Qiskit Code Migration with LLMs

Abstract

This work presents a hybrid Artificial Intelligence (AI) and Quantum Software Engineering (QSE) approach to address critical challenges in quantum software maintenance and availability. Aligning with core QSE objectives, we develop tools that raise the abstraction level for quantum programming, specifically tackling code migration and API obsolescence in rapidly evolving ecosystems like Qiskit. Our strategy leverages structured knowledge to guide Large Language Models (LLMs), providing a practical and automated solution to ensure quantum software functionality and availability, streamline deployment workflows, and reduce both technical and technological gaps.

ACM Reference Format:

. 2026. Qiskit Code Migration with LLMs. In *Proceedings of 7th International Workshop on Quantum Software Engineering (Q-SE 2026).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

As quantum computing advances [3, 5], driven partly by increasing hardware availability, quantum software ecosystems must evolve rapidly. This progress, however, introduces traditional software challenges like API obsolescence across version updates. These updates can critically affect software functionality, ultimately compromising the availability and reliability of quantum-based products and services. Simultaneously, AI agents are emerging as disruptive forces in software development. This work focuses on creating tools to raise the abstraction level in QSE [6, 8, 10].

Automated code migration and generation represent a relevant and complex field of study that remains relatively unexplored in the QSE context. Our approach integrates AI to improve development workflows in quantum software engineering, specifically targeting the refactoring of Qiskit [1] code. We employ low-code tools for workflow design, focusing our study on the Qiskit ecosystem due to its relevance and active community. Our core strategy involves guiding Large Language Models (LLMs) with automatically generated, structured information [9, 16]. This approach offers a pragmatic alternative to relying solely on a model's internal knowledge or to implementing more complex and resource-intensive strategies like Retrieval-Augmented Generation (RAG) [2, 7]. This methodology originated from the need for a viable solution for small teams, where the considerable complexity of full RAG pipelines is often prohibitive. Consequently, we compare our streamlined method against advanced techniques like RAG, highlighting the strengths, weaknesses, and practical obstacles encountered with each strategy.

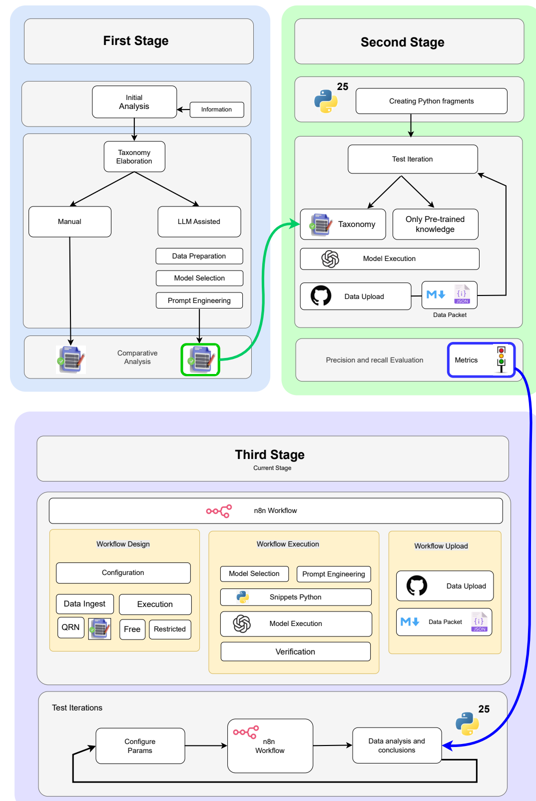


Figure 1: Structure of the Experimental Workflow

This work continues our research line. Initially, we evaluated the feasibility of constructing a taxonomy that summarizes migration scenarios for a specific major Qiskit version [14]. Using a hybrid approach, we demonstrated the benefits of automation over manual effort. Subsequently, we established metrics to evaluate LLMs using this structure, assessing their performance on two key tasks: detecting migration scenarios and generating adaptation suggestions, based on a set of carefully constructed synthetic code fragments [13]. In this current, third stage, we deepen the comparison between our minimalist LLM-guidance approach and more complex strategies like RAG to identify the scenarios where each is most effective. At the same time, the use of automated workflow tools represents a significant advance over previous stages, as it allows us to further isolate the experimental environment, thereby reinforcing the isolation, replicability and extensibility, see Figure 1.

[1]IBM Qiskit - https://www.ibm.com/quantum/qiskit

Q-SE 2026, Rio de Janeiro, Brazil

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Methodology

We designed the experimental workflow using a public, open-source GitHub repository [2]. Its structure includes a directory of synthetically generated Python code fragments for the target Qiskit version. We designed these fragments to have controlled complexity and pre-categorized them according to migration scenarios to streamline the evaluation process. This workflow is structured into the following directories and subdirectories:

- **Data-ingestion**: Contains the markdown files that will compose the model's knowledge base.
- **Data-rag-chatbot**: Input sources for the model.
  - **Prompts**: User and system prompts for each operational mode of the target model.
  - **Scripts**: Execution of auxiliary functionalities.
  - **Snippets**: 26 synthetically generated Python code scenarios, structured by functionality.
- **Answers**: Contains a directory for each test run, which includes 26 processing results and a metadata file for the invocation.

We implemented the core processing pipeline using the n8n tool [3] [4], summarizing it in four stages, see Figure 2:

- Global configuration and data ingestion. [4]
- Creation of an embeddings database.
- Execution of migration tests against the target model.
- Parameterized execution of validations on the refactored code generated by the model.
- Upload of results (data and metadata) to the GitHub repository.

We evaluated the following models: **Gpt-oss-20b** [5] (local infrastructure provided by *Purrfect AI* [6] [4], **Gemini-2.5-Flash** [7], and **GPT4o-mini** [8]. We used the Qiskit Release Notes and our own automated Taxonomy of Migration Scenarios as official information sources. We tasked the models with:

- Detecting migration scenarios
- Recommending refactorings (enabling comparison with our previous work)
- Generating adapted code for the target version.

To assess the impact of external guidance, we tested two operational modalities:

- **Free mode**, where the model could utilize its pre-acquired knowledge.
- **Restricted mode**, where we specifically guided the model to use the embeddings database generated from our official sources.

To ensure experimental rigor, we incorporated several safeguards. Prompts were dynamically generated based on the experimental mode. We used a dedicated **qdrant_id** reference field to trace each model response to its source in the Qdrant embeddings database [9] [12], allowing us to quantify data access patterns. Furthermore, configurable post-processing stages, such as syntactic validation of the generated code, were implemented within the n8n workflow to reinforce output quality. All auxiliary functionalities are configurable and documented in the official repository.

A significant sub-experiment evaluated whether model performance improved when the database contained only the **Qiskit Release Notes** (QRN) versus when it was augmented with our **Automatically Generated Taxonomy**. This required considerable effort, as we established specific metrics to count detected scenarios and evaluate the quality of the responses and suggested refactorings. Our analysis confirmed that the taxonomy effectively complements the official release notes, leading to more complete responses and better-adapted code. This finding, which is both empirically demonstrable and logically sound—as the taxonomy serves as a condensed informational layer over the release notes—reinforces the taxonomy's relevance, as indicated in our previous work [14]. Consequently, all subsequent model testing utilized both information sources.

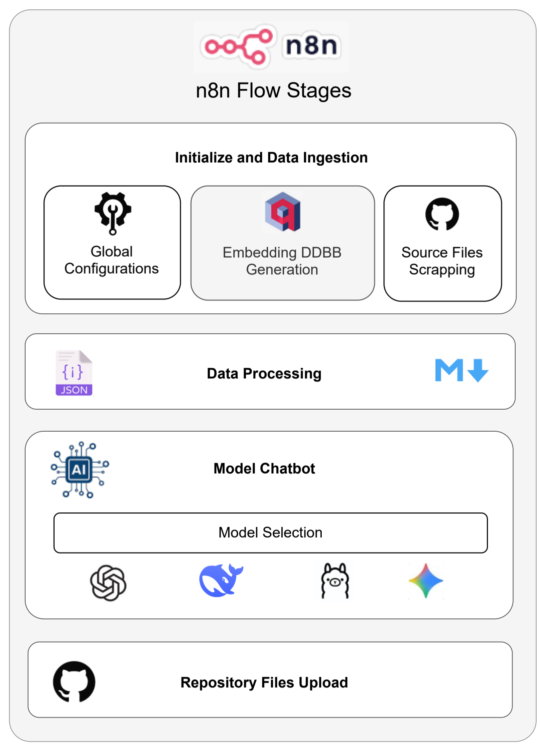


Figure 2: Stages in the n8n experimental workflow

Besides addressing a complex and timely research topic in QSE, this work consistently maintains methodological rigor and objectivity. We emphasize replicability and extensibility, which are essential requirements when applying AI tools in research.

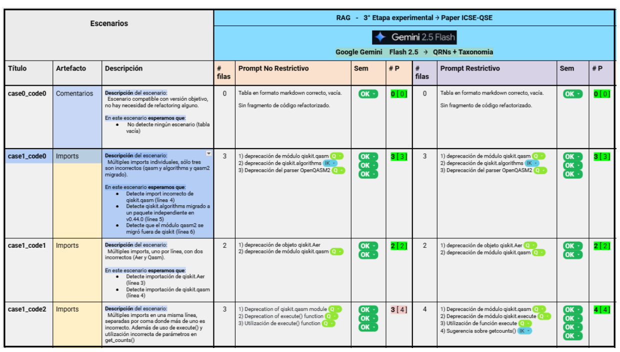
[2] GitHub Repository - https://github.com/jose-manuel-suarez/qiskit_rag
[3] n8n tool - https://n8n.io/
[4] All parameterizations associated with the experimental workflow are documented in the official repository: https://github.com/jose-manuel-suarez/qiskit_rag
[5] Model OpenAI Gpt-oss-20b - https://openai.com/es-ES/index/introducing-gpt-oss/
[6] Purrfect AI - https://purrfectai.com.ar/
[7] Model Google Gemini-2.5-Flash - https://www.skills.google/focuses/115004?locale=es&parent=catalog
[8] Model OpenAI GPT4o-mini - https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/
[9] Qdrant Database - https://qdrant.tech/rag/

3 Results

Although the final evaluation of the RAG architecture enhanced with our taxonomy is still ongoing, preliminary findings strongly support our initial hypothesis from prior work. The current analysis involves testing new models -**Google GeminiFlash-2.5 - OpenAI GPT-4o-mini - GPT-OSS-20b**— across 26 distinct scenarios under two operational modes. This has resulted in a manual review of over 150 individual analysis cases by a panel of experts evaluating the metrics in isolation to ensure cross-consistency in the process.

Our core hypothesis states that the quality of migration scenario detection and refactoring suggestions improves when official information sources are expanded, properly structured, and condensed. Initial data substantiates this claim, revealing a notable increase in correctly identified migration scenarios—from 38% to 63%. We also observed an improvement in the accuracy of generated refactorings, which currently demonstrate approximately 72% precision and 77% recall.

We expect that the RAG approach, empowered by our automatically generated taxonomy, will achieve significant improvements in both the precision and authoritativeness of the entire code migration process upon completion of the extensive manual evaluation, see Figure 3.



Escenarios			RAG - 3° Etapa experimental → Paper ICSE-QSE Gemini 2.5 Flash Google Gemini Flash 2.5 → QRNs + Taxonomia							
Título	Artefacto	Descripción	# filas	Prompt No Restrictivo	Sem	# P	# filas	Prompt Restrictivo	Sem	# P
case0_code0	Comentarios	Descripción del escenario: Escenario compatible con versión objetivo, no hay necesidad de refactoring alguno. En este escenario **esperamos que**: • No detecte ningún escenario (tabla vacía)	0	Tabla en formato markdown correcto, vacía. Sin fragmento de código refactorizado.	OK	0 [0]	0	Tabla en formato markdown correcto, vacía. Sin fragmento de código refactorizado.	OK	0 [0]
case1_code0	Imports	Descripción del escenario: Múltiples imports individuales, sólo tres son incorrectos (qasm y algorithms y qasm2 migrado). En este escenario **esperamos que**: • Detecte import incorrecto de qiskit.qasm (línea 4) • Detecte qiskit.algorithms migrado a un paquete independiente en v0.44.0 (línea 5) • Detecte que el módulo qasm2 se migró fuera de qiskit (línea 6)	3	1) deprecación de módulo qiskit.qasm Q 2) deprecación de qiskit.algorithms IK 3) Deprecación del parser OpenQASM2 Q	OK OK OK	3 [3]	3	1) deprecación de módulo qiskit.qasm Q 2) deprecación de qiskit.algorithms IK 3) Deprecación del parser OpenQASM2 Q	OK OK OK	3 [3]
case1_code1	Imports	Descripción del escenario: Múltiples imports, uno por línea, con dos incorrectos (Aer y Qasm). En este escenario **esperamos que**: • Detecte importación de qiskit.Aer (línea 3) • Detecte importación de qiskit.qasm (línea 4)	2	1) deprecación de objeto qiskit.Aer 2) deprecación de módulo qiskit.qasm Q	OK OK	2 [2]	2	1) deprecación de objeto qiskit.Aer Q 2) deprecación de módulo qiskit.qasm Q	OK OK	2 [2]
case1_code2	Imports	Descripción del escenario: Múltiples imports en una misma línea, separadas por coma donde más de uno es incorrecto. Además de uso de execute() y utilización incorrecta de parámetros en get_counts()	3	1) Deprecation of qiskit.qasm module Q 2) Deprecation of execute() function Q 3) Utilización de execute() function Q	OK OK OK	3 [4]	4	1) Deprecación de módulo qiskit.qasm Q 2) Deprecación de módulo qiskit.execute Q 3) Utilización de función execute Q 4) Sugerencia sobre getcounts() IK	OK OK OK OK	4 [4]

Figure 3: Analysis Snapshot

4 Discussion and Conclusion

4.1 Discussion

Our study demonstrates that enhancing RAG architectures with an automatically generated taxonomy significantly improves quantum API migration tasks—not only in detecting relevant scenarios and generating high-quality refactoring suggestions but also in producing accurate, refactored code automatically. This approach provides a structured knowledge framework that guides LLMs more effectively than relying on unstructured documentation alone.

4.1.1 Key Insights and Challenges. A central insight from our work is the potential **transitivity of taxonomies**. We hypothesize that if a taxonomy for a single version jump is effective, maintaining a comprehensive set for all major versions would be highly advantageous. This would allow the model to dynamically incorporate multiple taxonomies, providing informed recommendations even for complex, multi-version migrations not explicitly covered by a single source.

However, we encountered several significant challenges. The foremost is the **scarcity of high-quality test data**. The limited availability of real-world GitHub projects for specific Qiskit versions makes large-scale validation difficult. While we have begun to identify 56 realistic code examples from pull requests, our current reliance on synthetically generated fragments, though necessary, limits experimental scalability. This data scarcity is compounded by the **high expertise required for manual verification**, a notoriously time-consuming process that often relies on incomplete official guides.

4.1.2 Technical Limitations and Design Choices. Our implementation also faced technical constraints. The choice of **Semantic text splitter** proved critical; we found the **Recursive Text Splitter** [10] more reliable than the Semantic Character Splitter [11] for preserving relevant context. Furthermore, the **tooling requirements of automated workflows** like n8n restricted our model selection, excluding candidates like DeepSeek that lacked the necessary agent support.

To ensure rigorous evaluation, we made key design decisions. We enforced **stateless model invocations** to guarantee atomicity and isolation between test cases, prioritizing reproducibility and comparability over potential benefits from short-term memory. We also implemented a traceability system using **reference keys** and **Qdrant IDs** to meticulously track the provenance of every model response.

4.2 Conclusion and Future Work

This work underscores the viability of using AI-guided tools to raise the abstraction level in Quantum Software Engineering. By providing structured, domain-specific knowledge to LLMs, we can mitigate the challenges of API obsolescence, ultimately helping to decouple development teams from the volatility of underlying quantum SDKs.

Future work will focus on several concrete directions:

- **Scaling data ingestion:** Augmenting the RAG's knowledge base with additional authoritative sources, such as Qiskit Change Logs[12] and official Migration Guides[13], and extracting realistic code from GitHub pull requests at scale.
- **Extending ecosystem coverage:** While Qiskit's prevalence makes it a primary candidate for this study, extending our methodology to non-Python quantum frameworks (e.g., Q# and OpenQASM) is crucial to evaluate LLM performance in more diverse linguistic environments and establish the generality of our approach. [1, 5, 11, 15]
- **Implementing robust validation:** Developing automated functional testing and syntax validation pipelines to improve experimental scalability and reliability. A critical test will involve evaluating our approach on a Qiskit version

[10] n8n - Recursive Character Text Splitter https://docs.n8n.io/integrations/builtin/cluster-nodes/sub-nodes/n8n-nodes-langchain.textsplitterrecursivecharactertextsplitter/

[11] n8n - Character Text Splitter https://docs.n8n.io/integrations/builtin/cluster-nodes/sub-nodes/n8n-nodes-langchain.textsplittercharactertextsplitter/

[12] https://github.com/qiskit/qiskit/releases

[13] https://quantum.cloud.ibm.com/docs/en/migration-guides

released *after* the model's knowledge cutoff, completely isolating the taxonomy's effect from the model's pre-trained knowledge.

The challenges of data scarcity and validation complexity highlight the pressing need for the very automated tools this research line aims to create. The promising results achieved with our taxonomy-enhanced RAG architecture establish a strong foundation for building more resilient and automated quantum software migration workflows.

References

[1] Dupuis, N., Buratti, L., Vishwakarma, S., Forrat, A. V., Kremer, D., Faro, I., Puri, R., and Cruz-Benito, J. Qiskit code assistant: Training LLMs for generating quantum computing code.

[2] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H. Retrieval-augmented generation for large language models: A survey.

[3] Gill, S. S., Cetinkaya, O., Marrone, S., Claudino, D., Haunschild, D., Schlote, L., Wu, H., Ottaviani, C., Liu, X., Machupalli, S. P., Kaur, K., Arora, P., Liu, J., Farouk, A., Song, H. H., Uhlig, S., and Ramamohanarao, K. Quantum computing: Vision and challenges. pp. 19–42.

[4] Hirzel, M. Low-code programming models. 76–85.

[5] 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. Quantum computing with qiskit.

[6] Khan, A. A., Taibi, D., and Akbar, M. A. Advancing quantum software engineering: A vision of hybrid full-stack iterative model.

[7] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks.

[8] Mandal, A. K., Nadim, M., Roy, C. K., Roy, B., and Schneider, K. A. Quantum software engineering and potential of quantum computing in software engineering research: A review.

[9] Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., and Gao, J. Large language models: A survey.

[10] 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. Quantum software engineering: Roadmap and challenges ahead. 3712002.

[11] Pathak, P., Tarakeshwar, K., Ali, S. S., Devendrababu, S., and Ganesan, A. The evolution of IBM's quantum information software kit (qiskit): A review of its applications.

[12] Plale, B., Jyesta, S. N., and Withana, S. Vector embedding of multi-modal texts: a tool for discovery?

[13] Suárez, J. M., Bibbó, L. M., Bogado, J., and Fernandez, A. Automatic qiskit code refactoring using large language models.

[14] Suárez, J. M., Bibbó, L. M., Bogado, J., and Fernandez, A. Taxonomy of migration scenarios for qiskit refactoring using LLMs.

[15] Vishwakarma, S., Harkins, F., Golecha, S., Bajpe, V. S., Dupuis, N., Buratti, L., Kremer, D., Faro, I., Puri, R., and Cruz-Benito, J. Qiskit HumanEval: An evaluation benchmark for quantum code generative models.

[16] Yuan, Z., Chen, W., Wang, H., Yu, K., Peng, X., and Lou, Y. TRANSAGENT: An LLM-based multi-agent system for code translation.