QuanBench Plus: A Unified Multi-Framework Benchmark for LLM-Based Quantum Code Generation

Ali Slim[1*] **Haydar Hamieh**[1*] **Jawad Kotaich**[1*] **Yehya Ghosn**[1*]

Mahdi Chehimi[1] **Ammar Mohanna**[1] **Hasan Abed El Kader Hammoud**[2] **Bernard Ghanem**[2]

[1]American University of Beirut
[2]King Abdullah University of Science and Technology (KAUST)

Abstract

Large language models (LLMs) are increasingly used for code generation and task automation. However, their effectiveness in quantum code generation across multiple major frameworks remains underexplored. This work introduces *QuanBench Plus*, a unified multi-framework benchmark spanning Qiskit, Pennylane, and Cirq. Specifically, 42 tasks are adapted across three foundational categories (quantum algorithms, gate decomposition, and state preparation) and framework-aligned canonical solutions are provided for automated grading. Following the functional-evaluation paradigm popularized by functional code-generation benchmarks such as HumanEval, correctness assessment is standardized using Pass@k-based functional evaluation and KL-divergence-based acceptance is added for probabilistic outputs. The Pass@1 results are reported using greedy decoding and Pass@5 results using $k = 5$ samples per task. Pass@1 after feedback (FB) is additionally reported when feedback to the model is triggered by an incorrect answer or a compilation error. Fidelity is excluded from primary scoring because circuit similarity may not reflect prompt-specific functional correctness. The best-performing models achieve Pass@1 results up to 42.9% in Pennylane, 54.8% in Cirq, and 59.5% in Qiskit, illustrating both progress and remaining gaps in using LLMs for reliable quantum code generation.

1 Introduction

LLMs perform well on classical code benchmarks like HumanEval Chen et al. (2021), but as quantum computing moves toward practice, developers increasingly rely on frameworks such as Qiskit Aleksandrowicz et al. (2019), Pennylane Bergholm et al. (2018), and Cirq of Cirq (2021). A key question is whether modern LLMs can generate *correct* quantum programs. Unlike classical programs, quantum outputs are *probabilistic* measurement statistics of quantum states, where a single-qubit state is given by $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, with probabilities $|\alpha|^2$ and $|\beta|^2$ to measure outputs "0" and "1", respectively Nielsen & Chuang (2010). Thus, correctness must account for output distributions, measurement schemes, and execution settings. Existing benchmarks, Qiskit HumanEval Vishwakarma et al. (2024), QHackBench Basit et al. (2025), QCircuitBench Wang et al. (2024), and QuanBench Guo et al. (2025), are mostly single-framework, making it hard to separate quantum reasoning from framework familiarity. A multi-framework benchmark can reveal two distinct failure modes: (i) *conceptual* errors in quantum reasoning (e.g., algorithmic or measurement mistakes) and (ii) *framework* errors (e.g., API misuse or simulator issues). We introduce *QuanBench Plus*, a multi-framework benchmark that keeps task intent constant while varying the framework. This design enables evaluation of whether model performance reflects true quantum reasoning or mere framework familiarity.

Research questions (RQs). We organize the paper around the following questions:

[*]Equal contribution.

- **RQ1:** How accurately can modern LLMs generate *correct* quantum code across Qiskit, Pennylane, and Cirq?
- **RQ2:** To what extent are observed gains driven by framework-specific boilerplate (prefill) versus true task-level reasoning?
- **RQ3:** How much can an automated feedback loop improve one-shot performance under the same functional test harness?

Answers in brief (A1–A3). Our experiments answer these questions as follows:

- **A1:** Current models show meaningful progress but remain far from fully reliable; performance varies substantially across frameworks.
- **A2:** Prefill sometimes improves correctness by reducing interface and boilerplate errors, but its benefit is not consistent.
- **A3:** Feedback-based repair improves Pass@1 further, indicating that many errors are recoverable under iterative correction.

Contributions We introduce *QuanBench Plus*, a unified multi-framework benchmark that spans Qiskit, Pennylane, and Cirq by adapting 42 tasks into framework-aligned prompts that preserve identical functional goals, and provide canonical solutions for automated grading. We standardize evaluation with functional Pass@k and KL-divergence-based acceptance for probabilistic outputs, and report Pass@1, Pass@5, and Pass@1 (FB). We analyze failure modes across frameworks and prompting conditions (prefill/no-prefill, with/without feedback).

2 QuanBench Plus Benchmark

Benchmarking Workflow We follow a standard benchmarking workflow: define the objective, select metrics aligned with task outputs, control the execution environment, construct a dataset with paired prompts and canonical solutions, select representative models and frameworks, and assess outputs under an automated harness.

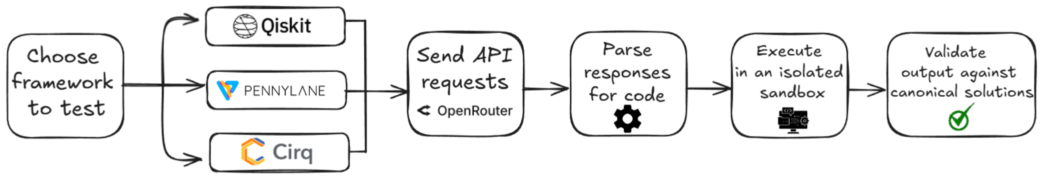

Figure 1: Benchmarking workflow.

Task Set and Categories QuanBench Plus is derived from the original QuanBench task set Guo et al. (2025). We retain tasks that admit clear numerical or functional correctness criteria and adapt them to Qiskit, Pennylane, and Cirq while preserving their objectives. Prompts were modified to account for differences in framework APIs and libraries. Two tasks from the original benchmark were removed due to ambiguities in multi-framework grading (Appendix A). The final benchmark consists of 42 tasks spanning three categories: **Quantum Algorithms**, **Gate Decomposition**, and **State preparation**. Their distribution is presented in Appendix B.

Why We Exclude Fidelity QuanBench Guo et al. (2025) reports *process fidelity* (unitary overlap) (Appendix F) between a reference and generated circuit. In *QuanBench Plus*, correctness is defined as *task success*: a solution is correct if it yields the required measurement statistics under the prompt-specified inputs and measurement scheme. Many circuits are therefore task-equivalent yet have low unitary overlap (e.g., basis-dependent phase changes), and compilation/optimization can produce globally different but functionally equivalent circuits, motivating circuit equivalence checking Burgholzer & Wille (2021); Yamashita & Markov (2010). Fidelity can thus create *false negatives* by penalizing prompt-correct solutions that deviate from a canonical reference. Moreover, as an average-case metric, fidelity may not track task-relevant error, particularly under coherent noise Kueng et al. (2016); Wallman (2015). We therefore prioritize executable functional evaluation (Pass@k) and distributional comparison (KL divergence) rather than fidelity.

Prompt and Output Standardization To ensure fair comparisons, the set of canonical solutions is unified for all models across all frameworks. Each model receives the same modified prompt to ensure that the correct libraries were imported per task and framework, with strict instructions on code-only output and expected function interfaces. For tasks requiring inputs, a random set of non-trivial inputs was generated once and used across all models and frameworks. Each canonical solution's output is standardized to a probability array representing the measurement distribution over computational basis states.

3 Experimental Setup

Models We evaluate a diverse set of LLMs (listed in Appendix C), covering both models studied in QuanBench and more recent releases. All requests are issued through a unified API router. For Pass@1, we use greedy decoding (temperature 0.0) and sample one completion per task. For Pass@5, we sample $k = 5$ completions per task at temperature 0.8. For FB, we use a feedback-loop setting with at most five feedback attempts and report Pass@1 at temperature 0.0 for the first attempt and 0.8 for the rest.

Execution Environment All generated solutions are executed in a controlled Python environment. To facilitate comparison with prior results, we use Python 3.10, Qiskit v0.46.0, and additionally evaluate Cirq v1.6.1 and Pennylane v0.43.1.

Metrics We adopt the HumanEval functional-correctness paradigm Chen et al. (2021). **Pass@k** is the probability that at least one of k sampled completions solves the task (i.e., passes the functional tests); we estimate it by sampling k completions per task and checking whether any is accepted. For probabilistic tasks, we compare the generated measurement distribution $\hat{P}$ to the canonical distribution P using **KL divergence**, which measures how different two probability distributions are (lower is better). We apply additive ε-smoothing and renormalize, and accept if $\mathrm{D}_{\mathrm{KL}}(P\|\hat{P}) < 0.05$; the exact metrics computation is given in Appendix D, and the calibration/proof for the 0.05 threshold is in Appendix E.

Execution and Grading Pipeline For each model completion, we apply the same evaluation procedure:

P1: Parse the completion and extract executable code.

P2: Execute the code in the target framework environment.

P3: Compare outputs using deterministic checks or a distributional threshold.

Feedback Loop In addition to standard one-shot generation, we evaluate a feedback loop setting that allows a model to repair its answer. The feedback loop is triggered on both runtime exceptions and wrong answer outputs. For each task, we execute the initial completion under the same harness used for Pass@k. If execution raises an exception (e.g., syntax/import/runtime errors), we provide the model with the exception trace and the original prompt, and request a corrected code-only solution. If the output of the generated code does not match the canonical solution, we provide the model with the wrong function and the original prompt, and request a corrected code-only solution. We report Pass@1 under this feedback loop as Pass@1 (FB). In all cases, we provide the models with a maximum of 5 repair chances.

4 Results

This section answers **RQ1–RQ3** via controlled functional experiments across frameworks and prompting conditions. We report Pass@1 results across Qiskit, Cirq, and Pennylane.

RQ1: Multi-Framework Functional Correctness Table 2 reports Pass@1 accuracy by framework. The best-performing models achieved Pass@1 scores of **59.5%** in Qiskit, **54.8%** in Cirq, and **42.9%** in Pennylane. These results indicate measurable progress, but also show that functional correctness remains far from reliable across frameworks (Appendix G & H).

RQ2: Prefill vs No-Prefill We evaluate two prompting conditions (prefill/no-prefill) to isolate errors caused by missing boilerplate and framework API mismatches. Detailed results and figures are provided in the Supplementary Material (Appendix I).

Table 1: **RQ2: Effect of prefill on Pass@1.** Δ denotes the change in Pass@1 (prefill $-$ no-prefill) in percentage points. Positive values indicate that prefill improves performance.

Model	**Qiskit Δ**	**Cirq Δ**	**Pennylane Δ**	**Avg. Δ**
`Claude-3.7-Sonnet`	+11.9	-4.8	0.0	**+2.4**
`DeepSeek-Chat`	-2.4	+4.8	-2.4	0.0
`DeepSeek-R1`	-7.2	-15.5	+2.2	-6.8
`Gemini-2.5-Flash`	-7.1	-2.4	-4.7	-4.7
`Gemini-3-Pro-Preview`	-0.2	-11.3	-3.1	-4.9
`Llama-4-Maverick`	+4.8	+4.8	-2.4	**+2.4**
`GPT-4.1`	+7.1	0.0	-2.4	+1.6
`GPT-5.1`	+2.4	0.0	-1.0	+0.5
`Qwen-2.5-7B-Instruct`	+2.4	-7.1	+9.5	+1.6

Prefill yields small, model-dependent average gains (best: Claude-3.7-Sonnet and Llama-4-Maverick at +2.4 pp), while several models degrade on average (worst: DeepSeek-R1 at -6.8 pp). Effects are strongly framework-dependent (e.g., Claude-3.7-Sonnet: +11.9 pp on Qiskit but -4.8 pp on Cirq; Qwen-2.5-7B-Instruct: +9.5 pp on Pennylane but -7.1 pp on Cirq).

RQ3: Feedback-Based Repair We also evaluate a feedback loop that allows up to 5 repair attempts after runtime errors or incorrect outputs. Detailed Pass@1 (FB) tables, heatmaps and errors distribution improvement are provided (Appendix K). Under the feedback-loop setting, the top scores increase to **83.3%** in Qiskit, **76.2%** in Cirq, and **66.7%** in Pennylane.

Table 2: Results of Pass@1 with and without feedback across all frameworks.

	Qiskit		**Cirq**		**Pennylane**	
Model	**Pass@1**	**FB**	**Pass@1**	**FB**	**Pass@1**	**FB**
`Gemini-3-Pro`	**59.5**	73.8	**54.8**	**76.2**	40.5	**66.7**
`GPT-5.1`	57.1	**83.3**	52.4	73.8	**42.9**	**66.7**
`DeepSeek-R1`	50.0	71.4	45.2	61.9	33.3	52.4
`MoonshotAI-Kimi-K2-Thinking`	47.6	69.0	38.1	57.1	26.2	38.1
`Claude-3.7-Sonnet`	45.2	57.1	35.7	59.5	26.2	47.6
`GPT-4.1`	50.0	57.1	33.3	57.1	23.8	45.2
`DeepSeek-Chat`	45.2	42.9	28.6	40.5	31.0	45.2
`Z-ai-GLM-4.7`	42.9	69.0	38.1	61.9	23.8	64.3
`Gemini-2.5-Flash`	40.5	61.9	35.7	50.0	23.8	40.5
`Llama-4-Maverick`	38.1	54.8	28.6	42.9	19.0	38.1
`MiniMax-M2.1`	28.6	57.1	23.8	47.6	31.0	47.6
`Qwen-2.5-7B-Instruct`	16.7	19.0	4.8	7.1	11.9	19.0

5 Discussion

Our multi-framework evaluation shows two consistent trends. First, newer models and improved prompt design yield measurable gains, yet overall performance remains modest. Second, performance varies markedly across Qiskit, Cirq, and Pennylane, suggesting that framework-specific API familiarity drives success more than abstract quantum reasoning. Common failure modes also point to **(i)** limited quantum-programming coverage in training corpora relative to classical code and **(ii)** outdated framework usage patterns in generated code (Appendix J).

Threats to Validity Our evaluation depends on the correctness and completeness of canonical solutions. Multi-framework adaptations can introduce subtle mismatches between prompts and reference implementations. We mitigate this risk by excluding tasks with ambiguous grading and reviewing framework-specific canonical code for functional equivalence. Our results show that quantum-algorithm tasks exhibit the lowest Pass@1 scores across all models, indicating that multi-step algorithmic reasoning remains a significant challenge for current LLMs. A potential threat to validity arises from unequal task distributions across categories. Framework versioning may also influence absolute performance. Models trained on older APIs may fail execution despite a correct high-level intent.

Limitations & Future Work QuanBench Plus contains 42 tasks and may not fully capture the long-tail of real-world quantum development. In addition, we report Pass@1, Pass@5, and Pass@1 (FB) in this version, which may be insufficient to fully characterize model behavior. Our benchmark currently focuses on Qiskit, Pennylane, and Cirq; extending to additional frameworks is left for future work.

6 Conclusion

We answer **RQ1–RQ3** by introducing *QuanBench Plus*, a unified multi-framework benchmark for evaluating LLMs on quantum code generation in Qiskit, Pennylane, and Cirq. By adapting a shared task set and standardizing functional correctness evaluation using Pass@k with KL-divergence-based acceptance for probabilistic tasks, we provide a clearer assessment of LLM capabilities across the broader quantum software ecosystem. Our results show improvement for newer models, yet performance remains uneven across frameworks. These findings indicate that further progress will likely depend not only on larger model scale, but also on richer exposure to quantum software data, stronger reasoning training for compositional circuit generation and error correction, and closer alignment with framework-specific APIs, abstractions, and implementation patterns. We hope QuanBench Plus provides a practical, reproducible basis for future work on multi-framework evaluation, feedback-based repair, and robust testing for probabilistic quantum programs. All code, datasets, and scripts used in this benchmark are publicly available in our GitHub repository[*].

7 Acknowledgments

This work was supported in part by the Vertical Integrated Project at the American University of Beirut, and the KAUST Center of Excellence for Generative AI under award number 5940.

References

Jack Achiam, Shaked Adler, Sandhini Agarwal, Lena Ahmad, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. URL https://arxiv.org/abs/2303.08774.

Meta AI. Llama 4 maverick: Model card. https://ai.meta.com/llama/, 2025. [Accessed: 2025-07-12].

Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, et al. Qiskit: An open-source framework for quantum computing. *Zenodo*, 2019. doi: 10.5281/zenodo.2562111. URL https://qiskit.org.

Anthropic. Claude 3.7 sonnet: Model release. https://www.anthropic.com/news/claude-3-7-sonnet, 2025. [Accessed: 2025-07-12].

Zhihong Bai, Weizhe Yang, Yuzhe Chen, Chenxi Qian, Shuobo Li, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Abdul Basit, Minghao Shao, Muhammad Haider Asif, et al. Qhackbench: Benchmarking large language models for quantum code generation using pennylane hackathon challenges. *arXiv preprint arXiv:2506.20008*, 2025. URL https://arxiv.org/abs/2506.20008.

Ville Bergholm, Josh Izaac, Maria Schuld, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018. URL https://arxiv.org/abs/1811.04968.

[*]Source code: https://github.com/JawadKotaichh/quanbench-plus

Lukas Burgholzer and Robert Wille. Advanced equivalence checking for quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(9):1810–1824, 2021. doi: 10.1109/TCAD.2020.3032630.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Google DeepMind. Gemini 3 (model family overview). https://deepmind.google/models/gemini/, 2025. [Accessed: 2026-01-15].

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. URL https://arxiv.org/abs/2412.19437.

Google. Gemini 2.5 flash (model documentation). https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash, 2025. [Accessed: 2026-01-15].

Xiaoyu Guo, Minggu Wang, and Jianjun Zhao. Quanbench: Benchmarking quantum code generation with large language models. *arXiv preprint arXiv:2510.16779*, 2025. URL https://arxiv.org/abs/2510.16779.

Richard Kueng, David M. Long, Andrew C. Doherty, and Steven T. Flammia. Comparing experiments to the fault-tolerance threshold. *Physical Review Letters*, 117:170502, 2016. doi: 10.1103/PhysRevLett.117.170502.

MiniMax. Minimax-m2.1 large language model. https://platform.minimax.chat/docs, 2024.

Moonshot AI. Kimi-k2 (thinking) large language model. https://platform.moonshot.cn/docs, 2024.

Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 10th anniversary edition edition, 2010.

Developers of Cirq. Cirq: A python framework for nisq algorithms. *Zenodo*, 2021. doi: 10.5281/zenodo.5182845. URL https://cirq.io.

OpenAI. Gpt-5: Model overview. https://platform.openai.com/docs/models, 2025. [Accessed: 2025-07-12].

Sanjay Vishwakarma, Francis Harkins, Siddharth Golecha, et al. Qiskit humaneval: An evaluation benchmark for quantum code generative models. *arXiv preprint arXiv:2406.14712*, 2024. URL https://arxiv.org/abs/2406.14712.

Joel J. Wallman. Bounding experimental quantum error rates relative to fault-tolerant thresholds, 2015. URL https://arxiv.org/abs/1511.00727.

Rui Wang, Xin Zhang, et al. Qcircuitbench: A large-scale benchmark for evaluating quantum circuit generation. *arXiv preprint arXiv:2410.07961*, 2024. URL https://arxiv.org/abs/2410.07961.

Shigeru Yamashita and Igor L. Markov. Fast equivalence-checking for quantum circuits. *Quantum Information and Computation*, 9(9–10):721–734, 2010. doi: 10.48550/arXiv.0909.4119.

Zhipu AI. Glm-4: Open large language models. *arXiv preprint arXiv:2404.03880*, 2024. URL https://arxiv.org/abs/2404.03880.

A Ambiguious Prompts

Prompt Modifications. All prompts were modified to ensure that the correct libraries were imported for each framework. In addition, we enforced that models return code only, without any accompanying explanation, to improve execution efficiency. This requirement was explicitly stated at the beginning of each prompt.

Table 3: **Summary of prompt modifications and task removals.** Lists removed tasks and key prompt edits made to enable consistent cross-framework grading.

Task	Edit	Rationale
5	Task removed	Did not output a clear probabilistic answer, making quantitative evaluation unreliable.
25	Prompt modified	The old version measures all qubits, while the new version measures the first 3 qubits.
28	Prompt modified	The prompt did not clearly instruct measuring all qubits, leading to ambiguous output distributions.
38	Task removed	The testing procedure for the machine learning task was not clearly specified, preventing consistent evaluation.
41	Prompt modified	The randomized input library was replaced by a pre-decided randomly generated input.

B Task Categories and Examples

Table 4: **Task categories.** Number of tasks in each category (total: 42).

Category	**Number of Tasks**
Quantum Algorithms	31
State Preparation	6
Decomposition	5
Total	**42**

QuanBench Plus organizes tasks equivalently across frameworks:

- **Quantum Algorithms**: implement known algorithms or subroutines.
- **Gate Decomposition**: convert high-level operations into native gates.
- **State Preparation**: construct circuits to produce target quantum states.

C Models Evaluated

Table 5: **Models evaluated.** List of models included in our experiments and the corresponding reference for each release.

Model	**Reference**
`Claude-3.7-Sonnet`	Anthropic (2025)
`DeepSeek-Chat`	DeepSeek-AI (2024)
`DeepSeek-R1`	DeepSeek-AI (2024)
`Gemini-2.5-Flash`	Google (2025)
`Gemini-3-Pro`	DeepMind (2025)
`GPT-4.1`	Achiam et al. (2023)
`GPT-5.1`	OpenAI (2025)
`Llama-4-Maverick`	AI (2025)
`Qwen-2.5-7B-Instruct`	Bai et al. (2024)
`MiniMax-M2.1`	MiniMax (2024)
`Z-ai-GLM-4.7`	Zhipu AI (2024)
`MoonshotAI-Kimi-K2-Thinking`	Moonshot AI (2024)

D Pass@k and KL divergence

Pass@k. We use Pass@k as our primary correctness metric. Pass@k measures the probability that at least one of the top-k generated solutions is correct:

$$\text{Pass@}k = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}, \tag{1}$$

where n is the number of generated samples and c is the number of correct samples. We report Pass@1 and Pass@5 in this version.

KL divergence for probabilistic outputs. We compute the KL divergence between the canonical distribution P and the model-generated distribution Q:

$$D_{\text{KL}}(P\|Q) = \sum_x P(x)\,\log\frac{P(x)}{Q(x)}. \tag{2}$$

To avoid undefined values when $Q(x) = 0$ for states with $P(x) > 0$, we apply a small additive smoothing constant ε to both distributions before renormalization to avoid infinite KL values from zero-probability bins. With $\varepsilon \ll 1/\text{shots}$ (the finite-shot probability resolution), smoothing is negligible on nonzero entries and mainly prevents zero bins.. A solution is accepted when the resulting divergence is below a global threshold set to 0.05.

E Calibration of the KL Acceptance Threshold

Some benchmark tasks are probabilistic: correctness is defined by matching a target measurement distribution. Even the canonical circuit exhibits finite-shot variability, so we calibrate an acceptance threshold from repeated canonical executions.

For each task t, we run the canonical reference circuit $R = 1000$ times to obtain empirical distributions $\{P_i^{(t)}\}_{i=1}^{R}$, and define the reference distribution as their mean (renormalized):

$$P_{\text{ref}}^{(t)} = \text{Normalize}\left(\frac{1}{R}\sum_{i=1}^{R} P_i^{(t)}\right). \tag{3}$$

We then define a task-specific null distribution of within-canonical variability:

$$d_i^{(t)} = D_{\text{KL}}\left(\widetilde{P}_{\text{ref}}^{(t)} \,\middle\|\, \widetilde{P}_i^{(t)}\right). \tag{4}$$

We select the acceptance threshold as a high quantile of the pooled null KL values across tasks:

$$\tau_{\text{global}}(q) = \text{Quantile}_q\left(\bigcup_t \{d_i^{(t)}\}_{i=1}^R\right). \tag{5}$$

With $q = 0.997$, $\tau_{\text{global}} = 0.048$, so we use $\tau = 0.05$ as a slightly more permissive constant to allow minor additional variability.

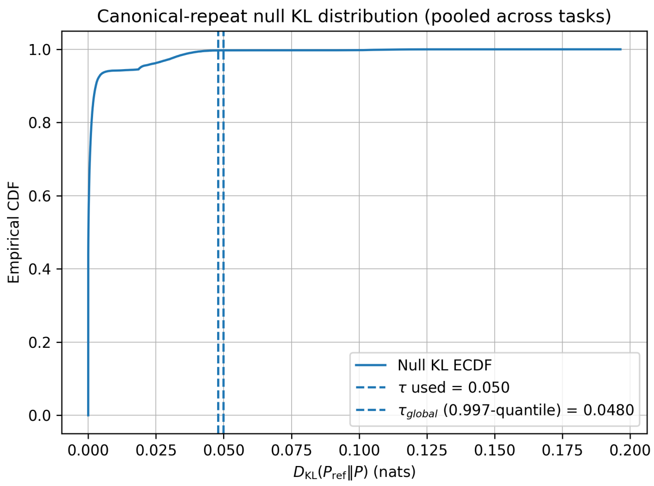

Figure 2: Null KL empirical cumulative distribution function (ECDF) from canonical repeats (pooled across tasks).

F Fidelity

$$\mathcal{F}(U_{\text{ref}}, U_{\text{gen}}) = \left|\frac{1}{d}\,\text{Tr}\left(U_{\text{ref}}^\dagger U_{\text{gen}}\right)\right|^2, \qquad d = 2^{n_q}, \tag{6}$$

where n_q is the number of qubits. It measures global similarity at the level of the implemented unitary.

G Pass@1 vs Pass@5 Comparisons

Pass@1 measures top-1 solution correctness, while Pass@5 measures correctness across the top 5 generated solutions.

Observations: Multiple attempts significantly increase correctness. Larger models (GPT-5.1, DeepSeek-R1) benefit more consistently from Pass@5.

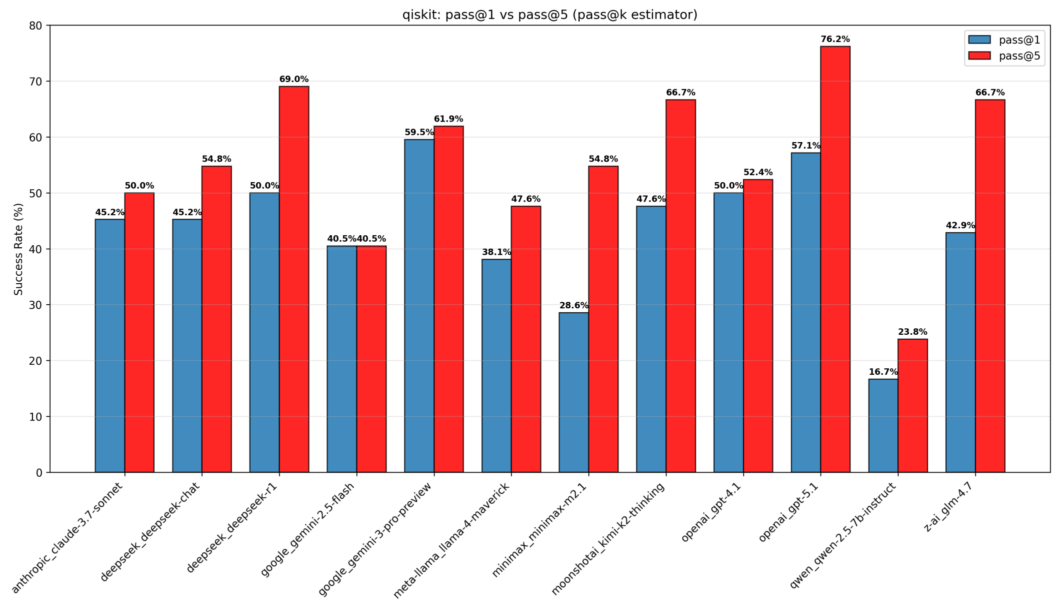

Figure 3: Pass@1 vs Pass@5 on Qiskit tasks. The gap highlights tasks where multiple attempts improve success rates.

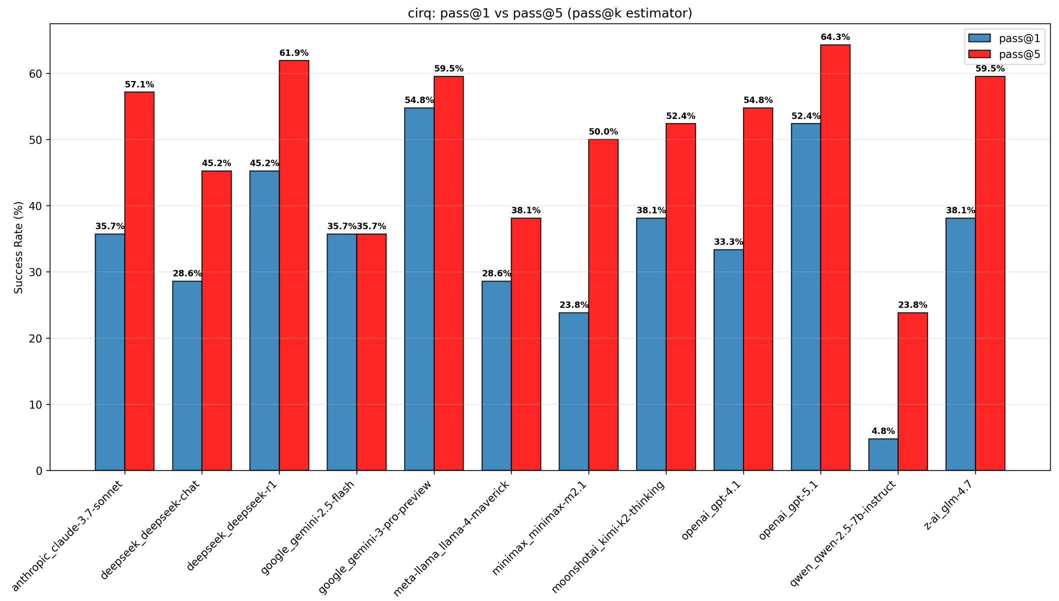

Figure 4: Pass@1 vs Pass@5 on Cirq tasks. Multi-attempt generation is particularly helpful for intermediate-sized models.

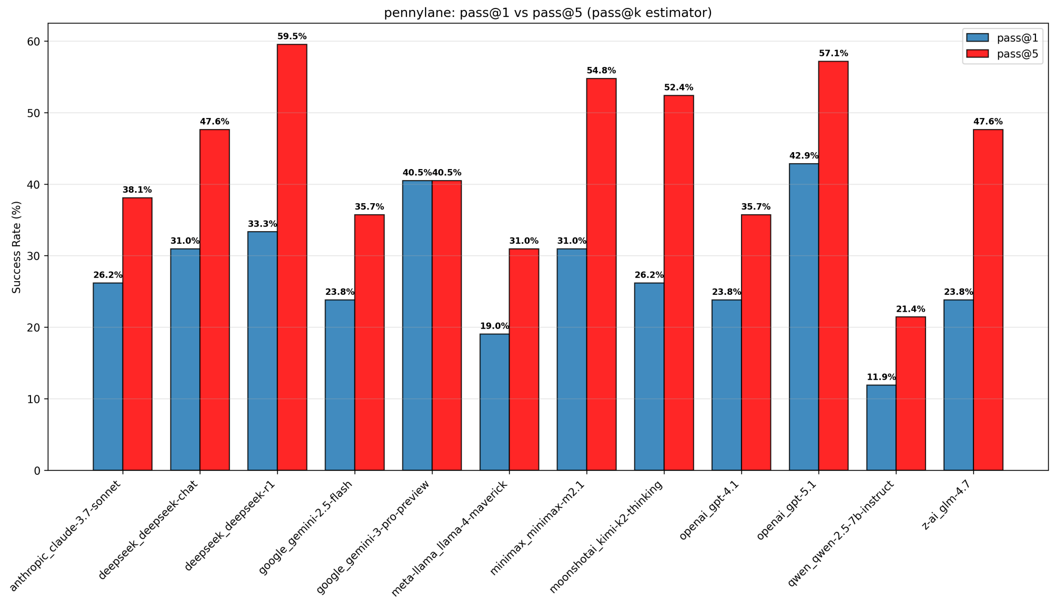

Figure 5: Pass@1 vs Pass@5 on Pennylane tasks. The largest improvements appear for smaller models with weaker initial performance.

H PER-TASK HEATMAPS

PASS@1 HEATMAPS

Observations: The Pass@1 heatmaps provide a strict view of first-attempt reliability, highlighting tasks where models succeed immediately versus those that fail on the first try. We next evaluate the same tasks under additional retries (Pass@5) to quantify how much performance improves when models are allowed multiple attempts and to visualize this progress at the task level.

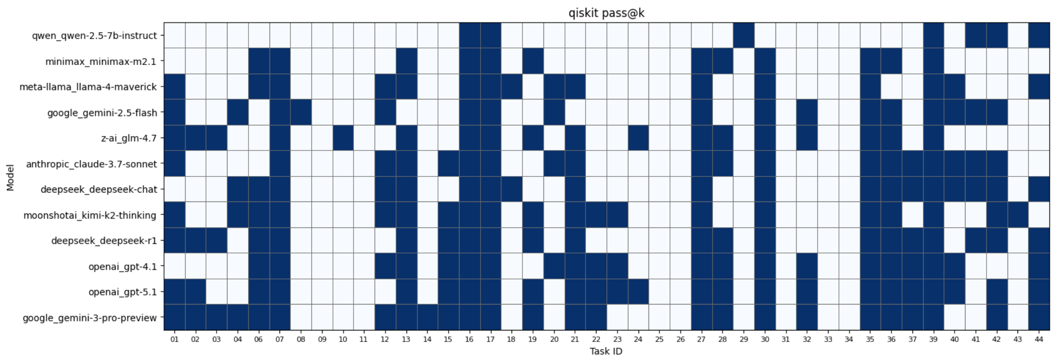

Figure 6: Per-task Pass@1 results for Qiskit.

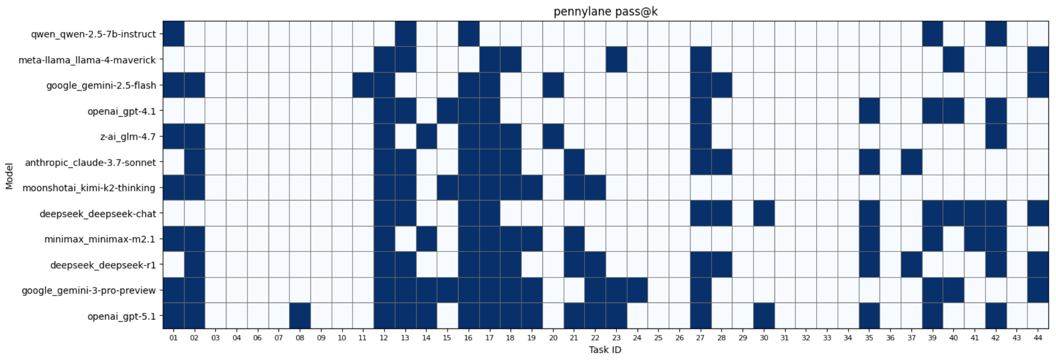

Figure 7: Per-task Pass@1 results for Pennylane.

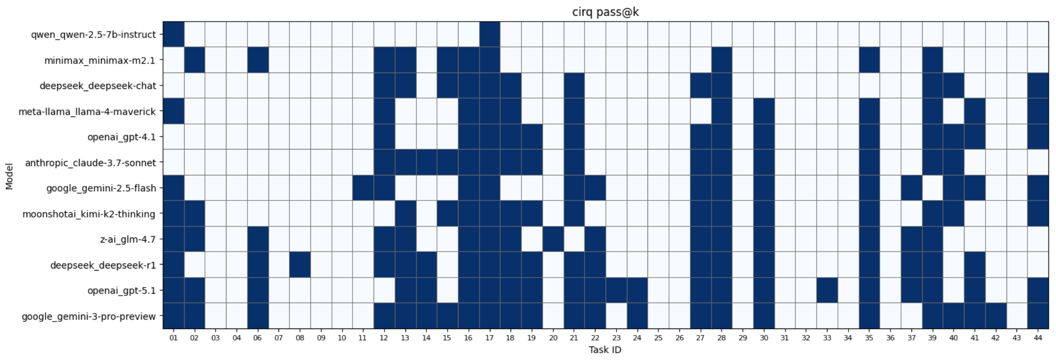

Figure 8: Per-task Pass@1 results for Cirq. Trends are similar to Pennylane and Qiskit.

PASS@5 HEATMAPS

Observations: The Pass@5 heatmaps show that many tasks unsuccessful at Pass@1 become solvable when the model is given multiple attempts. This pattern indicates that correct solutions are often present among a small set of candidates, and that a portion of failures at Pass@1 reflects sampling variability or correctable implementation details rather than a complete lack of the underlying quantum logic.

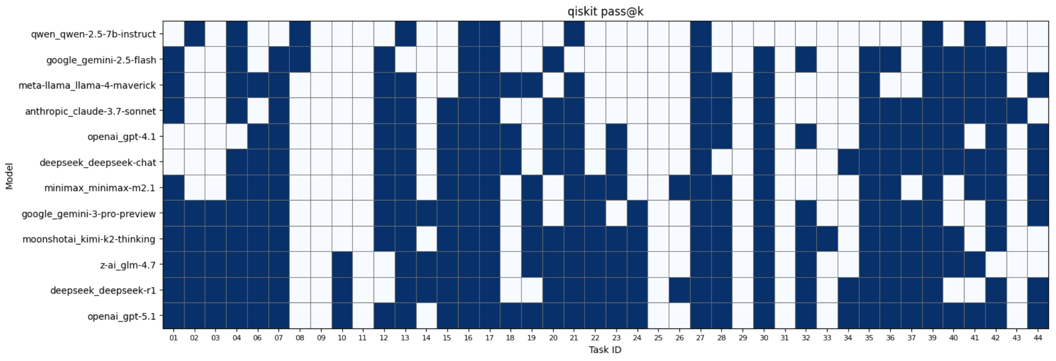

Figure 9: Per-task Pass@5 results for Qiskit.

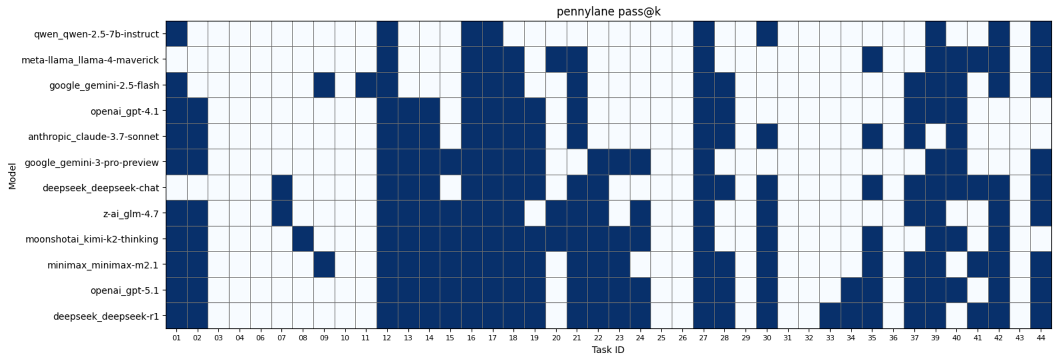

Figure 10: Per-task Pass@5 results for Pennylane.

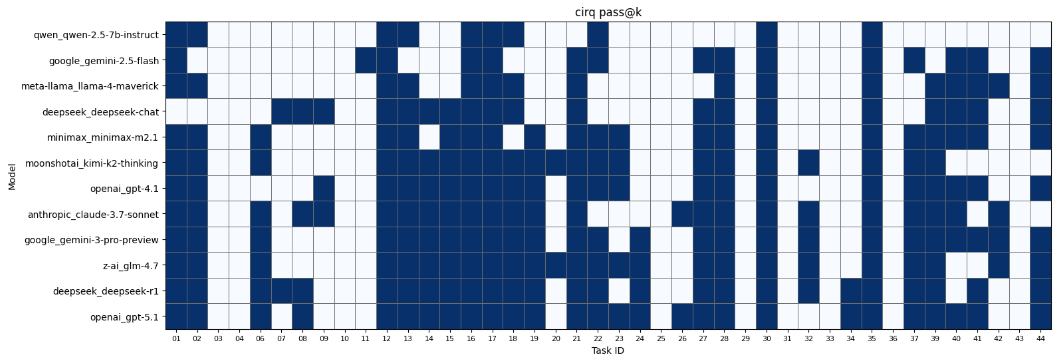

Figure 11: Per-task Pass@5 results for Cirq.

Table 6: **Pass@5 by framework.** Accuracy (%) over benchmark tasks.

Model	**Qiskit**	**Cirq**	**Pennylane**
`Gemini-3-Pro`	61.9	59.5	40.5
`GPT-5.1`	**76.2**	**64.3**	57.1
`DeepSeek-R1`	69.0	61.9	**59.5**
`MoonshotAI-Kimi-K2-Thinking`	66.7	52.4	52.4
`Claude-3.7-Sonnet`	50.0	57.1	38.1
`GPT-4.1`	52.4	54.8	35.7
`DeepSeek-Chat`	54.8	45.2	47.6
`Z-ai-GLM-4.7`	66.7	59.5	47.6
`Gemini-2.5-Flash`	40.5	35.7	35.7
`Llama-4-Maverick`	47.6	38.1	31.0
`MiniMax-M2.1`	54.8	50.0	54.8
`Qwen-2.5-7B-Instruct`	23.8	23.8	21.4

I PREFILL VS NO-PREFILL

We evaluate two prompting conditions for all models and frameworks:

- **Prefill**: the prompt includes required imports, function signature, and minimal boilerplate.
- **No-prefill**: the model generates the full solution from scratch.

Observations: Prefill often improves performance for smaller models and reduces variance in outputs. Larger models are less sensitive to the scaffolding. Although, the benefit of prefill was not consistent across models and frameworks.

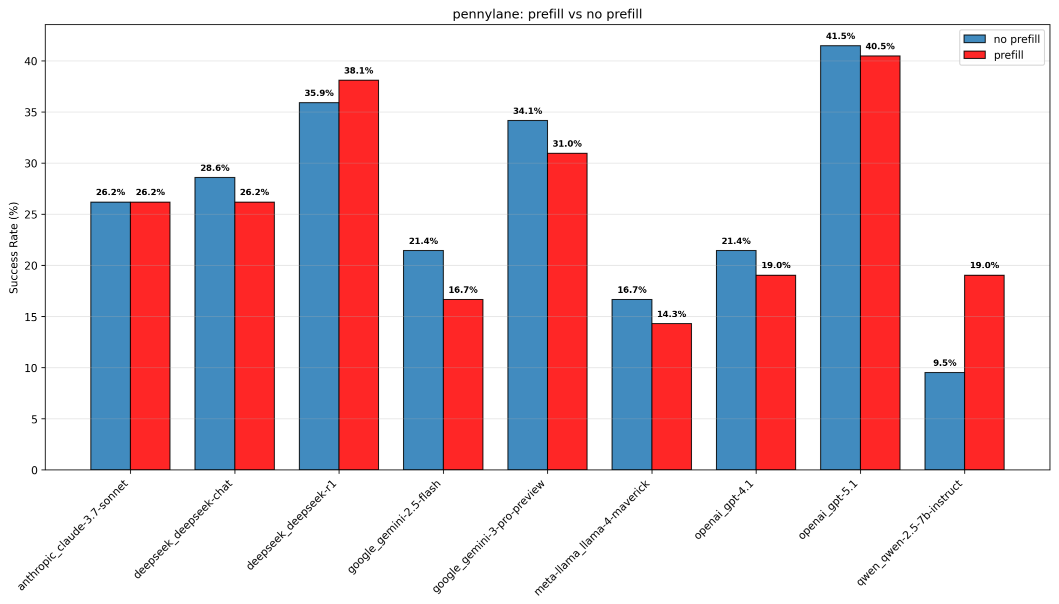

Figure 12: Prefill vs no-prefill comparison on Pennylane tasks.

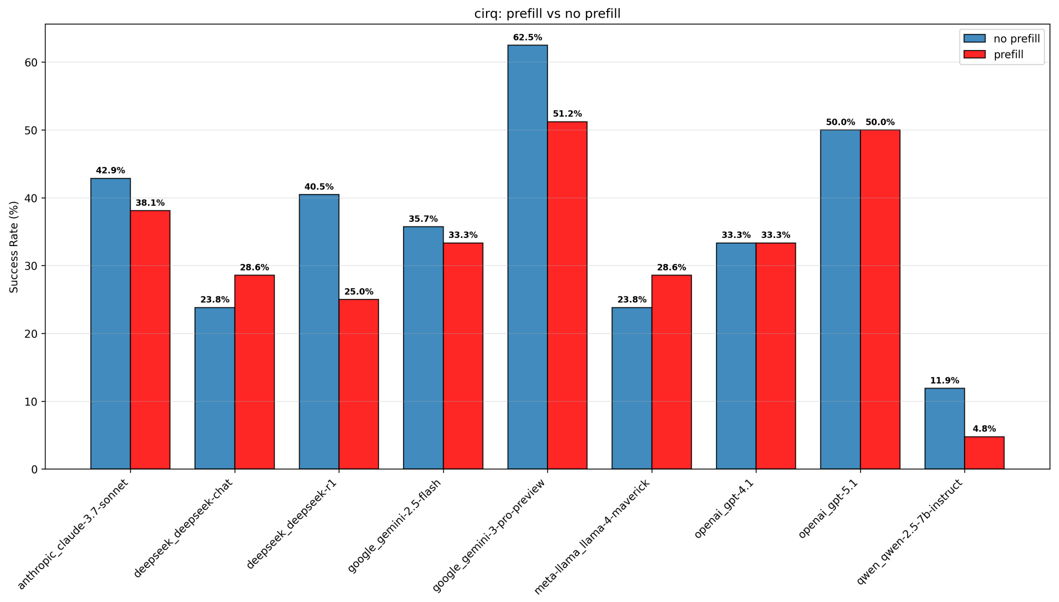

Figure 13: Prefill vs no-prefill comparison on Cirq tasks.

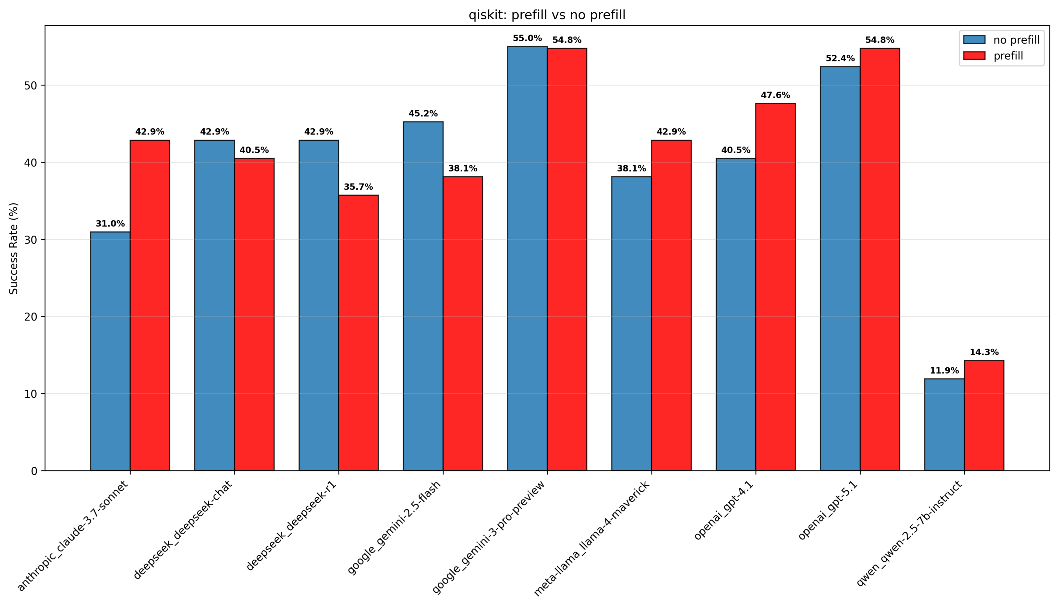

Figure 14: Prefill vs no-prefill comparison on Qiskit tasks.

J Error Distributions

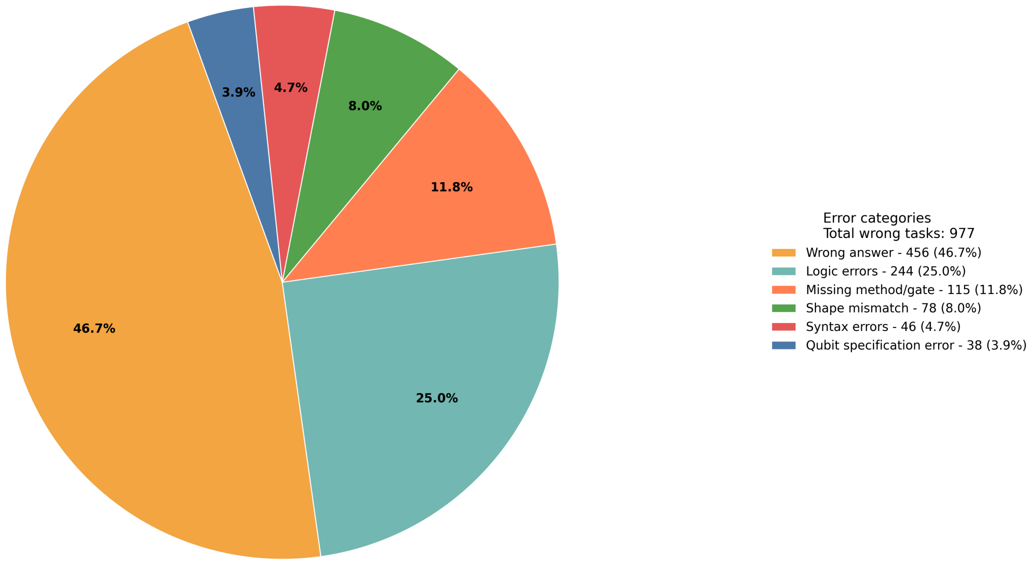

Figure 15: Distribution of errors across all frameworks for pass@1 results.

Observation: Figure 15 shows that most Pass@1 failures are driven by semantic mistakes—wrong answers (46.7%) and logic errors (25.0%). A sizable fraction, however, comes from more direct implementation/API issues, including missing methods/gates (11.8%), shape mismatches (8.0%), syntax errors (4.7%), and qubit specification errors (3.9%). This explains why feedback loops present a real opportunity for models to improve their results when the exact error is provided for the model to fix in the subsequent attempt. In the next section, we report the performance of all models under the feedback-loop setting and quantify the resulting gains.

K Feedback-Loop Results

We applied up to 5 repair attempts via feedback loops.

Observations: As shown in the following plots, the feedback loop substantially improves model outputs by steering subsequent generations toward corrections: models often fix earlier conceptual mistakes and replace deprecated or incorrect framework usage after receiving feedback. While a subset of tasks remains challenging, the dominant effect of feedback is a consistent shift toward more accurate, framework-compliant solutions. Darker cells in the figures below correspond to higher success rates.

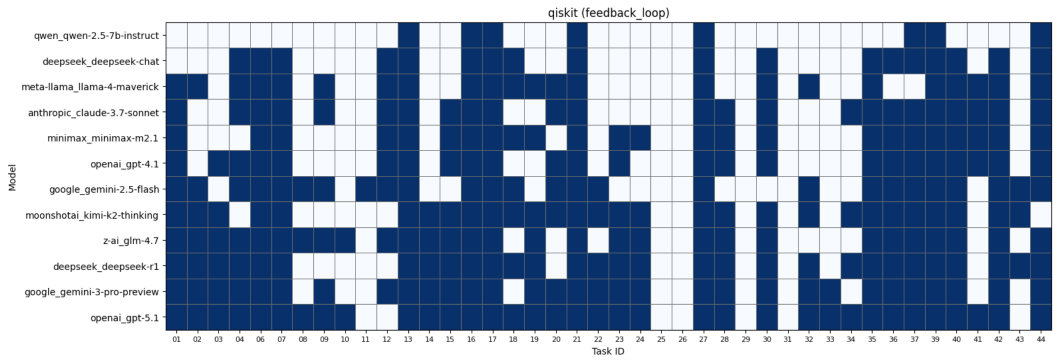

Figure 16: Pass@1 after feedback-based repair on Qiskit tasks.

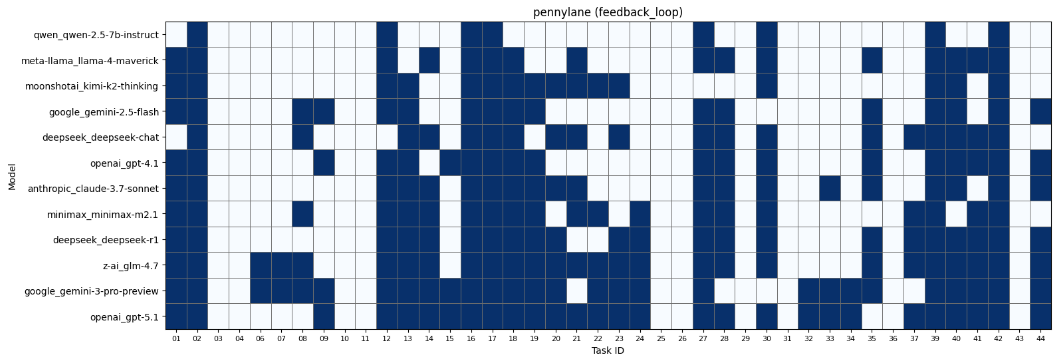

Figure 17: Pass@1 after feedback-based repair on Pennylane tasks.

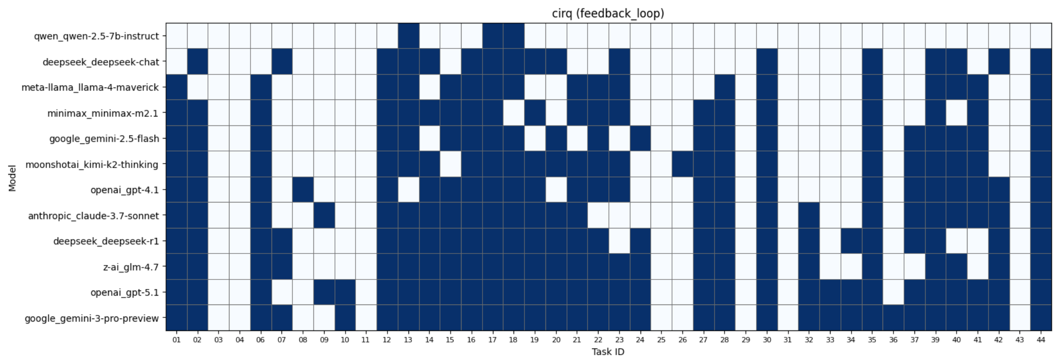

Figure 18: Pass@1 after feedback-based repair on Cirq tasks.

Observations: (i) Across the following figures, performance increases monotonically with the number of feedback-loop attempts, indicating that iterative feedback generally improves functional correctness. **(ii)** The largest gains typically occur early (attempts 1→2), followed by diminishing returns after ∼3 attempts. **(iii)** Qiskit (Fig. 19) shows earlier saturation for the strongest models, with curves flattening around attempts 3–4, whereas Pennylane (Fig. 20) and Cirq (Fig. 21) often exhibit more gradual improvements through attempts 4–5. **(iv)** Feedback reduces the spread among mid/high-performing models (several curves converge by attempts 4–5), but the weakest models show limited benefit and plateau quickly, suggesting persistent failure modes that retries do not resolve.

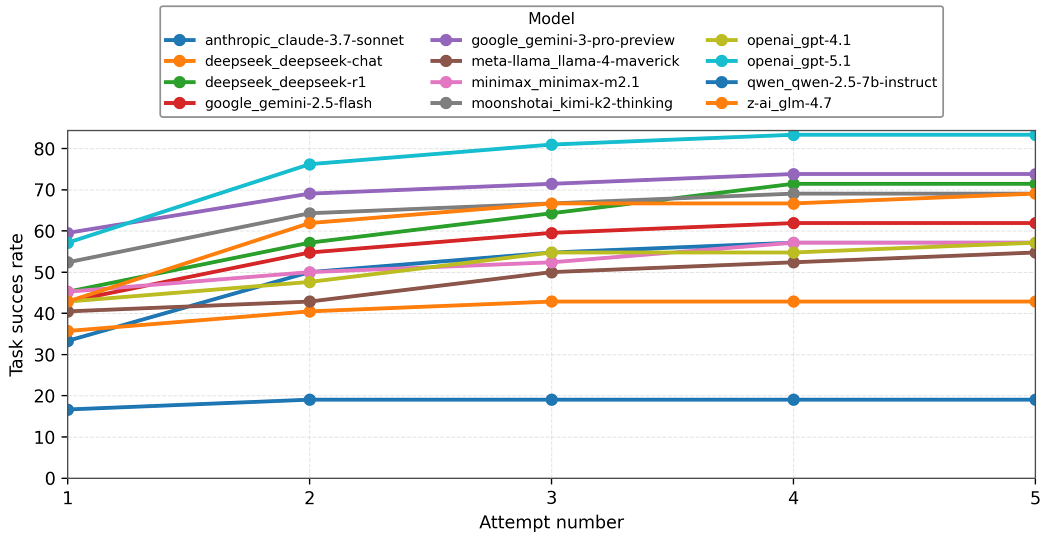

Figure 19: Improvements after feedback-loop attempts on Qiskit tasks.

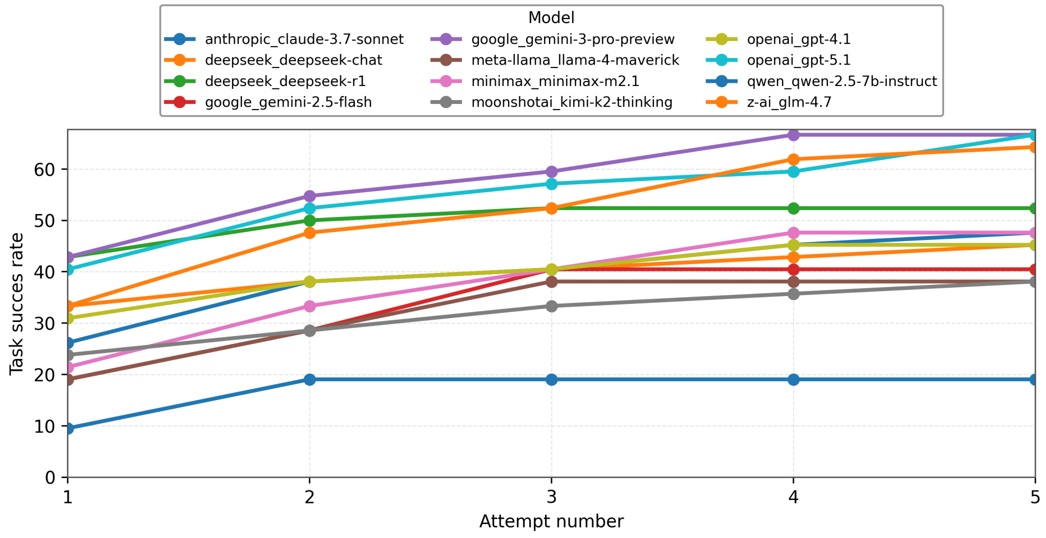

Figure 20: Improvements after feedback-loop attempts on Pennylane tasks.

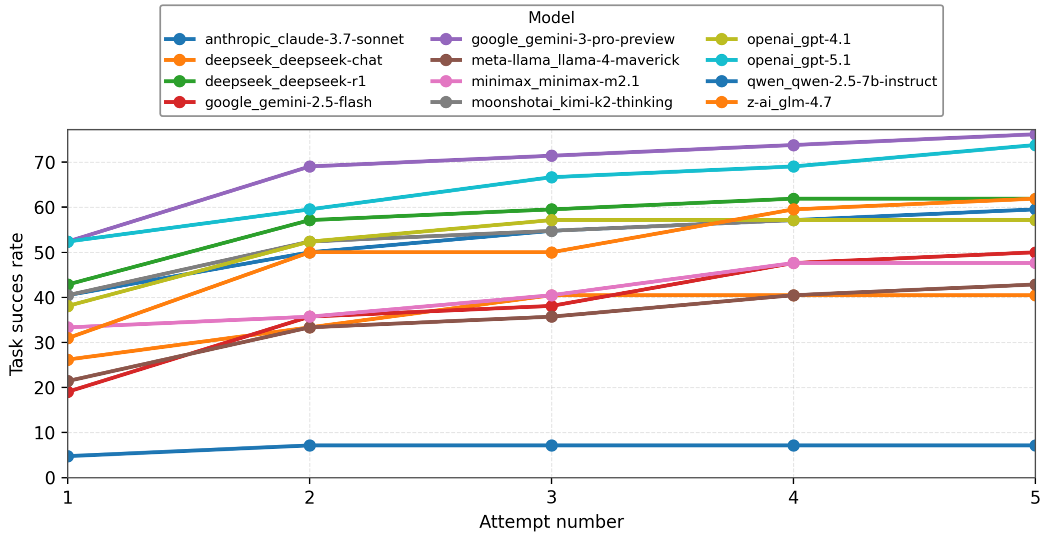

Figure 21: Improvements after feedback-loop attempts on Cirq tasks.

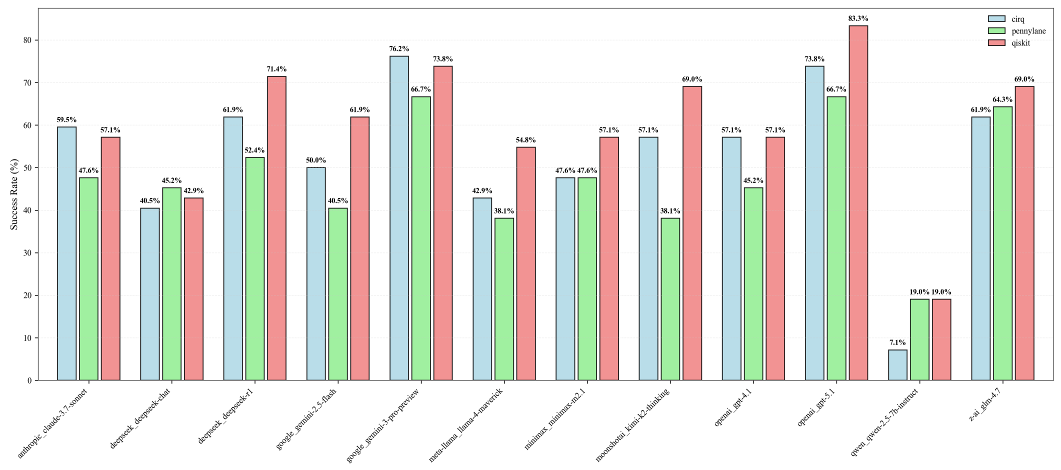

Figure 22: Aggregate success rates after up to 5 feedback-based repair attempts across all frameworks. Feedback reduces the performance gap between weaker and stronger models.

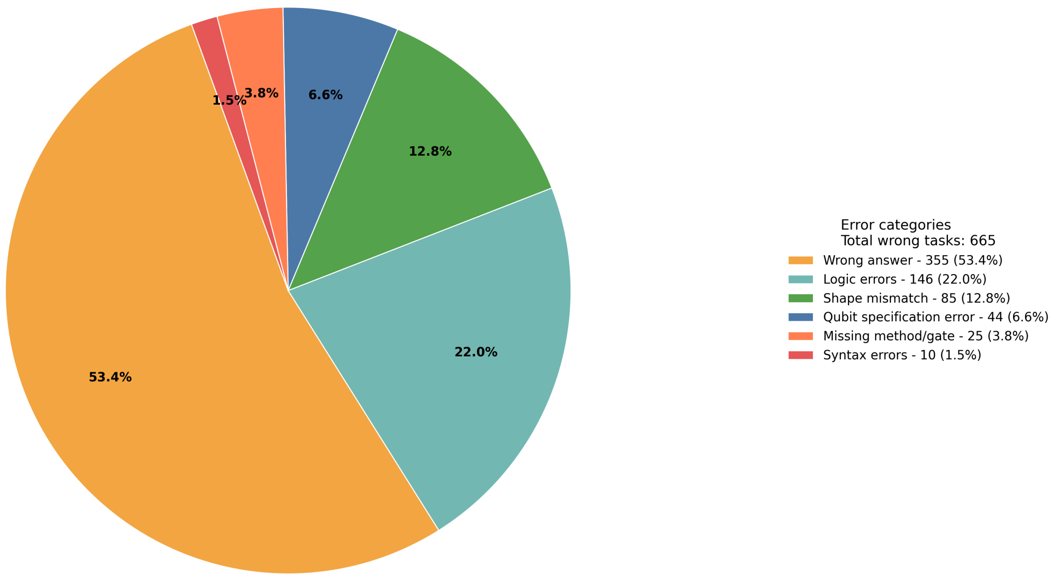

Figure 23: Distribution of errors across all frameworks after feedback loop.

Observations: After the feedback loop, as shown in Fig. 23 the total number of wrong tasks decreases substantially, from 977 to 665, indicating that iterative feedback helps repair a meaningful fraction of failures. The remaining errors are now dominated by deeper semantic issues, with wrong answers accounting for 53.4% of all failures, followed by logic errors (22.0%) and shape mismatches (12.8%). In contrast, surface-level implementation issues, such as missing methods/gates (3.8%) and syntax errors (1.5%), become much less frequent. Overall, this distribution suggests that the feedback loop is effective at fixing easily identifiable coding mistakes, while the main remaining challenge lies in achieving correct reasoning and functional behavior.