Taxonomy of migration scenarios for Qiskit refactoring using LLMs

José Manuel Suárez, Luís Mariano Bibbó, Joaquín Bogado, and Alejandro Fernandez

Universidad Nacional de La Plata, Facultad de Informática
Laboratorio de Investigación y Formación en Informática Avanzada (LIFIA), Argentina

Abstract. As quantum computing advances, quantum programming libraries' heterogeneity and steady evolution create new challenges for software developers. Frequent updates in software libraries break working code that needs to be refactored, thus adding complexity to an already complex landscape. These refactoring challenges are, in many cases, fundamentally different from those known in classical software engineering due to the nature of quantum computing software. This study addresses these challenges by developing a taxonomy of quantum circuit's refactoring problems, providing a structured framework to analyze and compare different refactoring approaches. Large Language Models (LLMs) have proven valuable tools for classic software development, yet their value in quantum software engineering remains unexplored. This study uses LLMs to categorize refactoring needs in migration scenarios between different Qiskit versions. Qiskit documentation and release notes were scrutinized to create an initial taxonomy of refactoring required for migrating between Qiskit releases. Two taxonomies were produced: one by expert developers and one by an LLM. These taxonomies were compared, analyzing differences and similarities, and were integrated into a unified taxonomy that reflects the findings of both methods. By systematically categorizing refactoring challenges in Qiskit, the unified taxonomy is a foundation for future research on AI-assisted migration while enabling a more rigorous evaluation of automated refactoring techniques. Additionally, this work contributes to quantum software engineering (QSE) by enhancing software development workflows, improving language compatibility, and promoting best practices in quantum programming. This research marks the first step in a broader effort to assess various refactoring strategies, ultimately guiding the development of AI-powered tools to support quantum software engineers.

Keywords: Quantum Computing (QC), Quantum Software Engineering (QSE), Large Language Models (LLMs), Generative AI, Qiskit, Migration Code.

1 Introduction

The recent advancements in quantum computing (QC) [Preskill, 2018] underscore the necessity for robust methodologies within Quantum Software Engineering (QSE) [Jiménez-Navajas et al., 2025]. Consequently, well-established topics in classical Software Engineering (SE)—such as code generation [Jin et al., 2025] [Asif et al., 2025] [Dupuis et al., 2024], refactoring [Zhao, 2023] [Tsantalis et al., 2022], metric definition
[Nation et al., 2025] [Quetschlich et al., 2023] [Zhao, 2021], algorithmic strategy design [Xu et al., 2018], development stacks [Guo et al., 2023], debugging and testing strategies [Ramalho et al., 2024] [Ali et al., 2021] [Li et al., 2020], and languages and compilers
[Amy and Gheorghiu, 2020] [Sivarajah et al., 2021]—must now be re-examined within the QSE framework, demanding more sophisticated approaches.

Specifically, the complexity of processes such as code migration and the analysis of refactoring scenarios, despite being widely studied topics within the framework of SE, require a novel approach. Such is the case with the challenges present in highly dynamic development ecosystems like Qiskit, whose evolution imposes a release schedule that often affects the functionality of algorithms. In addition, its proximity to QC as an interdisciplinary development field adds the difficulty that not all those involved are experts in SE areas.

At the same time, the rise of generative AIs [Weisz et al., 2024] [Bengesi et al., 2023] and large language models (LLMs) in particular [OpenAI, 2024]
[Gemini, 2024] [Rozière et al., 2024] presents an unprecedented opportunity for automation and optimization of code migration and refactoring processes. However, their full integration into the QSE development cycle [Murillo et al., 2025] is still in early stages, with unresolved challenges, especially in terms of correctness and reliability.

The relevance of our work revolves around the development of a resolutive and at the same time replicable hybrid methodology that integrates SE strategies and the use of LLMs, based on the analysis of migration and refactoring scenarios in Qiskit, in order to assess their feasibility and summarize their key factors and implications. Complementarily, this study aims to be the initial step toward the development of tools that enhance workflow agility and reduce the technological gap through the early adoption of more recent and effective functionalities in the QSE domain.

The evolution of QC has been directly linked to the accelerated maturity of quantum development environments such as Qiskit [1] [Javadi-Abhari et al., 2024],
Cirq [Isakov et al., 2021], Quipper [Green et al., 2013], Q# [Svore et al., 2018], PennyLane[2], among others; they have played a fundamental role regarding accessibility

[1] IBM Qiskit official site - https://www.ibm.com/quantum/qiskit
[2] Xanadu Pennylane - https://docs.pennylane.ai/en/stable/

and quality in the development, simulation, execution, and deployment process of quantum algorithms, increasing their versatility and successfully integrating with external libraries and different quantum technologies. In this work, we will particularly focus on the Qiskit ecosystem, launched in 2017 by IBM, an open-source environment compatible with Python, backed by a broad community, which supports the simulation and real execution of quantum algorithms, integration with external environments and extensions (Qiskit extensions), platforms and hardware architectures, support for Quantum Machine Learning (QML) [Biamonte et al., 2017] and hybrid execution, among other features, making it one of the most relevant development ecosystems today. [3]

The structure of this work proceeds as follows: in Section 2 addresses related work, allowing us to more precisely establish the current state of research; Section 3 then focuses on a detailed description of the experimental methodology used; in Section 4 we present the preliminary results; Section 5 outlines the main discussions in the field, concluding in Section 6 with some conclusions and future work and research guidelines that should follow in the short and medium term.

2 Related work

We can subdivide the works analyzed in this section according to the use of automatic generation tools and based on their relation to QC, in addition to a handful of works that combine both subgroups:

At the intersection of automatic code migration using LLMs, we can mention recent significant efforts [Almeida et al., 2024] where the work related to prompts engineering is significant. Even if some of the test cases involve some subjectivity that could introduce biases and unreliable conclusions, they allow pondering the implication of prompts in the metrics obtained; on the other hand [Sahoo et al., 2024] provides a very complete review on prompt engineering and the most relevant associated works, considering query techniques, contextual refinement and precision on the associated metrics, as well as prompt iteration and rephrasing [Deng et al., 2024]. Other works aligned with our research promote the generation of surpassing LLMs [Cordeiro et al., 2024] focused on the Java language, although our case study is more complex due to its recent expansion, high dynamism and limited code sources, in addition to difficulties inherent to the quantum paradigm and its distinctive properties.

At the intersection of QC and migration, we consider the work of [Zhao, 2023], focused on the challenges of refactoring in quantum programs on Q#, although limited to maintainability and efficiency criteria and not so closely related to the migration process between versions; likewise, it proposes a catalog based on the

[3] IBM Quantum Documentation (https://docs.quantum.ibm.com/)

analysis of algorithms, similar to our taxonomy, and just as their proposal of *QSharp Refactoring Tool*, our work aims at the construction of a hybrid automated migration tool.

We also consider works related to Qiskit [Dupuis et al., 2024] because of its closeness to the framework we are studying, in addition to some few works that relate LLMs strategies together with quantum code migration ([Asif et al., 2025]) where RAG [Lewis et al., 2021] is used for a code generation wizard over PennyLane and evaluation metrics. While these works address LLMs code generation performance and evaluation, do not address the specifics of version migration and the refactoring issues that arose from them.

3 Methodology

3.1 General description

The methodological structure of our experiment is divided into three fundamental stages: the first consists of the collection of prior and necessary supporting information to establish the experiment, then a forked into a manual and automated LLM assisted stages for the elaboration of a taxonomy of migration scenarios on Qiskit. Each taxonomy section is Qiskit version specific. The third stage relates to the comparison of the taxonomies obtained in the previous stage. See Figure 1.

3.2 Manual elaboration of taxonomy of migration scenarios

For the elaboration of the taxonomies we used the official sources of IBM Qiskit exclusively. We consider this documentation authoritative, accurate and detailed. This source of documentation maintains high precision in terms of API changes, obsolescence and deprecation warning flows, alternatives, recommendations, accompanied by concrete examples and best practice guidelines.

In this regard, we consider two main official sources:

– Qiskit release notes. [4]
– Qiskit changelog. [5]

We consider secondary sources in the manual stage to allow for disambiguation or additional explanation in cases we saw fit. These secondary sources are:

– Qiskit Leatest updates. [6]

[4] Qiskit SDK release notes (https://docs.quantum.ibm.com/api/qiskit/release-notes)
[5] Qiskit Changelog (https://github.com/qiskit/qiskit/releases)
[6] Qiskit Leatest updates (https://docs.quantum.ibm.com/guides/latest-updates)

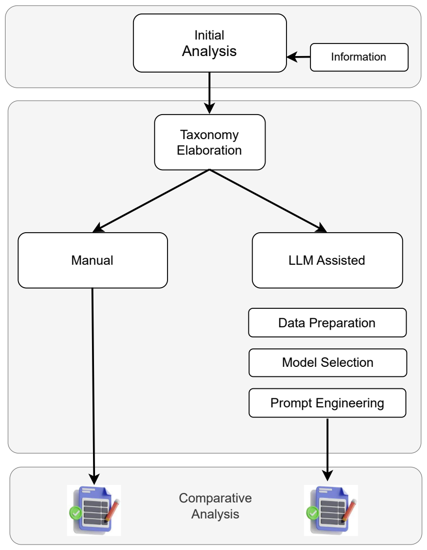


Fig. 1. Experimental flow diagram

- Qiskit Documentation GitHub. [7]
- Qiskit Migration guides. [8]
- Qiskit release summary. [9]
- Youtube Qiskit chanel. [10]

Regarding the information supported by the taxonomy, it seemed important to classify each migration scenario based on its functionality or objective. Many times these categories have arisen from the documentation itself and others based on well-known areas of refactoring within the SE field [Zhao, 2023], see Table 1.

Another dimension that the taxonomy addresses is the degree of difficulty that each migration scenario entails. Although this concept is relatively difficult to evaluate due to its subjectivity, we made an effort to apply it as accurately as possible. We consider lines of code required, time needed to implement the refactoring, atomicity

[7] Qiskit Documentation GitHub (https://github.com/Qiskit/documentation/tree/main)
[8] Qiskit Migration guides ((https://docs.quantum.ibm.com/migration-guides)
[9] Qiskit release summary (https://www.ibm.com/quantum/blog/qiskit-1-0-release-summary)
[10] Youtube Qiskit chanel (https://www.youtube.com/qiski)

Table 1. Manual taxonomy - Categorization of scenarios

Category	**Possible values**	**Description**
Structure	Reorganization, Package, Module, Inheritance, etc.	Modular ecosystem reorganization, functionality reassignment, new modules or deprecations, etc.
Language	Python, Rust, C++	Scenario where the underlying language has changed, usually for performance reasons.
Implication	Primitive, Gate, Algorithm, Transpilation, Organization, Parameterization, Class, Method	Degree of relationship with the artifact affected.
Module	Q-Aer, Q-Aqua, Q-Terra, Q-Ignis, Q-IBMQuantum, Q-Experiments, Q-Chemistry, Q-Optimization, Q-Finance, Q-ML, Q-Nature, etc.	Qiskit ecosystem module affected.
Type	Increased functionality, upgrade, deprecation, bug fixes	Categorization from release notes.
SE Refactoring	Usability, extensibility, readability, modularity, security, flexibility, etc.	Classic software engineering affectations.

or chaining of changes, and complexity related to QSE specifics. This classification is related to the impact of the change from the perspective of a developer i.e.: with role migrator, associated to the question: *"how complex is it to make progress on the migration scenario?"*, see Table 2.

Table 2. Manual taxonomy - Categorization of degrees of difficulty

Category	**Description**
Minimal	No migration difficulties. e.g.: internal performance optimization.
Low	Can be migrated without significant effort, little time and code consumed. e.g.: new method parameterization.
Moderate	Involves considerable effort. e.g.: change in simulation flow or job execution.
High	Implementing the change is of significant difficulty, requiring chained changes on different sections of the code, considerable time and test case execution. e.g.: new module with replacement functions.

We considered relevant to evaluate the degree of relationship that each scenario maintains with the field of classical and quantum software engineering. This allows us to weight each version in relation to its QSE impact and lays the foundations for a tool that considers the migration impact and QSE relevance. See Table 3.

Table 3. Manual taxonomy - Degree of impact on SE and/or QSE

Category	**Description**
Classic Software Engineering (SE)	The scenario affects aspects linked to classic software engineering. e.g.: modularity, extensibility, etc.
Quantum Software Engineering (QSE)	The scenario has implications on QC related issues. e.g.: transpilation, gates, primitives, simulation, integration with external services, etc.

To conclude, the general outline of the manual taxonomy includes the following columns. See Table 4.: We considered the version updates 0.46.0 and 1.0.0 exhaus-

Table 4. Manual taxonomy - classification dimensions

Column	**Description**
Category	Type of each migratory scenario.
Migration Flow	Source and Target of the version upgrade associated with the scenario.
Summary	Brief description of the scenario.
Artifacts	Summary of keywords of the artifacts involved.
Example code in source version	Python example code in the source version of the flow or previous versions.
Example code in target version	Python example code in the target version of the flow or later.
Degree of Difficulty	Complexity weighting involved in the migration scenario.
Degree of impact in SE/QSE	Relationship of the migrating scenario to aspects of classical software engineering and/or associated with quantum software engineering.
References	Links to the authoritative sources of the scenario, either primary or secondary.

tively. Most of the changes between minor versions do not contain rich refactoring scenarios compared to the mentioned update. We exclude *bug-fixes*, *patch* and *prelude* releases due to these changes do not impact on any refactoring scenario. The update from 0.46.0 to 1.0.0 represents a major version bump[11]. We analyzed 13 version updates, with special attention to the releases 0.46.0 and 1.0.0.

[11] At the time of writing this article March 2025 where v2.0.0 is in development.

3.3 Automatic elaboration of taxonomy of migration scenarios

For the automatic taxonomy building stage, we decided to initially perform local tests by downloading models Google gemma and DeepSeek using the LMStudio application [12]:

- **Google gemma-3-27b-it-GGUF** [13], released in March 2025. In principle, with high accuracy and efficient management of hardware resources that enable its execution on a wide range of devices, the possibility of extended token windows that enable the processing of large amounts of text in a single prompt (128k), allowing more flexible and extensible work approaches. In short, it is a tool that is currently in the state of the art, in terms of lightweight open source models.
- **DeepSeek-R1-Distill-Qwen-32b-GGUF** [14], a distilled Qween-32B model fine tuned with the outputs of DeepSeek-r1. This model is a 32 billion parameter model, available under the Apache 2.0 license. It support 32k token context.

We used the quantized to 8 bits version of the models. We loaded the models in a computer with a single AMD Ryzen 9 7950X3D 16-Core Processor, with 128 GB of RAM, and 2x NVIDIA RTX 3090 24 GB VRAM. With GPU offloading the models achieve between 10 to 20 tokens per second.

This initial phase allowed us to improve the prompts without the need monetary expenses, while establishing a baseline prompts for the commercial models. We tested four of the latest models available commercially: OpenAI's **gpt-4o** [15], OpenAI's **gpt-4o-mini** [16], **DeepSeek-V3** [17] and **DeepSeek-R1** [18]. The models where instructed to generate a taxonomy using markdown with the same structure described in the previous section, over the same Qiskit versions. We used few shot learning to enforce the format of the desired output.

As previously mentioned, among the main axes of our experiment are the replicability, accessibility and improvement of the experiment through the GitHub repository and the development of a script that allows the parameterization of the test language models, internal parametric configurations, as well as data traceability through the download of collected data and stages of verification of the complete observability of source data by the model, which are the same as those considered by the manual taxonomy generation agent. In order to guaranty the replicability, we wrote a series of Python scripts that allows us to reproduce the experiments. The

[12] https://lmstudio.ai/
[13] https://docs.api.nvidia.com/nim/reference/google-gemma-3-27b-it
[14] https://huggingface.co/bartowski/DeepSeek-R1-Distill-Qwen-32B-GGUF
[15] https://platform.openai.com/docs/models/gpt-4o
[16] https://platform.openai.com/docs/models/gpt-4o-mini
[17] https://api-docs.deepseek.com/news/news250325
[18] https://api-docs.deepseek.com/news/news250120

scripts tasks are divided into ”documentation extraction”, ”documentation verification” and, ”model access”. The document extraction retrieves the documentation from the primary source and is parametrized by the version number and language [19]. The verification script checks the correctness of the retrieval of documentation based on a manual retrieval example. The model access script allows for model selection programmatically, while allows experiments automation using the same prompts to test different LLMs.

The prompt structure contains a system part describing the task, that is to generate the taxonomy in an specific markdown format with few shot example, defines the column names and expected content, and also include guidelines for the output, i.e.: to restrict scenario grouping, discourage line braking in row lines, avoid replicated scenarios, etc. The user prompt includes a one-shot example of a row and the information retrieved by the document extraction script for a specific Qiskit version.

A Python script was developed to trigger the stages described above. This script also logs all the information returned by the different models and group it by the target version. We ensure that the information accessed by the LLMs is the same to that used to generate the manual taxonomy.

The scripts and examples are accessible in a GitHub repository, to ensure experimental replicability [20].

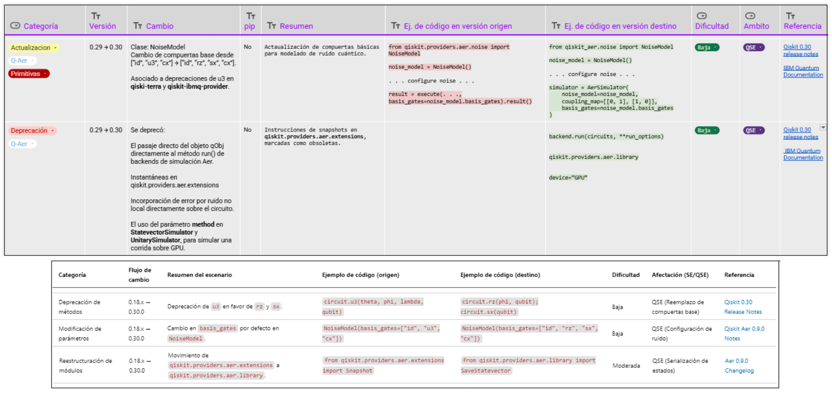
4 Preliminary results

Initial tests didn't produce satisfactory results. Only by refining the prompt structure and wording the models started to return consistent and well formatted output. For example the first results return very few rows, with very few details and incomplete examples and empty cells or duplicate rows. However, when a refined prompt were used, the number of rows tend to match or exceed the manual taxonomy, as well as the level of details and examples without introducing the manual constrains, i.e: ask for a number of rows.

There was a clear quality improve produced by the introduction of the text of the documentation inside the user prompt. However this methodology is constrained by the context length of the model measured in tokens. This is not a limitation given that the context for modern LLMs ranges from 32K to 128K tokens, while the most extensive Qiskit documentation is about 20K tokens. See Fig. 4. The mentioned context length limitation can be walk around by the uses techniques such as RAG or context summarization.

[19] The Qiskit documentation is localized automatically based on browser preferences of the user. We used the Spanish and English versions.

[20] GitHub Repository(https://github.com/jwackito/qiskit_llm_experiment)



Categoría	Versión	Cambio	pip	Resumen	Ej. de código en versión origen	Ej. de código en versión destino	Dificultad	Ambito	Referencia

Actualizacion
Q-Aer
Primitivas
	0.29 → 0.30	Clase: NoiseModel
		Cambio de compuertas base desde ["id", "u3", "cx"] → ["id", "rz", "sx", "cx"].
		
		Asociado a deprecaciones de u3 en **qiski-terra** y **qiskit-ibmq-provider**.
			No	`Actaualización de compuertas básicas para modelado de ruido cuántico.`	`from qiskit.providers.aer.noise import NoiseModel`
					
					`noise_model = NoiseModel()`
					
					`. . . configure noise . . .`
					
					`result = execute(. . ., basis_gates=noise_model.basis_gates).result()`
						`from qiskit_aer.noise import NoiseModel`
						
						`noise_model = NoiseModel()`
						
						`. . . configure noise . . .`
						
						`simulator = AerSimulator(`
						`    noise_model=noise_model,`
						`    coupling_map=[[0, 1], [1, 0]],`
						`    basis_gates=noise_model.basis_gates`
						`)`
							Baja	QSE	Qiskit 0.30 release notes
									
									IBM Quantum Documentation

Deprecación
Q-Aer
	0.29 → 0.30	Se deprecó:
		
		El pasaje directo del objeto qObj directamente al método run() de backends de simulación Aer.
		
		Instantáneas en qiskit.providers.aer.extensions
		
		Incorporación de error por ruido no local directamente sobre el circuito.
		
		El uso del parámetro **method** en **StatevectorSimulator** y **UnitarySimulator**, para simular una corrida sobre GPU.
			No	`Instrucciones de snapshots en` **`qiskit.providers.aer.extensions`**`, marcadas como obsoletas.`		`backend.run(circuits, **run_options)`
						
						`qiskit.providers.aer.library`
						
						`device="GPU"`
							Baja	QSE	Qiskit 0.30 release notes
									
									IBM Quantum Documentation

Categoría	Flujo de cambio	Resumen del escenario	Ejemplo de código (origen)	Ejemplo de código (destino)	Dificultad	Afectación (SE/QSE)	Referencia
Deprecación de métodos	0.18.x → 0.30.0	Deprecación de `u3` en favor de `rz` y `sx`.	`circuit.u3(theta, phi, lambda, qubit)`	`circuit.rz(phi, qubit); circuit.sx(qubit)`	Baja	QSE (Reemplazo de compuertas base)	Qiskit 0.30 Release Notes
Modificación de parámetros	0.18.x → 0.30.0	Cambio en `basis_gates` por defecto en `NoiseModel`.	`NoiseModel(basis_gates=["id", "u3", "cx"])`	`NoiseModel(basis_gates=["id", "rz", "sx", "cx"])`	Baja	QSE (Configuración de ruido)	Qiskit Aer 0.9.0 Notes
Reestructuración de módulos	0.18.x → 0.30.0	Movimiento de `qiskit.providers.aer.extensions` a `qiskit.providers.aer.library`.	`from qiskit.providers.aer.extensions import Snapshot`	`from qiskit.providers.aer.library import SaveStatevector`	Moderada	QSE (Serialización de estados)	Aer 0.9.0 Changelog

Fig. 2. Examples of manual vs. automatic taxonomy scenarios

In terms of correctness and coherence of scenarios, there is a high correspondence between the manual taxonomy and the output from the commercial models. Regarding the classification of SE vs QSE scenarios, we determined that more complex and information rich prompts allow the models to discern between them with greater precision, providing appropriate descriptions, with a high degree of alignment with the manual taxonomy. When extracted documentation is included into the prompt, the models behave the best, even surpassing the manual taxonomy in paradigmatic examples. We also noticed a slight improvement in the precision, completeness of examples, and explanation when we used English language prompts. See Fig. 2

We observed a a trend in the experiment that goes as follows: **Initial Phase**, where the scenarios obtained by the manual taxonomy represent a superset of those obtained automatically and the agreement between taxonomies is limited. This is mainly due to rudimentary prompts and limited information included. To only information the model has about the particular problem was gained during training. **Intermediate Phase**, where prompts include information of the official documentation. In this phase, the intersection between scenarios increase. The LLMs are able to detect scenarios not present in the manual taxonomy or provide a better version of them with improved categorization, richer examples, while still detecting scenarios described by the manual taxonomy. However, this is not the general case yet and some times the models overlooked some important scenarios.

Additionally, we believe the trend flows to an hypothetical **Future Phase**, as shown in the diagram, see Figure 3. In this phase, the LLM models will be able to retrieve all the scenarios of the manual taxonomy while provide information about other scenarios not present in the manual taxonomy.

From what has been previously stated, we can infer that the use of LLMs is plausible and suitable for automatic taxonomy generation of QSE migration scenarios, substantially reducing the required time. This is more clearly evidenced in versions where the documentation is extensive, includes a greater number of disruptive API changes or substantial improvements associated with QSE (usually in major version updates).

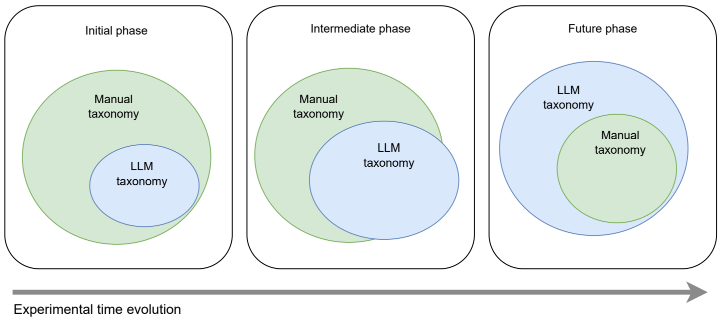


Fig. 3. Flow chart of experimental steps

5 Discussion

The creation of a manual taxonomy of migration scenarios is a complex process of review and analysis that requires considerable effort and time. It often requires cross-referencing different authoritative sources, code review, and interaction with other members of the development team. The quality of the final result is directly related to the level and experience of the professional in SE. The use of LLMs to aid in the elaboration of this kind of taxonomies can save time of the SE expert making the process faster and less prone to errors.

However, careful prompting is needed in order to reduce ambiguities, replication, contradictions, or hallucinations, impacting the accuracy of the responses. Adding more contextual information directly into the prompt increase extensibility and completeness of the responses. In this regard, the need for complementary strategies to address constraints associated with context length of the current models also becomes evident. An analysis of RAG techniques that allows the models to access large amount of documentation is pending, but given the current context length of the models and the size in tokens of the documentation, it may be necessary to implement and asses. See Figure 4.

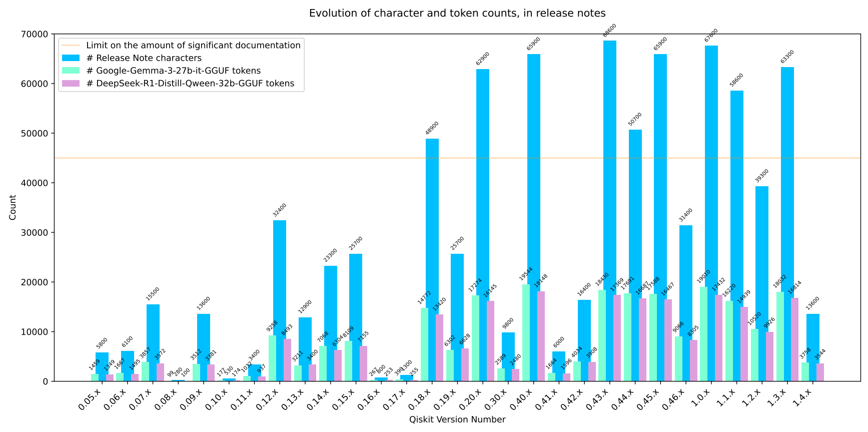


Fig. 4. Qiskit release notes size distribution

Another relevant topic is the knowledge of updates introduced by a specific Qiskit version to quantum components and associated performance. In this regard, the dimensions of our taxonomy "Migration Difficulty" and "Relation to QSE/SE" are intended to serve as the link for such types of assessments.

Having a complete taxonomy will allow us to measure the performance of LLMs, copilots and other automatic tools for code migration grouped by migration scenarios between Qiskit versions, and to prioritize the efforts towards the development of techniques and tools that solve the more difficult cases, or towards the QSE specific scenarios were no other tools are available.

We know LLMs have different performance depending on the programming language [Twist et al., 2025]. Python is one of the languages were most of LLMs excels at coding. Using LLMs to address migration scenarios in Qiskit could present an advantage over other quantum specific programming languages like Q# which has C-like syntax. However, this claims needs validation. Our taxonomy can be the foundation for those experiments.

6 Conclusion and Future Work

This work allowed us to identify the strengths of LLMs to generate a taxonomy of migration scenarios which was validated using an hybrid approach and comparing the output of the models with a manually generated taxonomy. The LLMs ability to detect cases not covered by an SE expert, in addition to greater descriptive precision and completeness in code examples that support each migration scenario, have proven useful enough to aid quantum software programmers to identify the performance gain, future deprecations, and improvement on circuit visualization easily. Given that these models can recognize the migration scenarios automatically, the next step is to asses the LLMs and copilots capabilities to solve this problems, also in an automatic or semi-automatic fashion.

The development of a taxonomy, automated or otherwise, allow us to move our research forward towards automatic tools for quantum software development. Out next steps are:

– Generate the taxonomy for all the Qiskit versions, including the newly released version 2.0.0. This step includes continue refining the prompts and testing models with larger context size, test more advanced techniques like prompt summarizing, Retrieval-Augmented Generation (RAG) [Lewis et al., 2021], Chain of Thought (CoT)
  [Wei et al., 2023] and other reasoning architectures.
– To evaluate the ability of LLMs to detect different migration scenarios in quantum software code and apply the migration automatically. We are interested in asses the LLMs performance regarding detecting and fixing the different migration categories, with special attention to those with high difficulty and those that are QSE specific.
– Survey or develop metrics that allow us to measure the performance of different models to detect and fix the migration scenarios. Ideally, a copilot may scan the code and detect migration scenarios and suggest ways to fix or improve. However, to check code accuracy in quantum software may require formal software verification techniques applied to QSE.

References

[Ali et al., 2021] Ali, S., Arcaini, P., Wang, X., and Yue, T. (2021). Assessing the Effectiveness of Input and Output Coverage Criteria for Testing Quantum Programs. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 13–23. ISSN: 2159-4848.

[Almeida et al., 2024] Almeida, A., Xavier, L., and Valente, M. T. (2024). Automatic Library Migration Using Large Language Models: First Results. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 427–433. arXiv:2408.16151 [cs].

[Amy and Gheorghiu, 2020] Amy, M. and Gheorghiu, V. (2020). staq – A full-stack quantum processing toolkit. *Quantum Sci. Technol.*, 5(3):034016. arXiv:1912.06070 [quant-ph].

[Asif et al., 2025] Asif, H., Basit, A., Innan, N., Kashif, M., Marchisio, A., and Shafique, M. (2025). PennyLang: Pioneering LLM-Based Quantum Code Generation with a Novel PennyLane-Centric Dataset. arXiv:2503.02497 [cs].

[Bengesi et al., 2023] Bengesi, S., El-Sayed, H., Sarker, M. K., Houkpati, Y., Irungu, J., and Oladunni, T. (2023). Advancements in Generative AI: A Comprehensive Review of GANs, GPT, Autoencoders, Diffusion Model, and Transformers. arXiv:2311.10242 [cs].

[Biamonte et al., 2017] Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., and Lloyd, S. (2017). Quantum machine learning. *Nature*, 549(7671):195–202.

[Cordeiro et al., 2024] Cordeiro, J., Noei, S., and Zou, Y. (2024). An Empirical Study on the Code Refactoring Capability of Large Language Models. arXiv:2411.02320 [cs].

[Deng et al., 2024] Deng, Y., Zhang, W., Chen, Z., and Gu, Q. (2024). Rephrase and Respond: Let Large Language Models Ask Better Questions for Themselves. arXiv:2311.04205 [cs].

[Dupuis et al., 2024] Dupuis, N., Buratti, L., Vishwakarma, S., Forrat, A. V., Kremer, D., Faro, I., Puri, R., and Cruz-Benito, J. (2024). Qiskit Code Assistant: Training LLMs for generating Quantum Computing Code. arXiv:2405.19495 [quant-ph].

[Gemini, 2024] Gemini (2024). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv:2403.05530 [cs].

[Green et al., 2013] Green, A. S., Lumsdaine, P. L., Ross, N. J., Selinger, P., and Valiron, B. (2013). Quipper: A Scalable Quantum Programming Language. *SIGPLAN Not.*, 48(6):333–342. arXiv:1304.3390 [cs].

[Guo et al., 2023] Guo, J., Lou, H., Li, R., Fang, W., Liu, J., Long, P., Ying, S., and Ying, M. (2023). isQ: Towards a Practical Software Stack for Quantum Programming. arXiv:2205.03866 [quant-ph].

[Isakov et al., 2021] Isakov, S. V., Kafri, D., Martin, O., Heidweiller, C. V., Mruczkiewicz, W., Harrigan, M. P., Rubin, N. C., Thomson, R., Broughton, M., Kissell, K., Peters, E., Gustafson, E., Li, A. C. Y., Lamm, H., Perdue, G., Ho, A. K., Strain, D., and Boixo, S. (2021). Simulations of Quantum Circuits with Approximate Noise using qsim and Cirq. arXiv:2111.02396 [quant-ph].

[Javadi-Abhari et al., 2024] Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C. J., Lishman, J., Gacon, J., Martiel, S., Nation, P. D., Bishop, L. S., Cross, A. W., Johnson, B. R., and Gambetta, J. M. (2024). Quantum computing with Qiskit. arXiv:2405.08810 [quant-ph].

[Jiménez-Navajas et al., 2025] Jiménez-Navajas, L., Pérez-Castillo, R., and Piattini, M. (2025). Code generation for classical-quantum software systems modeled in UML. *Softw Syst Model*.

[Jin et al., 2025] Jin, H., Chen, H., Lu, Q., and Zhu, L. (2025). Towards Advancing Code Generation with Large Language Models: A Research Roadmap. arXiv:2501.11354 [cs].

[Lewis et al., 2021] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks.

[Li et al., 2020] Li, G., Zhou, L., Yu, N., Ding, Y., Ying, M., and Xie, Y. (2020). Proq: Projection-based Runtime Assertions for Debugging on a Quantum Computer. arXiv:1911.12855 [cs].

[Murillo et al., 2025] Murillo, J. M., Garcia-Alonso, J., Moguel, E., Barzen, J., Leymann, F., Ali, S., Yue, T., Arcaini, P., Castillo, R. P., Guzmán, I. G. R. d., Piattini, M., Ruiz-Cortés, A., Brogi, A., Zhao, J., Miranskyy, A., and Wimmer, M. (2025). Quantum Software Engineering: Roadmap and Challenges Ahead. *ACM Trans. Softw. Eng. Methodol.*, page 3712002. arXiv:2404.06825 [cs].

[Nation et al., 2025] Nation, P. D., Saki, A. A., Brandhofer, S., Bello, L., Garion, S., Treinish, M., and Javadi-Abhari, A. (2025). Benchmarking the performance of quantum computing software. arXiv:2409.08844 [quant-ph].

[OpenAI, 2024] OpenAI (2024). GPT-4 Technical Report. arXiv:2303.08774 [cs].

[Preskill, 2018] Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79. arXiv:1801.00862 [quant-ph].

[Quetschlich et al., 2023] Quetschlich, N., Burgholzer, L., and Wille, R. (2023). MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing. *Quantum*, 7:1062. arXiv:2204.13719 [quant-ph].

[Ramalho et al., 2024] Ramalho, N. C. L., Souza, H. A. d., and Chaim, M. L. (2024). Testing and Debugging Quantum Programs: The Road to 2030. arXiv:2405.09178 [cs].

[Rozière et al., 2024] Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. (2024). Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs].

[Sahoo et al., 2024] Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., and Chadha, A. (2024). A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. arXiv:2402.07927 [cs].

[Sivarajah et al., 2021] Sivarajah, S., Dilkes, S., Cowtan, A., Simmons, W., Edgington, A., and Duncan, R. (2021). t$|$ket$\rangle$: A Retargetable Compiler for NISQ Devices. *Quantum Sci. Technol.*, 6(1):014003. arXiv:2003.10611 [quant-ph].

[Svore et al., 2018] Svore, K. M., Geller, A., Troyer, M., Azariah, J., Granade, C., Heim, B., Kliuchnikov, V., Mykhailova, M., Paz, A., and Roetteler, M. (2018). Q#: Enabling scalable quantum computing and development with a high-level domain-specific language. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, pages 1–10. arXiv:1803.00652 [quant-ph].

[Tsantalis et al., 2022] Tsantalis, N., Ketkar, A., and Dig, D. (2022). RefactoringMiner 2.0. *IEEE Transactions on Software Engineering*, 48(3):930–950. Conference Name: IEEE Transactions on Software Engineering.

[Twist et al., 2025] Twist, L., Zhang, J. M., Harman, M., Syme, D., Noppen, J., and Nauck, D. (2025). Llms love python: A study of llms' bias for programming languages and libraries.

[Wei et al., 2023] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. (2023). Chain-of-thought prompting elicits reasoning in large language models.

[Weisz et al., 2024] Weisz, J. D., He, J., Muller, M., Hoefer, G., Miles, R., and Geyer, W. (2024). Design Principles for Generative AI Applications. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–22. arXiv:2401.14484 [cs].

[Xu et al., 2018] Xu, Z., Ying, M., and Ying, S. (2018). A Logic for Recursive Quantum Programs. arXiv:1812.00349 [cs].

[Zhao, 2021] Zhao, J. (2021). Some Size and Structure Metrics for Quantum Software. arXiv:2103.08815 [cs].

[Zhao, 2023] Zhao, J. (2023). On Refactoring Quantum Programs. arXiv:2306.10517 [cs].