. Alper, C. Başdar. „A Comparison of TOPSIS and ELECTRE Methods: an Application on the Factoring Industry“. In: *Business and Economics Research Journal* 8.3 (2017), S. 627 (Zitiert auf S. 48).

[AC16] S. Aaronson, L. Chen. *Complexity-Theoretic Foundations of Quantum Supremacy Experiments*. 2016. arXiv: `1612.05903 [quant-ph]` (Zitiert auf S. 33).

[ADK+08] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, O. Regev. „Adiabatic Quantum Computation Is Equivalent to Standard Quantum Computation“. In: *SIAM Review* 50.4 (2008), S. 755–787 (Zitiert auf S. 29).

[AG20] M. Amy, V. Gheorghiu. „staq—A full-stack quantum processing toolkit“. In: *Quantum Science and Technology* 5.3 (2020), S. 034016 (Zitiert auf S. 15, 33, 44, 45, 99).

[AIS77] C. Alexander, S. Ishikawa, M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, Aug. 1977 (Zitiert auf S. 54, 99).

[AK15] M. Awad, R. Khanna. „Support Vector Regression“. In: *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Apress, 2015, S. 67–80 (Zitiert auf S. 47).

[Alp16] E. Alpaydin. *Machine learning: the new AI*. MIT press, 2016 (Zitiert auf S. 106).

[Ama23] Amazon Web Services. *Amazon Braket*. `https://aws.amazon.com/braket/`. 2023 (Zitiert auf S. 44, 45).

[Aru+19] F. Arute et al. „Quantum supremacy using a programmable superconducting processor“. In: *Nature* 574.7779 (2019), S. 505–510 (Zitiert auf S. 13).

[BACA14] A. Bilbao-Terol, M. Arenas-Parra, V. Cañal-Fernández, J. Antomil-Ibias. „Using TOPSIS for assessing the sustainability of government bond funds“. In: *Omega* 49 (2014), S. 1–17 (Zitiert auf S. 48, 49, 52).

[Bar22] J. Barzen. „From Digital Humanities to Quantum Humanities: Potentials and Applications“. In: *Quantum Computing in the Arts and Humanities: An Introduction to Core Concepts, Theory and Applications*. Springer, 2022, S. 1–52 (Zitiert auf S. 146, 147).

[BBB+23a] M. Beisel, J. Barzen, M. Bechtold, F. Leymann, F. Truger, B. Weder. „QuantME4VQA: Modeling and Executing Variational Quantum Algorithms Using Workflows“. In: *Proceedings of the 13th International Conference on Cloud Computing and Services Science (CLOSER 2023)*. SciTePress, 2023, S. 306–315 (Zitiert auf S. 152).

[BBB+23b] L. Bello, A. M. Brańczyk, S. Bravyi, A. Carrera Vazquez, A. Eddins, D. J. Egger, B. Fuller, J. Gacon, J. R. Garrison, J. R. Glick, T. P. Gujarati, I. Hamamura, A. I. Hasan, T. Imamichi, C. Johnson, I. Liepuoniute, O. Lockwood, M. Motta, C. D. Pemmaraju, P. Rivero, M. Rossmannek, T. L. Scholten, S. Seelam, I. Sitdikov, D. Subramanian, W. Tang, S. Woerner. *Circuit Knitting Toolbox*. `https://github.com/Qiskit-Extensions/circuit-knitting-toolbox`. `https://doi.org/10.5281/zenodo.7987997`. 2023 (Zitiert auf S. 151).

[BBB+23c] F. Bühler, J. Barzen, M. Beisel, D. Georg, F. Leymann, K. Wild. „Patterns for Quantum Software Development“. In: *Proceedings of the 15th International Conference on Pervasive Patterns and Applications (PATTERNS 2023)*. Xpert Publishing Services (XPS), 2023, S. 30–39 (Zitiert auf S. 46).

[BBC+17] L. Bishop, S. Bravyi, A. Cross, J. M. Gambetta, J. Smolin. *Quantum Volume*. Techn. Ber. 2017 (Zitiert auf S. 21, 36, 83, 100).

[BBG+23] M. Beisel, J. Barzen, S. Garhofer, F. Leymann, F. Truger, B. Weder, V. Yussupov. „Quokka: A Service Ecosystem for Workflow-Based Execution of Variational Quantum Algorithms“. In: *Service-Oriented Computing – ICSOC 2022 Workshops*. Springer, 2023, S. 369–373 (Zitiert auf S. 14, 41, 152).

[BBH+22] F. Bühler, J. Barzen, L. Harzenetter, F. Leymann, P. Wundrack. „Combining the Best of Two Worlds: Microservices and Micro Frontends as Basis for a New Plugin Architecture“. In: *Proceedings of the 16th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2022)*. Springer, 2022, S. 3–23 (Zitiert auf S. 147).

[BBL+22a] M. Beisel, J. Barzen, F. Leymann, F. Truger, B. Weder, V. Yussupov. „Configurable Readout Error Mitigation in Quantum Workflows“. In: *Electronics* 11.19 (2022) (Zitiert auf S. 109, 120, 151).

[BBL+22b] M. Beisel, J. Barzen, F. Leymann, F. Truger, B. Weder, V. Yussupov. „Patterns for Quantum Error Handling“. In: *Proceedings of the 14th International Conference on Pervasive Patterns and Applications (PATTERNS 2022)*. Xpert Publishing Services (XPS), 2022, S. 22–30 (Zitiert auf S. 87, 109, 120, 151).

[BBL+23] M. Bechtold, J. Barzen, F. Leymann, A. Mandl, J. Obst, F. Truger, B. Weder. „Investigating the effect of circuit cutting in QAOA for the MaxCut problem on NISQ devices“. In: *Quantum Science and Technology* 8.4 (2023), S. 045022 (Zitiert auf S. 151).

[BBLM23] M. Bechtold, J. Barzen, F. Leymann, A. Mandl. „Circuit Cutting with Non-Maximally Entangled States“. In: (2023). arXiv: 2306.12084 [quant-ph] (Zitiert auf S. 151).

[BCMS19] C. D. Bruzewicz, J. Chiaverini, R. McConnell, J. M. Sage. „Trapped-ion quantum computing: Progress and challenges“. In: *Applied Physics Reviews* 6.2 (2019) (Zitiert auf S. 128).

[BDG+22] S. Bravyi, O. Dial, J. M. Gambetta, D. Gil, Z. Nazario. „The future of quantum computing with superconducting qubits“. In: *Journal of Applied Physics* 132.16 (2022) (Zitiert auf S. 151).

[BDP21] S. Brandhofer, S. Devitt, I. Polian. „ArsoNISQ: Analyzing Quantum Algorithms on Near-Term Architectures“. In: *2021 IEEE European Test Symposium (ETS)*. 2021, S. 1–6 (Zitiert auf S. 101).

[BFKW13] S. Barz, J. F. Fitzsimons, E. Kashefi, P. Walther. „Experimental verification of quantum computation“. In: *Nature Physics* 9.11 (2013), S. 727–731 (Zitiert auf S. 109).

[BGT23] C. K.-U. Becker, I.-D. Gheorghe-Pop, N. Tcholtchev. „A Testing Pipeline for Quantum Computing Applications“. In: *2023 IEEE International Conference on Quantum Software (QSW)*. IEEE, 2023, S. 54–59 (Zitiert auf S. 100, 146).

[BIS+18] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, H. Neven. „Characterizing quantum supremacy in near-term devices“. In: *Nature Physics* 14.6 (2018), S. 595–600 (Zitiert auf S. 33).

[BKM+14] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, J. M. Martinis. „Superconducting quantum circuits at the surface code threshold for fault tolerance“. In: *Nature* 508.7497 (2014), S. 500–503 (Zitiert auf S. 128).

[BKS+21] A. Bączkiewicz, B. Kizielewicz, A. Shekhovtsov, J. Wątróbski, W. Sałabun. „Methodical Aspects of MCDM Based E-Commerce Recommender System“. In: *Journal of Theoretical and Applied Electronic Commerce Research* 16.6 (2021), S. 2192–2229 (Zitiert auf S. 75, 107).

[BL19] J. Barzen, F. Leymann. „Quantum humanities: a vision for quantum computing in digital humanities“. In: *SICS Software-Intensive Cyber-Physical Systems* (2019), S. 1–6 (Zitiert auf S. 147).

[BL20] J. Barzen, F. Leymann. „Quantum Humanities: A First Use Case for Quantum-ML in Media Science“. In: Bd. 4. 1. eMedia Gesellschaft für Elektronische Medien mbH, 2020, S. 102–103 (Zitiert auf S. 147).

[BL21] J. Barzen, F. Leymann. „Quantencomputing in den Digital Humanities: innovativ oder übertrieben?“ In: *ZfdG - Zeitschrift für digitale Geisteswissenschaften. Fabrikation von Erkenntnis: Experimente in den Digital Humanities* (2021), S. 1–22 (Zitiert auf S. 147).

[BL22] J. Barzen, F. Leymann. „Continued Fractions and Probability Estimations in Shor's Algorithm: A Detailed and Self-Contained Treatise“. In: *AppliedMath* 2.3 (2022), S. 393–432 (Zitiert auf S. 14, 29, 41).

[BL23] J. Barzen, F. Leymann. „Quantencomputing als Integrationsproblem: Quantenanwendungen sind in der Praxis immer hybride“. In: *Chancen und Risiken von Quantentechnologien*. Springer, 2023, S. 115–123 (Zitiert auf S. 144, 152).

[BLF+21] J. Barzen, F. Leymann, M. Falkenthal, D. Vietz, B. Weder, K. Wild. „Relevance of Near-Term Quantum Computing in the Cloud: A Humanities Perspective“. In: *Cloud Computing and Services Science* 1399 (2021), S. 25–58 (Zitiert auf S. 15, 40–42, 69, 146).

[BM05] J.-P. Brans, B. Mareschal. „Promethee Methods“. In: *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer, 2005, S. 163–186 (Zitiert auf S. 49, 52, 122).

[BMB+22] C. H. Baldwin, K. Mayer, N. C. Brown, C. Ryan-Anderson, D. Hayes. „Re-examining the quantum volume test: Ideal distributions, compiler optimizations, confidence intervals, and scalable resource estimations“. In: *Quantum* 6 (2022), S. 707 (Zitiert auf S. 37).

[BMSS23] D. Bhoumik, R. Majumdar, A. Saha, S. Sur-Kolay. *Distributed Scheduling of Quantum Circuits with Noise and Time Optimization*. 2023. arXiv: `2309.06005 [quant-ph]` (Zitiert auf S. 110).

[BMT+22] M. E. Beverland, P. Murali, M. Troyer, K. M. Svore, T. Hoefler, V. Kliuchnikov, G. H. Low, M. Soeken, A. Sundaram, A. Vaschillo. *Assessing requirements to scale to practical quantum advantage*. 2022. arXiv: `2211.07629 [quant-ph]` (Zitiert auf S. 79).

[Boo12] J. Booth. *Quantum Compiler Optimizations*. 2012. arXiv: `1206.3348 [quant-ph]` (Zitiert auf S. 32, 33).

[Bös06] J. Bös. „Numerical optimization of the thickness distribution of three-dimensional structures with respect to their structural acoustic properties“. In: *Structural and Multidisciplinary Optimization* 32.1 (2006), S. 12–30 (Zitiert auf S. 51, 52).

[Bre01] L. Breiman. „Random forests“. In: *Machine learning* 45.1 (2001), S. 5–32 (Zitiert auf S. 132).

[Bre96] L. Breiman. „Bagging predictors“. In: *Machine Learning* 24.2 (1996), S. 123–140 (Zitiert auf S. 122).

[BSK+21] S. Bravyi, S. Sheldon, A. Kandala, D. C. Mckay, J. M. Gambetta. „Mitigating measurement errors in multiqubit experiments“. In: *Phys. Rev. A* 103 (4 2021), S. 042605 (Zitiert auf S. 109, 120).

[BV97] E. Bernstein, U. Vazirani. „Quantum Complexity Theory“. In: *SIAM Journal on Computing* 26.5 (1997), S. 1411–1473 (Zitiert auf S. 128).

[BY20] R. Blume-Kohout, K. C. Young. „A volumetric framework for quantum computer benchmarks“. In: *Quantum* 4 (2020), S. 362 (Zitiert auf S. 34, 37, 84).

[CAB+21] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, P. J. Coles. „Variational quantum algorithms“. In: *Nature Reviews Physics* 3.9 (2021), S. 625–644 (Zitiert auf S. 144).

[CB18] J. A. Cortese, T. M. Braje. *Loading Classical Data into a Quantum Computer*. 2018. arXiv: `1803.01958 [quant-ph]` (Zitiert auf S. 87).

[CBS+19] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, J. M. Gambetta. „Validating quantum computers using randomized model circuits“. In: *Phys. Rev. A* 100.3 (2019), S. 032328 (Zitiert auf S. 37, 83, 96).

[CBSG17] A. W. Cross, L. S. Bishop, J. A. Smolin, J. M. Gambetta. *Open Quantum Assembly Language*. 2017. arXiv: `1707.03429 [quant-ph]` (Zitiert auf S. 43).

[CCD+23] C. Campbell, F. T. Chong, D. Dahl, P. Frederick, P. Goiporia, P. Gokhale, B. Hall, S. Issa, E. Jones, S. Lee, A. Litteken, V. Omole, D. Owusu-Antwi, M. A. Perlin, R. Rines, K. N. Smith, N. Goss, A. Hashim, R. Naik, E. Younis, D. Lobser, C. G. Yale, B. Huang, J. Liu. *Superstaq: Deep Optimization of Quantum Programs*. 2023. arXiv: `2309.05157 [quant-ph]` (Zitiert auf S. 79, 99).

[CDD+19] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, S. Sivarajah. „On the Qubit Routing Problem“. In: *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*. Bd. 135. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 5:1–5:32 (Zitiert auf S. 33).

[CGS13] S. Corrente, S. Greco, R. Słowiński. „Multiple Criteria Hierarchy Process with ELECTRE and PROMETHEE“. In: *Omega* 41.5 (2013), S. 820–846 (Zitiert auf S. 50).

[Cir23] Cirq Developers. *Cirq*. Version v1.2.0. `https://doi.org/10.5281/zenodo.8161252`. 2023 (Zitiert auf S. 42, 72).

[CJK+16] K. Choi, D.-H. Jang, S.-I. Kang, J.-H. Lee, T.-K. Chung, H.-S. Kim. „Hybrid Algorithm Combing Genetic Algorithm With Evolution Strategy for Antenna Design“. In: *IEEE Transactions on Magnetics* 52.3 (2016), S. 1–4 (Zitiert auf S. 51).

[CMP21] J. A. Cruz-Lemus, L. A. Marcelo, M. Piattini. „Towards a Set of Metrics for Quantum Circuits Understandability“. In: *Quality of Information and Communications Technology*. Springer, 2021, S. 239–249 (Zitiert auf S. 30, 31).

[CN97] I. L. Chuang, M. A. Nielsen. „Prescription for experimental determination of the dynamics of a quantum black box“. In: *Journal of Modern Optics* 44.11-12 (1997), S. 2455–2467 (Zitiert auf S. 39).

[Cop02] D. Coppersmith. *An approximate Fourier transform useful in quantum factoring*. 2002. arXiv: `quant-ph/0201067 [quant-ph]` (Zitiert auf S. 89).

[Cop96] J. O. Coplien. *Software Patterns*. SIGS Books & Multimedia, 1996 (Zitiert auf S. 54, 85).

[CVK18] P. Cerda, G. Varoquaux, B. Kégl. „Similarity Encoding for Learning with Dirty Categorical Variables“. In: *Machine Learning* 107.8–10 (2018), S. 1477–1494 (Zitiert auf S. 47).

[CWV+20] S. Cao, L. Wossnig, B. Vlastakis, P. Leek, E. Grant. „Cost-function embedding and dataset encoding for machine learning with parametrized quantum circuits“. In: *Phys. Rev. A* 101 (5 2020), S. 052309 (Zitiert auf S. 93).

[CZX+18] Z.-Y. Chen, Q. Zhou, C. Xue, X. Yang, G.-C. Guo, G.-P. Guo. „64-qubit quantum circuit simulation“. In: *Science Bulletin* 63.15 (2018), S. 964–971 (Zitiert auf S. 34, 109).

[DF00] G. De'ath, K. E. Fabricius. „Classification and regression trees: a powerful yet simple technique for ecological data analysis". In: *Ecology* 81.11 (2000), S. 3178–3192 (Zitiert auf S. 48).

[DGK14] A. Daskin, A. Grama, S. Kais. „Quantum random state generation with predefined entanglement constraint". In: *International Journal of Quantum Information* 12.05 (2014), S. 1450030 (Zitiert auf S. 95, 96).

[Dod08] Y. Dodge. „Spearman Rank Correlation Coefficient". In: *The Concise Encyclopedia of Statistics*. Springer, 2008, S. 502–505 (Zitiert auf S. 136, 137).

[DS19] D. Duarte, N. Ståhl. „Machine Learning: A Concise Overview". In: *Data Science in Practice*. Springer, 2019, S. 27–58 (Zitiert auf S. 92).

[DYY+20] B. Duan, J. Yuan, C.-H. Yu, J. Huang, C.-Y. Hsieh. „A survey on HHL algorithm: From theory to application in quantum machine learning". In: *Physics Letters A* 384.24 (2020), S. 126595 (Zitiert auf S. 99).

[Edw77] W. Edwards. „How to Use Multiattribute Utility Measurement for Social Decisionmaking". In: *IEEE Transactions on Systems, Man, and Cybernetics* 7.5 (1977), S. 326–340 (Zitiert auf S. 50, 106).

[EWP+19] A. Erhard, J. J. Wallman, L. Postler, M. Meth, R. Stricker, E. A. Martinez, P. Schindler, T. Monz, J. Emerson, R. Blatt. „Characterizing large-scale quantum computers via cycle benchmarking". In: *Nature communications* 10.1 (2019), S. 5347 (Zitiert auf S. 39).

[FBB+23] A. Furutanpey, J. Barzen, M. Bechtold, S. Dustdar, F. Leymann, P. Raith, F. Truger. „Architectural Vision for Quantum Computing in the Edge-Cloud Continuum". In: *2023 IEEE International Conference on Quantum Software (QSW)*. IEEE, 2023, S. 88–103 (Zitiert auf S. 151).

[FBBL14] C. Fehling, J. Barzen, U. Breitenbücher, F. Leymann. „A Process for Pattern Identification, Authoring, and Application". In: *Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP 2014)*. ACM, 2014 (Zitiert auf S. 85).

[FBW18] M. Fingerhuth, T. Babej, P. Wittek. „Open source software in quantum computing". In: *PLOS ONE* 13.12 (2018), S. 1–28 (Zitiert auf S. 33, 41, 42).

[FGGS00] E. Farhi, J. Goldstone, S. Gutmann, M. Sipser. *Quantum Computation by Adiabatic Evolution*. 2000 (Zitiert auf S. 29).

[FJJB18] S. Farshidi, S. Jansen, R. de Jong, S. Brinkkemper. „A Decision Support System for Cloud Service Provider Selection Problem in Software Producing Organizations“. In: *2018 IEEE 20th Conference on Business Informatics (CBI)*. Bd. 01. 2018, S. 139–148 (Zitiert auf S. 48).

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, 2014, S. 367 (Zitiert auf S. 54).

[Fog94] D. Fogel. „An introduction to simulated evolutionary optimization“. In: *IEEE Transactions on Neural Networks* 5.1 (1994), S. 3–14 (Zitiert auf S. 50, 51, 122).

[FS97] Y. Freund, R. E. Schapire. „A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting“. In: *Journal of Computer and System Sciences* 55.1 (1997), S. 119–139 (Zitiert auf S. 122).

[GCA+21] M. Grossi, L. Crippa, A. Aita, G. Bartoli, V. Sammarco, E. Picca, N. Said, F. Tramonto, F. Mattei. *A Serverless Cloud Integration For Quantum Computing*. 2021. arXiv: `2107.02007 [cs.ET]` (Zitiert auf S. 110).

[GK65] G. Golub, W. Kahan. „Calculating the Singular Values and Pseudo-Inverse of a Matrix“. In: *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis* 2.2 (1965), S. 205–224 (Zitiert auf S. 95).

[GKK19] A. Gheorghiu, T. Kapourniotis, E. Kashefi. „Verification of Quantum Computation: An Overview of Existing Approaches“. In: *Theor. Comp. Sys.* 63.4 (2019), S. 715–808 (Zitiert auf S. 109).

[GKS+22] S. S. Gill, A. Kumar, H. Singh, M. Singh, K. Kaur, M. Usman, R. Buyya. „Quantum computing: A taxonomy, systematic review and future directions“. In: *Software: Practice and Experience* 52.1 (2022), S. 66–114 (Zitiert auf S. 29).

[GL14] J. Geldermann, N. Lerche. „Leitfaden zur Anwendung von Methoden der multikriteriellen Entscheidungsunterstützung“. In: *Methode: Promethee* (2014) (Zitiert auf S. 49, 50, 52, 53, 107).

[GLM08] V. Giovannetti, S. Lloyd, L. Maccone. „Quantum Random Access Memory“. In: *Phys. Rev. Lett.* 100 (16 2008), S. 160501 (Zitiert auf S. 89–91).

[Gro96] L. K. Grover. „A fast quantum mechanical algorithm for database search“. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, S. 212–219 (Zitiert auf S. 29, 89, 124, 128).

[GRV+22] J. Garcia-Alonso, J. Rojo, D. Valencia, E. Moguel, J. Berrocal, J. M. Murillo. „Quantum Software as a Service Through a Quantum API Gateway“. In: *IEEE Internet Computing* 26.1 (2022), S. 34–41 (Zitiert auf S. 110).

[GZL+21] M. Guo, Q. Zhang, X. Liao, F. Y. Chen, D. D. Zeng. „A hybrid machine learning framework for analyzing human decision-making through learning preferences“. In: *Omega* 101 (2021), S. 102263 (Zitiert auf S. 106).

[Hau95] R. Haupt. „An introduction to genetic algorithms for electromagnetics“. In: *IEEE Antennas and Propagation Magazine* 37.2 (1995), S. 7–15 (Zitiert auf S. 51).

[HC18] L. E. Heyfron, E. T. Campbell. „An efficient quantum compiler that reduces T count“. In: *Quantum Science and Technology* 4.1 (2018), S. 015004 (Zitiert auf S. 33).

[HH18] M. Hassan, M. Hamada. „Genetic Algorithm Approaches for Improving Prediction Accuracy of Multi-criteria Recommender Systems“. In: *International Journal of Computational Intelligence Systems* 11.1 (2018), S. 146–162 (Zitiert auf S. 51, 122).

[HHL09] A. W. Harrow, A. Hassidim, S. Lloyd. „Quantum Algorithm for Linear Systems of Equations“. In: *Phys. Rev. Lett.* 103 (15 2009), S. 150502 (Zitiert auf S. 91, 92).

[HKP22] T. Hur, L. Kim, D. K. Park. „Quantum convolutional neural network for classical data classification“. In: *Quantum Machine Intelligence* 4.1 (2022) (Zitiert auf S. 92, 94).

[Hol73] J. H. Holland. „Genetic Algorithms and the Optimal Allocation of Trials“. In: *SIAM Journal on Computing* 2.2 (1973), S. 88–105 (Zitiert auf S. 50, 122).

[HOR21] D. Herr, B. Obert, M. Rosenkranz. „Anomaly detection with variational quantum generative adversarial networks“. In: *Quantum Science and Technology* 6.4 (2021), S. 045004 (Zitiert auf S. 143).

[HSST18] T. Häner, D. S. Steiger, K. Svore, M. Troyer. „A software methodology for compiling quantum programs“. In: *Quantum Science and Technology* 3.2 (2018), S. 020501 (Zitiert auf S. 32, 33).

[HW04] G. Hohpe, B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2004 (Zitiert auf S. 76).

[HY81] C.-L. Hwang, K. Yoon. „Methods for Multiple Attribute Decision Making“. In: *Multiple Attribute Decision Making: Methods and Applications A State-of-the-Art Survey*. Springer, 1981, S. 58–191 (Zitiert auf S. 49, 52, 122).

[IBM22] IBM Research. *With fault tolerance the ultimate goal, error mitigation is the path that gets quantum computing to usefulness*. 2022. URL: `https://research.ibm.com/blog/gammabar-for-quantum-advantage` (besucht am 25. 04. 2024) (Zitiert auf S. 38).

[IKLL12] S. Islam, J. Keung, K. Lee, A. Liu. „Empirical prediction models for adaptive resource provisioning in the cloud“. In: *Future Generation Computer Systems* 28.1 (2012), S. 155–162 (Zitiert auf S. 77).

[IRM+21] R. Iten, O. Reardon-Smith, E. Malvetti, L. Mondada, G. Pauvert, E. Redmond, R. S. Kohli, R. Colbeck. *Introduction to UniversalQCompiler*. 2021. arXiv: `1904.01072 [quant-ph]` (Zitiert auf S. 96).

[JAA+22] A. J., A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev, D. Gunter, S. Karra, N. Lemons, S. Lin, A. Malyzhenkov, D. Mascarenas, S. Mniszewski, B. Nadiga, D. O'malley, D. Oyen, S. Pakin, L. Prasad, R. Roberts, P. Romero, N. Santhi, N. Sinitsyn, P. J. Swart, J. G. Wendelberger, B. Yoon, R. Zamora, W. Zhu, S. Eidenbenz, A. Bärtschi, P. J. Coles, M. Vuffray, A. Y. Lokhov. „Quantum Algorithm Implementations for Beginners“. In: *ACM Transactions on Quantum Computing* 3.4 (2022), S. 1–92 (Zitiert auf S. 124).

[JJB+21] P. Jurcevic, A. Javadi-Abhari, L. S. Bishop, I. Lauer, D. F. Bogorin, M. Brink, L. Capelluto, O. Günlük, T. Itoko, N. Kanazawa, A. Kandala, G. A. Keefe, K. Krsulich, W. Landers, E. P. Lewandowski, D. T. McClure, G. Nannicini, A. Narasgond, H. M. Nayfeh, E. Pritchett, M. B. Rothwell, S. Srinivasan, N. Sundaresan, C. Wang, K. X. Wei, C. J. Wood, J.-B. Yau, E. J. Zhang, O. E. Dial, J. M. Chow, J. M. Gambetta. „Demonstration of quantum volume 64 on a superconducting quantum computing system“. In: *Quantum Science and Technology* 6.2 (2021), S. 025020 (Zitiert auf S. 37).

[JWHT21] G. James, D. Witten, T. Hastie, R. Tibshirani. In: *An Introduction to Statistical Learning: with Applications in R*. Springer, 2021, S. 59–128 (Zitiert auf S. 47, 48, 106, 129, 131).

[Kar+20] P. J. Karalekas et al. *PyQuil: Quantum programming in Python*. Version v2.17.0. `https://doi.org/10.5281/zenodo.3631770`. 2020 (Zitiert auf S. 43).

[Kas21] V. Kasirajan. „Fundamentals of Quantum Computing: Theory and Practice“. In: Springer, 2021 (Zitiert auf S. 28–30).

[KIMK22] Y. Kharkov, A. Ivanova, E. Mikhantiev, A. Kotelnikov. *Arline Benchmarks: Automated Benchmarking Platform for Quantum Compilers*. 2022 (Zitiert auf S. 15, 21, 30, 45, 71, 96, 100).

[KLR+08] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, D. J. Wineland. „Randomized benchmarking of quantum gates“. In: *Phys. Rev. A* 77.1 (2008), S. 012307 (Zitiert auf S. 32, 39).

[KVP+24] D. Kremer, V. Villar, H. Paik, I. Duran, I. Faro, J. Cruz-Benito. *Practical and efficient quantum circuit synthesis and transpiling with Reinforcement Learning*. 2024. arXiv: `2405.13196 [quant-ph]` (Zitiert auf S. 99).

[LaR19] R. LaRose. „Overview and Comparison of Gate Level Quantum Software Platforms“. In: *Quantum* 3 (2019), S. 130 (Zitiert auf S. 13, 14, 32, 33, 40–42, 72, 124).

[LB20] F. Leymann, J. Barzen. „The bitter truth about gate-based quantum algorithms in the NISQ era“. In: *Quantum Science and Technology* (2020), S. 1–28 (Zitiert auf S. 14, 29, 30, 32–34, 41, 83, 84, 87, 89–96, 99).

[LB21] F. Leymann, J. Barzen. „Pattern Atlas“. In: Springer International Publishing, 2021, S. 67–76 (Zitiert auf S. 146).

[LBF+20] F. Leymann, J. Barzen, M. Falkenthal, D. Vietz, B. Weder, K. Wild. „Quantum in the Cloud: Application Potentials and Research Opportunities“. In: *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*. SciTePress, 2020, S. 9–24 (Zitiert auf S. 32, 33, 71).

[LBF19] F. Leymann, J. Barzen, M. Falkenthal. „Towards a Platform for Sharing Quantum Software“. In: *Proceedings of the 13th Advanced Summer School on Service Oriented Computing (2019)*. IBM Technical Report (RC25685). IBM Research Division, 2019, S. 70–74 (Zitiert auf S. 13, 69, 146).

[LC20] R. LaRose, B. Coyle. „Robust data encodings for quantum classifiers“. In: *Phys. Rev. A* 102 (3 2020), S. 032420 (Zitiert auf S. 92–94).

[LD21] L. Liu, X. Dou. „QuCloud: A New Qubit Mapping Mechanism for Multi-programming Quantum Computing in Cloud Environment“. In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, S. 167–178 (Zitiert auf S. 100).

[Ley19] F. Leymann. „Towards a Pattern Language for Quantum Algorithms“. In: *Quantum Technology and Optimization Problems, QTOP 2019*. Springer, 2019, S. 218–230 (Zitiert auf S. 14, 29, 30, 41, 54, 69, 85, 87, 89–92, 94, 96, 99, 146).

[LJV+21] T. Lubinski, S. Johri, P. Varosy, J. Coleman, L. Zhao, J. Necaise, C. H. Baldwin, K. Mayer, T. Proctor. *Application-Oriented Performance Benchmarks for Quantum Computing*. 2021 (Zitiert auf S. 37, 84).

[LMR+17] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, C. Monroe. „Experimental comparison of two quantum computing architectures“. In: *Proceedings of the National Academy of Sciences* 114.13 (2017), S. 3305–3310 (Zitiert auf S. 38).

[LQWC13] P. Li, H. Qian, J. Wu, J. Chen. „Sensitivity analysis of TOPSIS method in water quality assessment: I. Sensitivity to the parameter weights“. In: *Environmental Monitoring and Assessment* 185.3 (2013), S. 2453–2461 (Zitiert auf S. 52, 107, 108, 123).

[LR00] F. Leymann, D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000 (Zitiert auf S. 14, 147).

[LvM19] C. Luu, J. von Meding, M. Mojtahedi. „Analyzing Vietnam's national disaster loss database for flood risk assessment using multiple linear regression-TOPSIS“. In: *International Journal of Disaster Risk Reduction* 40 (2019), S. 101153 (Zitiert auf S. 50, 137).

[MAA21] S. Martiel, T. Ayral, C. Allouche. „Benchmarking Quantum Coprocessors in an Application-Centric, Hardware-Agnostic, and Scalable Way“. In: *IEEE Transactions on Quantum Engineering* 2 (2021), S. 1–11 (Zitiert auf S. 38).

[MBB+18] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, A. Kandala, A. Mezzacapo, P. Müller, W. Riess, G. Salis, J. Smolin, I. Tavernelli, K. Temme. „Quantum optimization using variational algorithms on near-term quantum devices“. In: *Quantum Science and Technology* 3.3 (2018), S. 030503 (Zitiert auf S. 21, 36, 83, 100).

[MBC+22] K. Miller, C. Broomfield, A. Cox, J. Kinast, B. Rodenburg. *An Improved Volumetric Metric for Quantum Computers via more Representative Quantum Circuit Shapes*. 2022 (Zitiert auf S. 37).

[MDMN08] D. Maslov, G. W. Dueck, D. M. Miller, C. Negrevergne. „Quantum Circuit Simplification and Level Compaction“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.3 (2008), S. 436–444 (Zitiert auf S. 33).

[Mei20] L. Meier. „Lineare Regression“. In: *Wahrscheinlichkeitsrechnung und Statistik: Eine Einführung für Verständnis, Intuition und Überblick*. Springer, 2020, S. 203–227 (Zitiert auf S. 47).

[MGE12] E. Magesan, J. M. Gambetta, J. Emerson. „Characterizing quantum gates via randomized benchmarking“. In: *Phys. Rev. A* 85.4 (2012), S. 042311 (Zitiert auf S. 39).

[MGJ+12] E. Magesan, J. M. Gambetta, B. R. Johnson, C. A. Ryan, J. M. Chow, S. T. Merkel, M. P. da Silva, G. A. Keefe, M. B. Rothwell, T. A. Ohki, M. B. Ketchen, M. Steffen. „Efficient Measurement of Quantum Gate Error by Interleaved Randomized Benchmarking“. In: *Phys. Rev. Lett.* 109.8 (2012), S. 080505 (Zitiert auf S. 128).

[MHP+23] D. C. McKay, I. Hincks, E. J. Pritchett, M. Carroll, L. C. G. Govia, S. T. Merkel. *Benchmarking Quantum Processor Performance at Scale*. 2023. arXiv: `2311.05933 [quant-ph]` (Zitiert auf S. 37, 38, 109).

[MKF19] K. Mitarai, M. Kitagawa, K. Fujii. „Quantum analog-digital conversion“. In: *Phys. Rev. A* 99 (1 2019), S. 012301 (Zitiert auf S. 91).

[MLD+20] A. J. McCaskey, D. I. Lyakh, E. F. Dumitrescu, S. S. Powers, T. S. Humble. „XACC: a system-level software infrastructure for heterogeneous quantum–classical computing“. In: *Quantum Science and Technology* 5.2 (2020), S. 024002 (Zitiert auf S. 100).

[MLM+19] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, C. H. Alderete. „Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design Insights“. In: *Proceedings of the 46th International Symposium on Computer Architecture*. ISCA '19. ACM, 2019, S. 527–540 (Zitiert auf S. 99).

[MM05] D. Maslov, D. M. Miller. *Comparison of the Cost Metrics for Reversible and Quantum Logic Synthesis*. 2005. arXiv: `quant-ph/0511008 [quant-ph]` (Zitiert auf S. 31).

[MSSD21] D. Mills, S. Sivarajah, T. L. Scholten, R. Duncan. *Application-Motivated, Holistic Benchmarking of a Full Quantum Computing Stack*. 2021 (Zitiert auf S. 31–33, 45, 71, 99).

[MVBS04] M. Mottonen, J. J. Vartiainen, V. Bergholm, M. M. Salomaa. *Transformation of quantum states using uniformly controlled rotations*. 2004. arXiv: `quant-ph/0407010 [quant-ph]` (Zitiert auf S. 93).

[Nat19] National Academies of Sciences, Engineering, and Medicine. *Quantum Computing: Progress and Prospects*. The National Academies Press, 2019 (Zitiert auf S. 13).

[NC11] M. A. Nielsen, I. L. Chuang. *Quantum Computation and Quantum Information*. 10th. Cambridge University Press, 2011 (Zitiert auf S. 28–32, 96).

[NT23] P. D. Nation, M. Treinish. „Suppressing Quantum Circuit Errors Due to System Variability“. In: *PRX Quantum* 4.1 (2023), S. 010327 (Zitiert auf S. 110).

[OAS18] S. Orak, R. A. Arapoğlu, M. A. Sofuoğlu. „Development of an ANN-Based Decision-Making Method for Determining Optimum Parameters in Turning Operation“. In: *Soft Comput.* 22.18 (2018), S. 6157–6170 (Zitiert auf S. 50).

[OBB+23] J. Obst, J. Barzen, M. Beisel, F. Leymann, M. Salm, F. Truger. „Comparing Quantum Service Offerings“. In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Bd. 02. IEEE, 2023, S. 181–184 (Zitiert auf S. 25, 32, 71).

[Ols04] D. Olson. „Comparison of weights in TOPSIS models“. In: *Mathematical and Computer Modelling* 40.7 (2004), S. 721–727 (Zitiert auf S. 50).

[ON23] P. Osypanka, P. Nawrocki. „QoS-Aware Cloud Resource Prediction for Computing Services“. In: *IEEE Transactions on Services Computing* 16.2 (2023), S. 1346–1357 (Zitiert auf S. 77).

[Pat21] S. Pathak. *Intelligent manufacturing*. Springer, 2021 (Zitiert auf S. 51).

[PB11] M. Plesch, C. Č. Brukner. „Quantum-state preparation with universal gate decompositions“. In: *Phys. Rev. A* 83 (3 2011), S. 032302 (Zitiert auf S. 93, 96).

[PBE22] E. Pelofske, A. Bärtschi, S. Eidenbenz. „Quantum Volume in Practice: What Users Can Expect From NISQ Devices“. In: *IEEE Transactions on Quantum Engineering* 3 (2022), S. 1–19 (Zitiert auf S. 36, 37).

[PDF+21] J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, M. S. Allman, C. H. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer, C. Ryan-Anderson, B. Neyenhuis. „Demonstration of the trapped-ion quantum CCD computer architecture“. In: *Nature* 592.7853 (2021), S. 209–213 (Zitiert auf S. 37).

[PFF23] F. Paul, M. Falkenthal, S. Feld. *Clever Design, Unexpected Obstacles: Insights on Implementing a Quantum Boltzmann Machine*. 2023. arXiv: 2301.13705 [quant-ph] (Zitiert auf S. 143).

[Pla24] PlanQK. *PlanQK - Platform and Ecosystem for Quantum-assisted Artificial Intelligence*. 2024. URL: https://platform.planqk.de/community (Zitiert auf S. 13, 146).

[Pow07] M. Powell. „A View of Algorithms for Optimization Without Derivatives“. In: *Mathematics TODAY* 43 (2007) (Zitiert auf S. 52).

[Pow94] M. J. D. Powell. „A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation“. In: *Advances in Optimization and Numerical Analysis*. Springer, 1994, S. 51–67 (Zitiert auf S. 51, 52, 122).

[PPR19] D. K. Park, F. Petruccione, J.-K. K. Rhee. „Circuit-Based Quantum Random Access Memory for Classical Data“. In: *Scientific Reports* 9.1 (2019) (Zitiert auf S. 91).

[Pre12] J. Preskill. *Quantum computing and the entanglement frontier*. 2012 (Zitiert auf S. 29).

[Pre18] J. Preskill. „Quantum Computing in the NISQ era and beyond“. In: *Quantum* 2 (2018), S. 79 (Zitiert auf S. 13, 31, 72, 83, 128, 151).

[PRY+22] T. Proctor, K. Rudinger, K. Young, E. Nielsen, R. Blume-Kohout. „Measuring the capabilities of quantum computers“. In: *Nature Physics* 18.1 (2022), S. 75–79 (Zitiert auf S. 32, 77).

[PSPP21] A. Pellow-Jarman, I. Sinayskiy, A. Pillay, F. Petruccione. „A comparison of various classical optimizers for a variational quantum linear solver“. In: *Quantum Information Processing* 20.6 (2021), S. 1–14 (Zitiert auf S. 52).

[PSW22] C. Piveteau, D. Sutter, S. Woerner. „Quasiprobability decompositions with reduced sampling overhead“. In: *npj Quantum Information* 8.1 (2022), S. 12 (Zitiert auf S. 38).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. „Scikit-learn: Machine learning in Python“. In: *the Journal of machine Learning research* 12 (2011), S. 2825–2830 (Zitiert auf S. 122).

[PW10] O. Pele, M. Werman. „The Quadratic-Chi Histogram Distance Family“. In: *Computer Vision – ECCV 2010*. Springer, 2010, S. 749–762 (Zitiert auf S. 34).

[QBW23a] N. Quetschlich, L. Burgholzer, R. Wille. „Compiler Optimization for Quantum Computing Using Reinforcement Learning“. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, S. 1–6 (Zitiert auf S. 78, 100).

[QBW23b] N. Quetschlich, L. Burgholzer, R. Wille. „MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing“. In: *Quantum* 7 (2023), S. 1062 (Zitiert auf S. 84, 100).

[QBW23c] N. Quetschlich, L. Burgholzer, R. Wille. *MQT Predictor: Automatic Device Selection with Device-Specific Circuit Compilation for Quantum Computing*. 2023. arXiv: 2310.06889 [quant-ph] (Zitiert auf S. 31, 100).

[QBW23d] N. Quetschlich, L. Burgholzer, R. Wille. „Predicting Good Quantum Circuit Compilation Options“. In: *2023 IEEE International Conference on Quantum Software (QSW)*. 2023, S. 43–53 (Zitiert auf S. 78, 100, 142).

[Qis23] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing*. https://doi.org/10.5281/zenodo.2573505. 2023 (Zitiert auf S. 42, 43, 72, 76, 97, 124, 125, 128, 135).

[RB01] R. Raussendorf, H. J. Briegel. „A One-Way Quantum Computer“. In: *Phys. Rev. Lett.* 86.22 (2001), S. 5188–5191 (Zitiert auf S. 29).

[Rig21] Rigetti. *Docs for the Forest SDK*. https://pyquil-docs.rigetti.com/. 2021 (Zitiert auf S. 42, 72, 123, 124).

[RK19] S. Resch, U. R. Karpuzcu. *Benchmarking Quantum Computers and the Impact of Quantum Noise*. 2019. arXiv: 1912.00546 [quant-ph] (Zitiert auf S. 21, 31, 37).

[RP11] E. Rieffel, W. Polak. *Quantum Computing: A Gentle Introduction*. 1st. The MIT Press, 2011 (Zitiert auf S. 13, 28, 29, 125).

[RSMC21] G. S. Ravi, K. N. Smith, P. Murali, F. T. Chong. „Adaptive job and resource management for the growing quantum cloud“. In: (2021) (Zitiert auf S. 42, 101, 110).

[Rus07] N. Russell. *Complexity of control of Borda count elections*. 2007 (Zitiert auf S. 75, 107, 117).

[RWJ+23] S. Ruan, Y. Wang, W. Jiang, Y. Mao, Q. Guan. „VACSEN: A Visualization Approach for Noise Awareness in Quantum Computing“. In: *IEEE Transactions on Visualization and Computer Graphics* 29.1 (2023), S. 462–472 (Zitiert auf S. 101).

[SAH+16] S. G. Sáez, V. Andrikopoulos, M. Hahn, D. Karastoyanova, F. Leymann, M. Skouradaki, K. Vukojevic-Haupt. „Performance and Cost Trade-Off in IaaS Environments: A Scientific Workflow Simulation Environment Case Study“. In: *Cloud Computing and Services Science*. Bd. 581. Communications in Computer and Information Science. Springer, 2016, S. 153–170 (Zitiert auf S. 151).

[SAL16] S. G. Sáez, V. Andrikopoulos, F. Leymann. „Consolidation of Performance and Workload Models in Evolving Cloud Application Topologies“. In: *Proceedings of the 6th International Conference on Cloud Computing and Service Science (CLOSER 2016)*. SciTePress, 2016, S. 160–169 (Zitiert auf S. 77).

[Sał20] Sałabun, Wojciech and Wątróbski, Jarosław and Shekhovtsov, Andrii. „Are MCDA Methods Benchmarkable? A Comparative Study of TOPSIS, VIKOR, COPRAS, and PROMETHEE II Methods“. In: *Symmetry* 12.9 (2020) (Zitiert auf S. 49, 136).

[SALS14] S. G. Sáez, V. Andrikopoulos, F. Leymann, S. Strauch. „Towards Dynamic Application Distribution Support for Performance Optimization in the Cloud“. In: *Proceedings of the 7th IEEE International Conference on Cloud Computing (IEEE CLOUD 2014)*. IEEE Computer Society, 2014, S. 248–255 (Zitiert auf S. 77).

[SB91] M. J. Swain, D. H. Ballard. „Color indexing“. In: *International Journal of Computer Vision* 7.1 (1991), S. 11–32 (Zitiert auf S. 34).

[SBB+20] M. Salm, J. Barzen, U. Breitenbücher, F. Leymann, B. Weder, K. Wild. „The NISQ Analyzer: Automating the Selection of Quantum Computers for Quantum Algorithms“. In: *Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2020)*. Springer, 2020, S. 66–85 (Zitiert auf S. 23, 33, 41, 67, 69, 81, 111, 119).

[SBL+21] M. Salm, J. Barzen, F. Leymann, B. Weder, K. Wild. „Automating the Comparison of Quantum Compilers for Quantum Circuits“. In: *Proceedings of the 15th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2021)*. Springer, 2021, S. 64–80 (Zitiert auf S. 24, 35, 42–45, 67, 81, 83, 97, 111, 118, 119, 127).

[SBLW20] M. Salm, J. Barzen, F. Leymann, B. Weder. „About a Criterion of Successfully Executing a Circuit in the NISQ Era: What $wd \ll 1/\epsilon_{\text{eff}}$ Really Means“. In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020)*. ACM, 2020, S. 10–13 (Zitiert auf S. 24, 35–37, 81, 83, 84).

[SBLW22a] M. Salm, J. Barzen, F. Leymann, B. Weder. „Prioritization of Compiled Quantum Circuits for Different Quantum Computers“. In: *Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2022)*. IEEE, 2022, S. 1258–1265 (Zitiert auf S. 24, 49, 50, 71–73, 97, 103, 105, 107, 111, 128).

[SBLW22b] M. Salm, J. Barzen, F. Leymann, P. Wundrack. „Optimizing the Prioritization of Compiled Quantum Circuits by Machine Learning Approaches“. In: *Proceedings of the 16th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2022)*. Springer, 2022, S. 161–181 (Zitiert auf S. 24, 49–52, 71, 97, 103, 105, 107, 111, 116, 127, 128, 137, 139, 140, 142).

[SBLW23] M. Salm, J. Barzen, F. Leymann, P. Wundrack. „How to Select Quantum Compilers and Quantum Computers Before Compilation“. In: *Proceedings of the 13th International Conference on Cloud Computing and Services Science (CLOSER 2023)*. SciTePress, 2023, S. 172–183 (Zitiert auf S. 24, 47, 67, 71, 107, 111, 114, 116, 122, 127, 129, 130, 132).

[SBM05] V. V. Shende, S. S. Bullock, I. L. Markov. „Synthesis of quantum logic circuits“. In: *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*. ASP-DAC '05. Association for Computing Machinery, 2005, S. 272–275 (Zitiert auf S. 93).

[SBSW20] M. Schuld, A. Bocharov, K. M. Svore, N. Wiebe. „Circuit-centric quantum classifiers“. In: *Phys. Rev. A* 101 (3 2020), S. 032308 (Zitiert auf S. 92).

[SCPP22] M. A. Serrano, J. A. Cruz-Lemus, R. Perez-Castillo, M. Piattini. „Quantum Software Components and Platforms: Overview and Quality Assessment“. In: *ACM Comput. Surv.* 55.8 (2022) (Zitiert auf S. 40–42).

[SCZ16] R. S. Smith, M. J. Curtis, W. J. Zeng. *A Practical Quantum Instruction Set Architecture*. 2016. arXiv: `1608.03355 [quant-ph]` (Zitiert auf S. 42, 43).

[SDC+20] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, R. Duncan. „t|ket>: A retargetable compiler for NISQ devices“. In: *Quantum Science and Technology* 6 (2020) (Zitiert auf S. 14, 15, 30, 32, 33, 44, 45, 72, 79, 97, 99, 124, 125, 128, 133, 135).

[SDH+14] L. Sun, H. Dong, F. K. Hussain, O. K. Hussain, E. Chang. „Cloud service selection: State-of-the-art and future research directions“. In: *Journal of Network and Computer Applications* 45 (2014), S. 134–150 (Zitiert auf S. 48).

[Sho95] P. W. Shor. „Scheme for reducing decoherence in quantum computer memory“. In: *Phys. Rev. A* 52.4 (1995), R2493–R2496 (Zitiert auf S. 31, 32).

[Sho97] P. W. Shor. „Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer“. In: *SIAM Journal on Computing* 26.5 (1997), S. 1484–1509 (Zitiert auf S. 29, 89, 128).

[SHT18] D. S. Steiger, T. Häner, M. Troyer. „ProjectQ: an open source software framework for quantum computing“. In: *Quantum* 2 (2018), S. 49 (Zitiert auf S. 45).

[SKF+13] M. Suchara, J. Kubiatowicz, A. Faruque, F. T. Chong, C. Lai, G. Paz. „QuRE: The Quantum Resource Estimator toolbox“. In: *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE Computer Society, 2013, S. 419–426 (Zitiert auf S. 42, 79).

[SM05] V. V. Shende, I. L. Markov. *Quantum Circuits for Incompletely Specified Two-Qubit Operators*. 2005. arXiv: `quant-ph/0401162 [quant-ph]` (Zitiert auf S. 92).

[Sod18] B. Sodhi. *Quality Attributes on Quantum Computing Platforms*. 2018. arXiv: `1803.07407 [cs.SE]` (Zitiert auf S. 32, 46).

[SP18] M. Schuld, F. Petruccione. *Supervised Learning with Quantum Computers*. Quantum Science and Technology. Springer, 2018 (Zitiert auf S. 89, 90, 92, 94, 99).

[Spe61] C. Spearman. „The Proof and Measurement of Association Between Two Things.“ In: *Studies in individual differences: The search for intelligence.* (1961), S. 45–58 (Zitiert auf S. 136).

[SSCQ18] M. Y. Siraichi, V. F. d. Santos, S. Collange, F. M. Quintão Pereira. „Qubit Allocation“. In: *CGO 2018 - International Symposium on Code Generation and Optimization*. 2018, S. 1–12 (Zitiert auf S. 33).

[SSP16] M. Schuld, I. Sinayskiy, F. Petruccione. „Prediction by linear regression on a quantum computer“. In: *Phys. Rev. A* 94.2 (2016), S. 022342 (Zitiert auf S. 47).

[SSS17] Y. Shikhar, V. P. Singh, R. Srivastava. „Comparative analysis of distance metrics for designing an effective content-based image retrieval system using colour and texture features“. In: *International Journal of Image, Graphics and Signal Processing* 9.12 (2017), S. 58 (Zitiert auf S. 34).

[SWS15] Y. R. Sanders, J. J. Wallman, B. C. Sanders. „Bounding quantum gate error rate based on reported average fidelity“. In: *New Journal of Physics* 18.1 (2015), S. 012002 (Zitiert auf S. 32).

[SZR16] E. A. Sete, W. J. Zeng, C. T. Rigetti. „A functional architecture for scalable quantum computing“. In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 2016, S. 1–6 (Zitiert auf S. 32, 35, 40, 98).

[TBG17] K. Temme, S. Bravyi, J. M. Gambetta. „Error Mitigation for Short-Depth Quantum Circuits“. In: *Phys. Rev. Lett.* 119.18 (2017), S. 180509 (Zitiert auf S. 38).

[TGO+22] T. Tomesh, P. Gokhale, V. Omole, G. S. Ravi, K. N. Smith, J. Viszlai, X.-C. Wu, N. Hardavellas, M. R. Martonosi, F. T. Chong. „SupermarQ: A Scalable Quantum Benchmark Suite“. In: *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2022, S. 587–603 (Zitiert auf S. 31, 79, 142).

[TM19] H. Thapliyal, E. Muñoz-Coreas. „Design of Quantum Computing Circuits“. In: *IT Professional* 21.6 (2019), S. 22–26 (Zitiert auf S. 31).

[TQ18] S. S. Tannu, M. K. Qureshi. *A Case for Variability-Aware Policies for NISQ-Era Quantum Computers*. 2018. arXiv: 1805.10224 [quant-ph] (Zitiert auf S. 38).

[TQ19a] S. S. Tannu, M. K. Qureshi. „Mitigating Measurement Errors in Quantum Computers by Exploiting State-Dependent Bias“. In: MICRO '52. Association for Computing Machinery, 2019, S. 279–290 (Zitiert auf S. 32, 38, 133).

[TQ19b] S. S. Tannu, M. K. Qureshi. „Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers“. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. ACM, 2019, S. 987–999 (Zitiert auf S. 21, 31–33, 38, 128, 133).

[Unr95] W. G. Unruh. „Maintaining coherence in quantum computers“. In: *Phys. Rev. A* 51.2 (1995), S. 992–997 (Zitiert auf S. 32).

[Vap95] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995 (Zitiert auf S. 47).

[VBE96] V. Vedral, A. Barenco, A. Ekert. „Quantum networks for elementary arithmetic operations“. In: *Phys. Rev. A* 54 (1 1996), S. 147–153 (Zitiert auf S. 87).

[VBL+21] D. Vietz, J. Barzen, F. Leymann, B. Weder, V. Yussupov. „An Exploratory Study on the Challenges of Engineering Quantum Applications in the Cloud“. In: *Proceedings of the 2nd Quantum Software Engineering and Technology Workshop (Q-SET 2021) co-located with IEEE International Conference on Quantum Computing and Engineering (QCE21)*. CEUR Workshop Proceedings, 2021, S. 1–12 (Zitiert auf S. 32, 71, 124).

[VBLW21] D. Vietz, J. Barzen, F. Leymann, K. Wild. „On Decision Support for Quantum Application Developers: Categorization, Comparison, and Analysis of Existing Technologies“. In: *Computational Science – ICCS 2021*. Springer, 2021, S. 127–141 (Zitiert auf S. 40, 41, 72).

[VBLW22] D. Vietz, J. Barzen, F. Leymann, B. Weder. „Splitting Quantum-Classical Scripts for the Generation of Quantum Workflows“. In: *Proceedings of the 26th Conference on Enterprise Design, Operations, and Computing (EDOC 2022)*. Springer, 2022, S. 255–270 (Zitiert auf S. 14, 147).

[VGN+16] M. Verma, G. R. Gangadharan, N. C. Narendra, R. Vadlamani, V. Inamdar, L. Ramachandran, R. N. Calheiros, R. Buyya. „Dynamic resource demand prediction and allocation in multi-tenant service clouds“. In: *Concurrency and Computation: Practice and Experience* 28.17 (2016), S. 4429–4442 (Zitiert auf S. 77).

[VM00] D. Ventura, T. Martinez. „Quantum associative memory“. In: *Information Sciences* 124.1 (2000), S. 273–296 (Zitiert auf S. 87–89, 182).

[WBB+20] M. Weigold, J. Barzen, U. Breitenbücher, M. Falkenthal, F. Leymann, K. Wild. „Pattern Views: Concept and Tooling of Interconnected Pattern Languages“. In: *Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2020)*. Springer International Publishing, 2020, S. 86–103 (Zitiert auf S. 146).

[WBBL22] B. Weder, J. Barzen, M. Beisel, F. Leymann. „Analysis and Rewrite of Quantum Workflows: Improving the Execution of Hybrid Quantum Algorithms“. In: *Proceedings of the 12th International Conference on Cloud Computing and Services Science (CLOSER 2022)*. SciTePress, 2022, S. 38–50 (Zitiert auf S. 152).

[WBBL23] B. Weder, J. Barzen, M. Beisel, F. Leymann. „Provenance-Preserving Analysis and Rewrite of Quantum Workflows for Hybrid Quantum Algorithms“. In: *SN Computer Science* 4.233 (2023), S. 1–19 (Zitiert auf S. 152).

[WBL+20] B. Weder, J. Barzen, F. Leymann, M. Salm, D. Vietz. „The Quantum Software Lifecycle“. In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020)*. ACM, 2020, S. 2–9 (Zitiert auf S. 25, 41, 146).

[WBL+21] B. Weder, J. Barzen, F. Leymann, M. Salm, K. Wild. „QProv: A provenance system for quantum computing“. In: *IET Quantum Communication* 2.4 (2021), S. 171–181 (Zitiert auf S. 23, 25, 72, 73, 98, 105, 112, 113).

[WBLS20] M. Weigold, J. Barzen, F. Leymann, M. Salm. „Data Encoding Patterns For Quantum Algorithms“. In: *Proceedings of the 27th Conference on Pattern Languages of Programs (PLoP '20)*. HILLSIDE, 2020, S. 1–11 (Zitiert auf S. 14, 24, 30, 41, 54, 81, 84–86, 88, 91, 94, 96).

[WBLS21a] B. Weder, J. Barzen, F. Leymann, M. Salm. „Automated Quantum Hardware Selection for Quantum Workflows“. In: *Electronics* 10.8 (2021), S. 1–18 (Zitiert auf S. 25, 147).

[WBLS21b] M. Weigold, J. Barzen, F. Leymann, M. Salm. „Encoding patterns for quantum algorithms“. In: *IET Quantum Communication* 2.4 (2021), S. 141–152 (Zitiert auf S. 25, 54, 81, 85–87, 90, 92, 95, 99).

[WBLS21c] M. Weigold, J. Barzen, F. Leymann, M. Salm. „Expanding Data Encoding Patterns For Quantum Algorithms“. In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2021, S. 95–101 (Zitiert auf S. 25, 54, 81, 85, 86, 90).

[WBLV21] M. Weigold, J. Barzen, F. Leymann, D. Vietz. „Patterns for Hybrid Quantum Algorithms“. In: *Proceedings of the 15th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2021)*. Springer, 2021, S. 34–51 (Zitiert auf S. 29, 69, 87, 146, 152).

[WBLV22] B. Weder, J. Barzen, F. Leymann, D. Vietz. „Quantum Software Development Lifecycle“. In: *Quantum Software Engineering*. Springer, 2022, S. 61–83 (Zitiert auf S. 14, 41, 109, 120, 146).

[WBLW20] B. Weder, U. Breitenbücher, F. Leymann, K. Wild. „Integrating Quantum Computing into Workflow Modeling and Execution“. In: *Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2020)*. IEEE Computer Society, 2020, S. 279–291 (Zitiert auf S. 14, 147).

[Wed24] B. Weder. „Workflow-basierte Modellierung, Ausführung und Überwachung hybrider Quantenanwendungen“. Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2024 (Zitiert auf S. 113).

[WJZ+19] J. Wątróbski, J. Jankowski, P. Ziemba, A. Karczmarczyk, M. Zioło. „Generalised framework for multi-criteria method selection“. In: *Omega* 86 (2019), S. 107–124 (Zitiert auf S. 48, 49).

[WJZ+24] J. Wątróbski, J. Jankowski, P. Ziemba, A. Karczmarczyk, M. Zioło. *MCDA Method Selection Tool*. 2024. URL: `http://mcda.it` (Zitiert auf S. 49).

[WJZZ09] J.-J. Wang, Y.-Y. Jing, C.-F. Zhang, J.-H. Zhao. „Review on multi-criteria decision analysis aid in sustainable energy decision-making“. In: *Renewable and Sustainable Energy Reviews* 13.9 (2009), S. 2263–2278 (Zitiert auf S. 50, 75, 107).

[WLC+22] H. Wang, P. Liu, J. Cheng, Z. Liang, J. Gu, Z. Li, Y. Ding, W. Jiang, Y. Shi, X. Qian, D. Z. Pan, F. T. Chong, S. Han. *QuEst: Graph Transformer for Quantum Circuit Reliability Estimation*. 2022. arXiv: `2210.16724 [quant-ph]` (Zitiert auf S. 78).

[WM05] C. J. Willmott, K. Matsuura. „Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance“. In: *Climate research* 30.1 (2005), S. 79–82 (Zitiert auf S. 74, 129).

[WPJ+21] A. Wack, H. Paik, A. Javadi-Abhari, P. Jurcevic, I. Faro, J. M. Gambetta, B. R. Johnson. *Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers*. 2021 (Zitiert auf S. 38).

[WYW+22] J. D. Whitfield, J. Yang, W. Wang, J. T. Heath, B. Harrison. *Quantum Computing 2022*. 2022. arXiv: `2201.09877 [quant-ph]` (Zitiert auf S. 128).

[YBS17] B. Yildiz, J. Bilbao, A. Sproul. „A review and analysis of regression and machine learning models on commercial building electricity load forecasting“. In: *Renewable and Sustainable Energy Reviews* 73 (2017), S. 1104–1122 (Zitiert auf S. 129).

[YIV16] F. Yan, A. M. Iliyasu, S. E. Venegas-Andraca. „A survey of quantum image representations“. In: *Quantum Information Processing* 15.1 (2016), S. 1–35 (Zitiert auf S. 94).

[ZPW19] A. Zulehner, A. Paler, R. Wille. „An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.7 (2019), S. 1226–1236 (Zitiert auf S. 32, 33, 96).

[ZWD+20] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu, P. Hu, X.-Y. Yang, W.-J. Zhang, H. Li, Y. Li, X. Jiang, L. Gan, G. Yang, L. You, Z. Wang, L. Li, N.-L. Liu, C.-Y. Lu, J.-W. Pan. „Quantum computational advantage using photons“. In: *Science* (2020) (Zitiert auf S. 13).

Alle Links wurden zuletzt am 25.04.2024 geprüft.

Abbildungsverzeichnis

Tabellenverzeichnis