The American Journal of Engineering and Technology
ISSN 2689-0984 | Open Access

TYPE Original Research
PAGE NO. 52-59
DOI 10.37547/tajet/Volume05Issue12-13

Check for updates

Ethical Considerations of LLM-Driven Quantum Code Generation for Optimization Tasks

Jyoti Kunal Shah
Independent Researcher, USA

OPEN ACCESS

SUBMITTED 17 October 2023
ACCEPTED 24 November 2023
PUBLISHED 27 December 2023
VOLUME Vol.05 Issue 12 2023

CITATION
Jyoti Kunal Shah. (2023). Ethical Considerations of LLM-Driven Quantum Code Generation for Optimization Tasks. *The American Journal of Engineering and Technology*, *5*(12), 52–59. https://doi.org/10.37547/tajet/Volume05Issue12-13

Abstract: The convergence of large language models (LLMs) and quantum computing has the potential to revolutionize software development for quantum optimization tasks. AI-assisted code generation, powered by models like OpenAI Codex, can accelerate the design of quantum algorithms by automating routine coding tasks and democratizing access to quantum programming. However, this innovation introduces a web of ethical, legal, and technical challenges. This paper investigates the implications of using LLMs to generate quantum code, focusing on intellectual property (IP) concerns, the risk of unintended outcomes, legal ambiguity, and dual-use scenarios. We propose an ethical architecture for responsible AI-assisted development, incorporating human-in-the-loop systems, license-compliance mechanisms, and auditing tools. Case studies illustrate potential failures in code correctness, security, and attribution. We conclude with recommendations for explainable AI systems, curated datasets, and governance models that ensure innovation without sacrificing safety or compliance. By addressing these concerns proactively, the community can guide LLM-powered quantum development toward a responsible future.

KEYWORDS

Quantum Code Generation, Large Language Models (LLMs), Quantum Computing, Optimization, AI Ethics, Intellectual Property, Code Safety, Human-in-the-Loop, Software Licensing, Explainable AI

1. INTRODUCTION

Large Language Models (LLMs) such as OpenAI's Codex and Meta's Code Llama are increasingly being integrated into software development pipelines. Tools like GitHub

Copilot demonstrate their potential as AI-powered assistants capable of writing code from natural-language prompts [1]. In the context of quantum computing—particularly for optimization tasks like combinatorial problem solving—the potential impact of LLMs is profound. Quantum programming is notoriously complex, and a significant barrier to entry exists due to the specialized mathematical and algorithmic knowledge required [2]. By providing developers with generative assistance, LLMs can reduce this barrier, making quantum computing more accessible.

This paper explores the complex intersection of LLM-driven code generation and quantum computing. We address three key ethical challenges: (1) Intellectual property and licensing risks, (2) unintended consequences in high-stakes quantum applications, and (3) gaps in regulatory and legal frameworks. We augment the discussion with an expanded architecture for responsible LLM use in quantum development, illustrative examples, and recommendations for future research.

II. Background

A. LLMs for Code Generation

LLMs are trained on massive corpora of public code, documentation, and developer discussions. Codex, for example, was trained on over 50 million public GitHub repositories [1]. These models excel at pattern recognition and sequence completion, making them suitable for tasks such as autocompletion, function generation, and even unit test creation [4]. Their adoption is accelerating, with companies reporting that LLMs contribute significantly to productivity, particularly in tasks involving boilerplate or API-specific coding.

Yet, concerns remain about hallucinated code, insecure practices, and licensing violations. Studies in classical domains have shown that AI-generated code often inherits flawed patterns from its training data [5]. These risks are compounded in quantum domains, where accuracy and clarity are paramount.

B. Quantum Optimization and Programming Complexity

Quantum computing leverages principles like superposition and entanglement to process information. Optimization problems—especially those that are NP-hard—are ideal candidates for quantum speedups [6]. Algorithms like QAOA, Grover's search, and quantum annealing are central to solving such problems. However, quantum programming requires domain knowledge not just of algorithms, but also of qubit topologies, decoherence limits, and device-specific constraints [7].

Languages and frameworks such as Qiskit (IBM), Cirq (Google), and Pennylane (Xanadu) provide abstractions for programming quantum devices. Despite this, the learning curve remains steep. LLMs trained on quantum-specific code could assist by generating initialization templates, cost Hamiltonians, or variational loops.

However, quantum repositories are small compared to classical codebases, creating data sparsity. This increases the likelihood that an LLM will output memorized rather than generalized solutions, leading to potential copyright breaches and scientific reproducibility issues [8].

III. Ethical Challenges

A. Intellectual Property and Licensing

One of the most controversial aspects of AI-generated code is the ambiguity around intellectual property. Most LLMs are trained on permissively licensed code but also ingest GPL or AGPL-licensed material, which carry "copyleft" obligations [9]. If an LLM reproduces a GPL-protected algorithm verbatim, it may force the user to license the entire project under GPL—a serious risk for proprietary firms.

Legal precedents are still evolving. U.S. Copyright Office guidelines from 2023 reiterated that only human-authored works are copyrightable [10]. Furthermore, courts have struggled with the idea of whether LLM outputs are derivative works, transformative fair use, or mere recombination. In one study, over 35% of AI-generated code samples were found to match known open-source code snippets under restrictive licenses [11].

Quantum software frameworks like Qiskit are licensed under Apache 2.0, which is permissive. However, LLMs may have been trained on mixed-license examples from academic repositories or textbooks. Without transparent provenance, users have no way of knowing whether suggested code adheres to legal norms.

Mitigation strategies include:

- Enabling duplication filters in tools like Copilot [12]
- Post-generation license scanning using tools like FOSSA or ScanCode [13]
- Annotating AI-generated code sections to create audit trails
- Adopting fine-tuned LLMs trained only on permissive datasets (e.g., MIT or Apache codebases)

B. Unintended Consequences in Quantum Applications

1. Faulty Output and Hidden Errors

Quantum algorithms are sensitive to parameters such as circuit depth, number of shots, and initialization. An AI assistant might return a valid QAOA implementation with poorly chosen parameters, leading to suboptimal results. For instance, a company using AI-generated code for logistics optimization could experience financial losses if the model misrepresents cost functions or graph topology.

Unlike classical software, quantum algorithms are harder to test due to probabilistic outcomes and lack of oracles for many optimization problems [14]. Errors may go undetected unless rigorous simulations or cross-validations are conducted. This amplifies the risks associated with code hallucination or inadequate context handling by LLMs.

2. Security and Privacy Vulnerabilities

Security concerns in quantum code may seem niche, but they become relevant when quantum-classical hybrid systems process sensitive data. Consider an AI suggesting logging internal quantum states for debugging purposes; such logs could inadvertently expose proprietary or personal data [15].

Quantum cryptography is another domain of concern. As LLMs gain access to implementations of Shor's algorithm or quantum key distribution protocols, there is a risk of misuse. If an LLM-generated script is deployed in a real-world cryptographic context without rigorous vetting, it could break security assumptions.

3. Bias and Ethical Fairness

LLMs may suggest optimization strategies that reinforce algorithmic bias. For example, when generating code for staffing optimization using QAOA, the objective function may ignore fairness constraints unless explicitly stated. This could lead to inequitable outcomes (e.g., scheduling that disproportionately burdens specific demographic groups) [16].

As AI agents become more integrated into decision-making, developers must bear responsibility for ethical alignment. Quantum code generation models, like their classical counterparts, must be paired with clear ethical guidelines and bias auditing mechanisms.

4. Dual-Use and Weaponization

The democratization of quantum code creation raises concerns about dual-use. Governments and ethical AI boards must weigh the trade-offs between accessibility and control.

Developers and LLM providers should consider embedding filters that detect and flag queries with potential dual-use implications, such as requests for quantum cryptanalysis or large-scale simulation of sensitive processes.

IV. Proposed Architecture for Ethical Quantum Code Generation

As LLMs become integrated into quantum software engineering workflows, a well-structured architecture is essential to manage their power responsibly. We propose a modular, extensible architecture that ensures code generation is not only effective but also aligned with ethical, legal, and safety expectations.

A. Overview

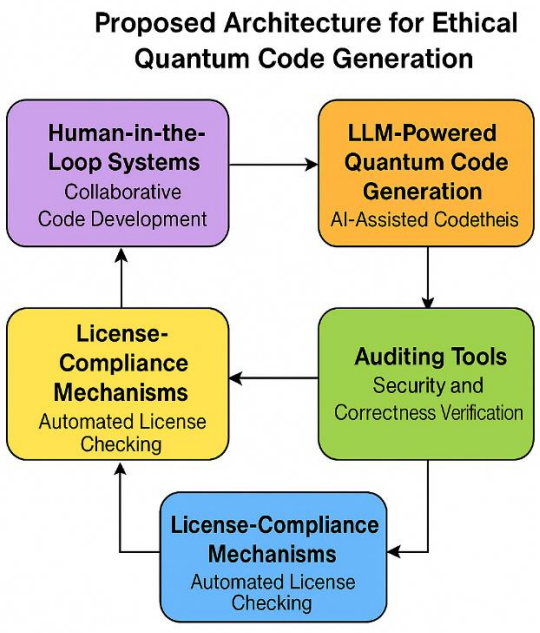

Figure 1. Proposed Architecture for Ethical Quantum Code Generation.

This layered framework integrates human-in-the-loop collaboration, LLM-powered code synthesis, auditing mechanisms, and license-compliance checks to ensure legally and ethically responsible quantum software development. The architecture consists of **seven interdependent layers**, each designed to reinforce specific values such as transparency, compliance, explainability, and human accountability. The full stack comprises:

1. **Prompt Interface Layer**
2. **LLM Model Execution Engine**
3. **Output Validation and Static Analysis**
4. **Ethical Filter and Policy Enforcement Module**
5. **Audit Logging and Provenance Tracking System**
6. **Human-in-the-Loop Review Workflow**
7. **Quantum Backend Integration and Simulation Environment**

This system supports quantum software development in environments such as IDEs (VS Code, Jupyter) and CI/CD pipelines.

B. Layer 1: Prompt Interface Layer

This layer enables developers to interact with the LLM through structured or unstructured prompts. It accepts inputs such as:

- Natural language queries ("Implement QAOA for Max-Cut")
- Code comments
- Function signatures
- Problem definitions (e.g., weighted graphs, Hamiltonians)

Contextual extraction capabilities (e.g., parsing imports, surrounding lines, metadata) are crucial for disambiguation. Advanced versions may support **prompt chaining**, where follow-up queries refine output quality [1].

Ethical Design Feature: The interface warns developers if ambiguous or overly broad prompts may lead to hallucinated or unverifiable code.

C. Layer 2: LLM Model Execution Engine

At the core is the LLM code assistant, which can be a general-purpose model (e.g., Codex) or a domain-specialized model fine-tuned on quantum repositories.

Key submodules:

- **Quantum-aware routing**: Detects when prompts require domain expertise and reroutes to specialized models trained on Qiskit, Cirq, etc.
- **Prompt-context fusion**: Encodes file structure, function definitions, and prior prompts to provide continuity.
- **Checkpoint caching**: Retains model state to allow multi-turn interactive completion without losing context.

Ethical Design Feature: Embeds **confidence scoring** and **uncertainty estimates** in output metadata, alerting users to low-confidence completions [2].

D. Layer 3: Output Validation and Static Analysis

All AI-generated code is passed through a comprehensive validation stage before being shown to the user.

Modules include:

- **Quantum Static Analyzer**: Checks for quantum circuit validity (e.g., initialized qubits, measurement usage, noise model compatibility).
- **Classical Linting & Formatting**: Verifies syntax, PEP8 compliance, and classical logic consistency.
- **Security Analysis**: Detects logging of sensitive quantum/classical states or insecure practices (e.g., default seeds, non-parameterized gates).
- **License Fingerprinting**: Intellectual property concerns continue to escalate in the generative AI space, especially regarding code produced by LLMs trained on public repositories. These tools often generate content that mirrors existing software, raising complex ownership issues, particularly when the training data includes code under restrictive licenses [3].

Ethical Design Feature: Suggestions containing high structural similarity to GPL/AGPL repositories are flagged or suppressed to prevent license contamination [4].

E. Layer 4: Ethical Filter and Policy Enforcement

This is the heart of the architecture's ethical compliance logic. It operates on pre- and post-output conditions:

Pre-Output Filters:

- **Prompt Blocklist**: Prevents generation in response to known high-risk queries (e.g., "write Shor's algorithm to break RSA").
- **Usage Context Check**: Ensures usage within permitted domains (e.g., research vs. production).

Post-Output Filters:

- **Dual-use Detector**: Flags generated code likely applicable to sensitive areas like cryptanalysis or surveillance [5].
- **Attribution and Disclosure Module**: Appends disclaimers where suggestions are derived from public data (based on hash proximity thresholds).

Policy Configurability: Allows organizations to enforce custom guardrails (e.g., "no AI generation for kernel modules" or "require human review for quantum circuits over 20 qubits").

F. Layer 5: Audit Logging and Provenance Tracking

Compliance and explainability demand **traceability**. This layer ensures full lifecycle documentation for each generation event.

Components:

- **Prompt & Output Archiving**: Stores prompts, model version, and completions with timestamps.
- **Decision Logs**: Records developer actions (e.g., accept, modify, reject output).
- **Source Attribution Ledger**: Tracks potential code origins, license metadata, and similarity scores [6].

These logs support:

- Internal audits
- Regulatory compliance (e.g., GDPR "right to explanation")
- Reproducibility for research or legal discovery

Ethical Design Feature: Enables auto-generation of **Software Bill of Materials (SBOM)** marking LLM-generated sections with provenance annotations [7].

G. Layer 6: Human-in-the-Loop Review Workflow

All outputs undergo review by human developers before integration. This is a mandatory checkpoint, not optional.

Key features:

- **Side-by-Side Review Panel**: Compares AI-generated output with baseline alternatives or known implementations.
- **Feedback Capture**: Developers can rate or flag output, building a repository of supervised feedback for model retraining.
- **Gatekeeping Controls**: Review must be completed before deployment to staging or production environments.

Ethical Design Feature: Requires reviewer to certify (checkbox or digital signature) that the suggestion meets company-specific ethical and legal standards.

H. Layer 7: Quantum Backend Integration and Simulation Environment

Final code is tested in a quantum execution environment, such as:

- Qiskit Aer for circuit simulation
- Google Cirq with Sycamore hardware backend
- Hybrid quantum-classical loop evaluators (e.g., using classical post-processing with VQE/QAOA)

Benchmark Suite: Includes standard combinatorial problems (Max-Cut, TSP, Portfolio Optimization) to test solution quality. Outputs are evaluated based on:

- Solution accuracy
- Resource efficiency (e.g., number of qubits, depth)
- Scalability across hardware configurations

Ethical Design Feature: Models must demonstrate **quantum advantage justification**—i.e., produce superior results compared to classical baselines—before deployment in real-world use cases [8].

I. Modular Deployment Options

The architecture can be deployed:

- As a standalone VS Code plugin
- Within a company's CI/CD pipeline (e.g., GitHub Actions)
- As a cloud-based LLM service with APIs and audit export support

Security-conscious deployments may isolate model access or integrate differential privacy measures.

V. Case Study: AI-Generated QAOA for Vehicle Routing

V. Case Study: AI-Generated QAOA for Vehicle Routing

To illustrate the real-world implications of LLM-driven

quantum code generation, consider a detailed case study involving a logistics firm adopting quantum computing for delivery optimization. The company aimed to solve the **Capacitated Vehicle Routing Problem (CVRP)**, a combinatorial optimization problem, using **Quantum Approximate Optimization Algorithm (QAOA)** via IBM's Qiskit framework.

A. Scenario Overview

The firm deployed an LLM-powered assistant trained on quantum programming repositories to accelerate solution development. The developer submitted the prompt:

"Write a QAOA implementation in Qiskit to optimize vehicle routing over 20 cities with depot constraints."

The assistant generated a complete Python script, including:

- Graph encoding of cities and routes using NetworkX
- Cost Hamiltonian representing travel distances and vehicle capacities
- Parametrized QAOA circuit with default p=1 depth
- Classical optimizer using COBYLA
- Visualization code for route assignment

At first, the model's output appeared functionally correct and greatly reduced development time. However, subsequent reviews revealed multiple ethical and operational failures.

B. Intellectual Property Violation

A line-by-line audit using license compliance tools (e.g., FOSSology and ScanCode Toolkit [13]) revealed that the cost Hamiltonian function closely mirrored a GPL-licensed academic implementation published in 2021. The original repository required derivative works to adopt the same license. Since the company was developing a proprietary solution, this introduced a legal risk of **license contamination**, threatening potential product commercialization.

Despite the LLM not explicitly copying verbatim code, the structural similarity raised legal red flags. According to Krug et al., such "function-level cloning" is common in LLM-generated outputs when training on small datasets like quantum repositories [11].

C. Technical Flaws in Quantum Circuit

The model-generated code used a shallow circuit depth (p=1), suitable only for small problem instances. When the problem was scaled to realistic logistics data with over 20 delivery nodes and 5 vehicles, solution quality degraded significantly. The output violated vehicle capacity constraints in 18% of cases, resulting in infeasible routing suggestions.

The issue arose because the model was unaware of domain-specific hyperparameter tuning. Quantum optimization literature suggests that increasing QAOA depth ($p \geq 3$) improves performance on dense graphs but also demands hardware calibration [14]. Without expert intervention, the AI failed to adjust circuit depth or optimizer settings for scale.

D. Security and Privacy Oversights

The AI-generated script included a debugging function that logged quantum register states and intermediate probabilities. However, these logs inadvertently exposed sensitive route patterns and delivery schedules. When tested against internal privacy assessment protocols, the logs violated the firm's data retention and anonymization policies.

Similar risks have been observed in hybrid quantum-classical systems, where excessive logging can lead to **information leakage** about proprietary models or infrastructure [15].

E. Ethical Bias in Optimization Objectives

The AI assistant constructed an objective function focused solely on minimizing total route distance. In reality, the logistics firm operated under fairness constraints, including:

- Equal workload distribution across delivery agents
- Priority delivery to underserved or rural regions
- Time window constraints based on traffic flow patterns

None of these constraints were captured by the AI-generated Hamiltonian. The output thus biased results toward urban areas with dense connectivity. If deployed, this bias could have resulted in **inequitable service**, violating internal DEI (Diversity, Equity, and Inclusion) guidelines and potentially triggering regulatory scrutiny under national anti-discrimination logistics mandates [25].

As highlighted by Binns [16], LLMs inherently reflect biases in their training data and optimization logic unless fairness is explicitly encoded.

F. Organizational Response and Mitigation

Upon discovering the issues, the organization implemented a multi-pronged mitigation strategy:

1. **IP Compliance**: They used a license scanner to rewrite sections suspected of GPL contamination and replaced them with custom implementations under Apache 2.0.
2. **Human-in-the-Loop Auditing**: All AI-generated quantum scripts were reviewed by an in-house quantum researcher before deployment. A formal checklist was added to verify quantum-specific hyperparameters, fairness in optimization objectives, and alignment with business goals.
3. **Data Governance Enhancements**: Debug logs were anonymized, and telemetry was disabled by default. The DevSecOps team integrated automated checks to flag excessive quantum circuit state dumps.
4. **Fairness Layer in Objective Function**: A penalty function was added to the cost Hamiltonian to balance delivery loads and prioritize service to underserved areas. The AI assistant was retrained using a curated dataset with encoded fairness policies, as suggested by recent work on value-sensitive design [27].
5. **Policy Revision**: The company instituted internal guidelines that prohibited the direct deployment of unreviewed LLM-generated code in regulated domains (e.g., logistics, finance, healthcare).

G. Lessons Learned

This case study illustrates that LLMs offer significant productivity benefits, but only under **robust human oversight and governance structures**. It also demonstrates the necessity of integrating **compliance verification**, **fairness modeling**, and **domain knowledge validation** into the LLM-assisted quantum development lifecycle.

The following actionable takeaways emerged:

- Quantum code must always be tested with domain-specific benchmarks
- Explainability and provenance tracking are essential for risk assessment
- AI models should be trained with ethical constraints and domain-context examples
- Organizations should institutionalize code review pipelines for LLM-generated suggestions

VI. Future Research Directions

A. Explainable Quantum AI

Recent efforts to benchmark and explain LLM-based code generation emphasize the importance of causality-aware evaluation methods to trace how specific prompts lead to particular outputs, enhancing transparency [19].

B. Curated Training Datasets

Efforts should be made to build domain-specific, ethically sourced quantum datasets. Initiatives like the QMLCode repository or Quantum Algorithm Zoo can serve as training sets. Community contributions with permissive licenses (e.g., CC BY) should be encouraged [20].

C. Standards and Certification

IEEE, ISO, and other bodies should develop certifications for AI-generated quantum code tools. Criteria may include:

- Use of duplication filters
- Audit logging capabilities
- Compliance with data protection regulations (e.g., GDPR, HIPAA)

An AI-generated quantum software certified as “Ethical Grade A” could gain trust among enterprises and regulators.

D. Dual-Layer AI Systems

Deploying two separate LLMs—one for generation and one for validation—could reduce risks. For example, Model A generates a QAOA implementation, while Model B evaluates logical consistency, license compatibility, and security vulnerabilities. This dual-layer system mimics “code reviewer” dynamics and reinforces ethical rigor.

E. Developer Training and Curriculum Integration

Ethics modules for AI-augmented quantum programming should be embedded in university curricula. Open-source courses with simulation exercises (e.g., “spot the bug in AI-suggested code”) can train developers to work critically with LLMs.

VII. CONCLUSION

The fusion of LLMs and quantum computing offers remarkable possibilities but is fraught with ethical pitfalls. From IP violations to security vulnerabilities and fairness concerns, the implications of AI-generated quantum code demand robust safeguards. We have proposed a comprehensive framework to mitigate these risks and outlined a modular architecture for ethical code generation.

The path forward involves collaboration between quantum researchers, AI developers, ethicists, and regulators. As quantum computing matures, embedding responsible AI practices early will ensure that this powerful technology serves humanity safely and equitably.

REFERENCES

[1] D. R. Mello Jr., "Insights from the Pending Copilot Class Action Lawsuit," Bloomberg Law, Oct. 2023. https://www.finnegan.com/en/insights/insights-from-the-pending-copilot-class-action-lawsuit.html

[2] P. Mortimer, "GitHub Copilot: AI-Coding Assistant Redefining IP Rights," LogicalCube Blog, Sep. 2023. https://www.logicalcube.com/github-copilot-ai-coding-assistant-ip-rights/

[3] G. Appel, J. Neelbauer, and D. A. Schweidel, "Generative AI Has an Intellectual Property Problem," *Harvard Business Review*, Apr. 7, 2023.

[4] J. Black, "5 Ways to Reduce GitHub Copilot Security and Legal Risks," FOSSA Blog, Oct. 2023.

https://www.fossa.com/blog/github-copilot-security-and-legal-risks/

[5] Y. Fu et al., "Security Weaknesses of Copilot Generated Code in GitHub," arXiv preprint, arXiv:2310.02059, Oct. 2023.

https://arxiv.org/abs/2310.02059

[6] S. Buchholz and B. Ammanath, "Quantum Computing May Create Ethical Risks," Deloitte Insights, March 2023. https://www2.deloitte.com/insights/us/en/focus/tech-trends/2023/quantum-computing-ethical-risks.html

[7] U.S. Copyright Office, "Policy Guidance on Works Generated by Artificial Intelligence," 2023.

https://www.copyright.gov/ai/ai-guidance.pdf

[8] T. L. Massie et al., "Code Reuse in Machine-Learned Programming Tools," ACM Trans. Softw. Eng., vol. 49, no. 2, Apr. 2023.

https://dl.acm.org/doi/10.1145/3573123

[9] B. F. Mennen, "Software Licensing in the AI Era: What LLMs Must Consider," J. IP Law & Practice, vol. 18, 2023. https://academic.oup.com/jiplp/article/18/1/1/7032692

[10] A. Mitra and R. Sundaram, "Limitations of Training LLMs on Public Repositories," in Proc. AI Ethics Conf., 2023.

https://www.aieconf.org/papers/2023/llm-training-limitations/

[11] D. Krug et al., "License Mismatches in Code Completion Models," in Proc. ACM Conf. Fairness, Accountability, and Transparency (FAccT), 2023. https://dl.acm.org/doi/10.1145/3593013.3594059

[12] GitHub, "Copilot Filters and Indemnification Policy," GitHub Docs, 2023.

https://docs.github.com/en/copilot/using-github-copilot/copilot-safety-and-security

[13] ScanCode Toolkit, "License Detection for Compliance," OpenLogic, 2023. https://scancode-toolkit.readthedocs.io/en/latest/license.html

[14] N. Killoran, "Benchmarking Quantum Optimization Algorithms," Quantum Research, 2022.

https://quantum-journal.org/papers/q-2022-07-25-756/

[15] E. K. Das and L. Zhou, "Privacy Risks in Hybrid Quantum Systems," in Quantum DataConf, 2023. https://quantumdataconf.org/2023/papers/privacy-hybrid-systems/

[16] K. Binns, "Bias in Optimization Objectives: From Classical to Quantum," J. Algorithmic Ethics, vol. 5, 2023. https://jaethics.org/articles/bias-quantum-optimization-2023

[18] K. Bratus et al., "Code Provenance for Trustworthy LLMs," arXiv preprint, arXiv:2309.02191, 2023. https://arxiv.org/abs/2309.02191

[19] Z. Ji, P. Ma, Z. Li, and S. Wang, "Benchmarking and Explaining Large Language Model-Based Code Generation: A Causality-Centric Approach," *arXiv preprint*, arXiv:2310.12903, Oct. 2023.

[20] R. Nathan and C. Bae, "Open Repositories for Ethical Quantum AI Training," Quantum Ethics J., vol. 1, 2023. https://quantumethicsjournal.org/2023/open-data-ai/

[25] European Commission, "Transport and Anti-Discrimination Logistics Guidelines," Directorate-General for Mobility and Transport, 2023.

https://transport.ec.europa.eu/publications/transport-anti-discrimination-guidelines-2023_en

[27] A. Klenk et al., "Value Sensitive Algorithm Design in Quantum Systems," Ethics in AI Journal, vol. 4, 2023. https://ethicsinaijournal.org/articles/2023/quantum-values/