AGENTIC AUTOMATED ALGORITHM SEARCH FOR QUANTUM-COMPUTING-INTEGRATED MACHINE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

By leveraging pretrained LLM, we present a LLM-based multi-agent framework that automatically searches in program space to synthesize/find quantum-computing-integrated machine learning algorithms from classical origins. Our agentic system operates as an orchestrated set of agents with capabilities of tool use, integration of internet knowledge, and planning. The searching workflow then utilizes Monte Carlo Tree Search for guided exploration, a nested dual-loop procedure for robust code generation and debugging, as well as a Webscraper for up-to-date knowledge injection. Set a algorithmic seed as a classical multi-layer perceptron, the system generates quantum-hybrid and quantum-inspired models by iteratively proposing, validating, and refining code. With the empirical evidence, the system can find novel algorithms with high performance. By coupling LLM reasoning and coding with principled search and verification, our approach offers a practical path toward accelerated design of algorithms for quantum-computing-integrated machine learning.

1 INTRODUCTION

Meta-learning in machine learning utilizes prior experience to facilitate the design of algorithms (Hospedales et al., 2020). Prominent examples include hyperparameter optimization (Franceschi et al., 2018) and neural architecture search (NAS; Wistuba et al., 2019). Recently, Large Language Models (LLMs), a type of foundation model, have shown remarkable potential in reasoning and planning across a wide array of tasks (Yao et al., 2023; Ahn et al., 2024). This has led to an emerging paradigm of meta-learning with foundation models, which leverages their extensive pre-trained knowledge and automated coding abilities for tasks like algorithmic generation and improving (Romera-Paredes et al., 2024; Novikov et al., 2025).

Quantum computing harnesses properties like superposition and entanglement to enable powerful parallel processing and operate efficiently on data distributions. These quantum advantages have drawn significant attention in fields such as healthcare (Rani et al., 2023), medical image analysis, and pattern recognition (Wei et al., 2023). A primary goal of this research is to accelerate machine learning tasks and design more advanced quantum machine learning algorithms. To date, notable improvements have been achieved in parameter optimization, execution efficiency, and error rate reduction (Li et al., 2019; 2020; Parisi et al., 2022).

Quantum computers can compute the same tasks as classical computers but offer the potential for much faster runtimes due to quantum parallelism. Additionally, entanglement can enhance the modeling of correlations in structured data, as exemplified by the superior performance of quantum strategies in the CHSH game compared to their classical counterparts (Clauser et al., 1969). To increase the expressive power and representation capability of quantum machine learning models, it is necessary to develop architectures analogous to classical deep neural networks, which can learn high-level abstractions. To this end, and in alignment with the long-term goal of advancing artificial intelligence, we have developed a meta-learning framework with foundation models to search for novel quantum-computing-integrated machine learning algorithms.

Compared to single-agent systems, LLM-based Multi-Agent (LLMMA) systems utilize collective intelligence and differentiated skills to offer advanced capabilities (Wang et al., 2024; Hong et al.,

2023; Qian et al., 2024; Mandi et al., 2023; Zhang et al., 2024; Guo et al., 2024). LLMs have also risen to prominence in scientific discovery due to their expansive knowledge bases, reasoning capabilities, and natural language interfaces (AI4Science & Quantum, 2023). Inspired by Google DeepMind's FunSearch (Romera-Paredes et al., 2024), which searches for solutions to algorithmic problems via iterative code generation, we propose an LLM-based Multi-Agent system for quantum-computing-integrated Machine Learning (QML) algorithm search (Figure A.1).

Rather than performing Quantum Architecture Search (QAS) over fixed gate sets of Parametrized Quantum Circuits (PQC; Wu et al., 2023; Du et al., 2022; Nakaji et al., 2024), searching in the program space allows agents to construct more expressive model architectures. Compared with single LLM optimizers for neural architecture search (Zhang et al., 2023; Yang et al., 2024), our approach extends agents with capabilities for tool use, memory via retrieval-augmented generation (RAG; Yan et al., 2024), and various optimization techniques (Dong et al., 2024; Rafailov et al., 2023). We construct the LLMMA for QML algorithm search such that given a classical deep learning algorithm, like the multi-layer perceptron (MLP) discussed subsequently, the system can automate the workflow to find its optimized quantum counterpart.

The operation can be abstracted as the following formulation.

$$\mathcal{A} = \sum_{ij} \alpha_{ij} \mathcal{L}_j$$

$$\mathcal{A}\left(\mathcal{C}_{ML}\right) \to \arg\min_{\pi} \mathcal{Q}_{ML}^{\pi}$$

The LLMMA ($\mathcal{A}$) is composed by several large language models ($\mathcal{L}_j$; distinct agents, or lobes (Li et al., 2024) where α_{ij} specifies the interaction among the lobes). Given a classical machine learning algorithm ($\mathcal{C}_{ML}$), $\mathcal{A}$ can facilitate to find the corresponding quantum machine learning algorithm ($\mathcal{Q}_{ML}$) with optimized policy (π). To receive steering signals from recursively control agents (Wong, 2025), it reforms as $\mathcal{A} = \sum_{ij} \alpha_{ij} \mathcal{L}_j + \beta_{ij} \mathbf{u}_j$, where $\mathbf{u}$ serves as an interface and β_{ij} as connection coefficients.

2 Can LLM-based Agent find quantum machine learning algorithm from its classical counterpart?

To demonstrate the LLM-based agent's capability, we begin with a classical algorithm and task the sub-agentic system (simple coder) to find a quantum counterpart. Given the name of the classical algorithm, the coder agent is instructed to generate an implementation the classical algorithm as initial condition, then conducts evolutionary loops of search and optimization to find its quantum counterpart (Figure A.2). The results (model performance) from simple coder also serve as a baseline compared with the full functioning multi-agentic framework described at a later part of the paper.

2.1 Multi-Layer Perceptron

We use the Multi-Layer Perceptron (MLP) as our classical seed algorithm. An MLP is a feedforward artificial neural network composed of an input layer, one or more hidden layers, and an output layer. The hidden layers use non-linear activation functions, which allows the network to learn complex, non-linear mappings and grants them universal approximation capabilities (Cybenko, 1989; Hornik, 1991). The agentic system was provided with a Python implementation of an MLP as input.

The agentic system then generated a quantum-computing-integrated version of the MLP. Figure A.3 shows a snippet of the resulting quantum integrated model and its training dynamics, confirming that the model learns effectively. The discovered quantum circuit, which uses 4 qubits, is composed of an initial rotation layer, structured controlled-Y rotations, and an alternating entanglement structure.

3 Test time scaling with Monte Carlo Tree Search (MCTS) and System Components

Searching the vast program space for novel algorithms requires an efficient strategy. To this end, we incorporate our coder agent into a Monte Carlo Tree Search (MCTS) framework (Metropolis &

Ulam, 1949). MCTS is well-suited for complex sequential decision-making problems. It iteratively builds a search tree by balancing exploration and exploitation (e.g., using the UCT algorithm), expanding promising nodes, simulating outcomes via rollouts, and backing up the estimated values through the tree (Kocsis & Szepesvári, 2006; Browne et al., 2012).

In our full grown LLMMA system, MCTS guides the overall search process. Given an initial classical algorithm, the agentic system first generates a plan, augmented with information from web searches (Figure A.5). It then executes the search as a series of evolutionary loops within the MCTS scheme.

Nested Dual Loops. To manage the coding-intensive nature of algorithm search, we designed a nested dual-loop procedure for code generation and refinement (Figure A.4). The inner loop iteratively handles fine-grained tasks like code chunking, debugging, and review until a satisfactory snippet is produced. This snippet is then passed to the outer loop, which coordinates system-wide components, simulates the code, reflects on the results, and updates the global plan. This design enables efficient context management and focused task execution.

Webscraper. To ensure our agentic system has access to the latest information, we developed a web scraping tool. Since the pre-trained knowledge of LLMs can become outdated, this tool supplements the planning phase by searching for and extracting relevant, up-to-date information from the internet based on given keywords. The extracted information is summarized (Figure A.6) and integrated into the agent's decision-making process.

4 RESULTS

We evaluated the performance of several QML algorithms discovered by our agentic system, with the results summarized in Table 1. A model generated by a simple coder sub-agent serves as the baseline. The models found by our full grown agentic system are categorized as "Intersect," "Fusion-concatenate," "Quantum-inspired," and "Quantum-inspired and superimposed." The accuracy scores validate that the generated programs execute correctly and achieve high performance. Notably, the "Quantum-inspired" model achieved the highest accuracy at 98%, a significant improvement over the baseline. The dataset employed is Pen-Based Recognition of Handwritten Digits (Alpaydin & Alimoglu, 1996). At the time when we ran the full grown agentic experiment, the base model was claude-4-sonnet-20250514.

Table 1: Performance of Discovered QML Models

Model	Accuracy
Simple coder (baseline)	9.40%
Intersect	56%
Fusion-concatenate	97.22%
Quantum-inspired	**98**%
Quantum-inspired and superimposed	97.78%

Intersection. This sample implements a quantum variational circuit between a classical preprocessing neural network and a classical post-processing neural network. The preprocessing neural network consists of two linear layers with a ReLU activation function in between. The first layer maps the input dimension to 16, and the second layer maps 16 to 4, which matches the number of qubits. 4 qubits are used in the quantum circuit. It includes a quantum feature map that applies RY and RZ rotations based on the input data, followed by a chain of CNOT gates. The variational circuit has two layers, each applying parameterized RY and RZ rotations followed by the same CNOT chain. The circuit returns expectation values of the Pauli Z operator for all 4 qubits. The post-processing neural network also consists of two linear layers with a ReLU activation function in between, mapping from 4 to 16 and then from 16 to the output dimension (Figure A.7).

Fusion-concatenate. This sample built an algorithm of fusion style concatenating quantum and classical features. 8--qubit variational quantum neural network is constructed. Inputs are amplitude-

embedded and the circuit outputs eight Pauli-Z expectation values. RX/RY/RZ rotations are applied on each qubit per layer. CNOT entanglement is circular connected in each layer and extra pairwise CNOTs on odd layers. A post-network with normalization maps to the target output dimension. The outputs from a parallel run of classical branch (linear + ReLU + dropout) are concatenate, then applying output head with dropout to produce the final classification label (Figure A.8).

Quantum-inspired. This sample constructed a quantum-inspired neural architecture with two parts: a custom layer mimicking quantum rotations/entanglement and a full network that fuses a quantum-inspired branch with a classical MLP. The quantum-inspired part applies per-feature trigonometric rotation transforms, adds a small "interference" term through a learned input-to-input matrix, projects features with an "entanglement" weight matrix, and normalization, handling batches through matrix multiplications. The quantum-inspired part then fuses with the classical part. The fused features are then applying an output layer to generate the final classification label (Figure A.9).

Quantum-inspired and superimposed. This model builds upon the quantum-inspired approach. It begins with a classical MLP-like pre-processing network. The output is then fed into quantum-inspired layers that simulate rotation and entanglement, similar to the previous model. A residual connection from the pre-processing network is then superimposed (added) to the quantum-inspired features. These combined features are passed to an output head for the final classification (Figure A.10).

5 Conclusion

We have introduced an LLM-based multi-agent framework that automates the synthesis of quantum-computing-integrated machine learning algorithms from classical starting points. Our approach leverages the principles of meta-learning, using foundation models to directly search the program space. The system integrates MCTS-guided exploration, a nested dual-loop for robust code generation/debugging, and web scraping for up-to-date knowledge. Starting with a classical MLP, our framework automatically discovered a diverse family of quantum-hybrid and quantum-inspired models. The empirical results are promising: fusion-style and quantum-inspired models achieved high accuracy (up to 98%), significantly outperforming a baseline code generator. This highlights the benefits of our approach, which combines agent orchestration, tool use, and guided search. Notably, the system not only reproduced known hybrid patterns but also discovered novel quantum-inspired layers and feature-fusion strategies, demonstrating the potential of LLM agents as practical assistants for accelerating QML design.

We highlight two key advantages of our methodology. First, searching in program space offers a flexible substrate that naturally spans classical, quantum, hybrid, and quantum-inspired designs, exceeding the expressivity of fixed gate-set architecture searches. Second, the use of MCTS and the nested dual-loop procedure substantially improves the stability and success rate for code-intensive generation tasks.

This study has several limitations. Our experiments were restricted to small-qubit circuits, and our reward metric focused primarily on accuracy, without considering multi-objective trade-offs such as circuit depth or parameter count. The search cost and reproducibility are sensitive to LLM stochasticity and tool reliability. Furthermore, we did not perform a detailed analysis of the robustness, generalization, or the security implications of the web-retrieval mechanism.

Future work will focus on several key areas. We plan to guide the system to utilize more quantum resources and incorporate multi-objective optimization, potentially including process reward models for more efficient MCTS. Since the MLP is a fundamental building block, we will expand the set of initial algorithms to include more complex architectures like Transformers, diffusion models, or self-supervised learning frameworks, which we expect will lead to the discovery of more novel algorithms. Other promising directions include implementing modules for QML interpretability and agentic self-improvement.

By coupling the reasoning and coding capabilities of LLM agents with principled search and verification, our approach provides a practical path toward automated QML discovery. We anticipate that scaling this framework will accelerate the co-design of classical and quantum algorithms, helping to surface novel patterns that are difficult to discover manually.

REFERENCES

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large Language Models for Mathematical Reasoning: Progresses and Challenges, April 2024.

Microsoft Research AI4Science and Microsoft Azure Quantum. The Impact of Large Language Models on Scientific Discovery : A Preliminary Study using GPT-4, December 2023.

E. Alpaydin and Fevzi. Alimoglu. Pen-Based Recognition of Handwritten Digits. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5MG6K.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012. ISSN 1943-0698. doi: 10.1109/TCIAIG.2012.2186810.

John F. Clauser, Michael A. Horne, Abner Shimony, and Richard A. Holt. Proposed Experiment to Test Local Hidden-Variable Theories. *Physical Review Letters*, 23(15):880–884, October 1969. doi: 10.1103/PhysRevLett.23.880.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989. ISSN 1435-568X. doi: 10.1007/BF02551274.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A Survey on In-context Learning, June 2024.

Yuxuan Du, Tao Huang, Shan You, Min-Hsiu Hsieh, and Dacheng Tao. Quantum circuit architecture search for variational quantum algorithms. *npj Quantum Information*, 8(1):1–8, May 2022. ISSN 2056-6387. doi: 10.1038/s41534-022-00570-y.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel Programming for Hyperparameter Optimization and Meta-Learning, July 2018.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large Language Model based Multi-Agents: A Survey of Progress and Challenges, April 2024.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jü rgen Schmidhuber. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework, November 2023.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4 (2):251–257, January 1991. ISSN 0893-6080. doi: 10.1016/0893-6080(91)90009-T.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-Learning in Neural Networks: A Survey, November 2020.

Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (eds.), *Machine Learning: ECML 2006*, pp. 282–293, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-46056-5. doi: 10.1007/11871842_29.

Yangyang Li, Junjie Xiao, Yanqiao Chen, and Licheng Jiao. Evolving deep convolutional neural networks by quantum behaved particle swarm optimization with binary encoding for image classification. *Neurocomputing*, 362:156–165, October 2019. ISSN 0925-2312. doi: 10.1016/j.neucom.2019.07.026.

YaoChong Li, Ri-Gui Zhou, RuQing Xu, Jia Luo, and WenWen Hu. A quantum deep convolutional neural network for image recognition. *Quantum Science and Technology*, 5(4):044003, July 2020. ISSN 2058-9565. doi: 10.1088/2058-9565/ab9f93.

Yuxiao Li, Eric J. Michaud, David D. Baek, Joshua Engels, Xiaoqing Sun, and Max Tegmark. The Geometry of Concepts: Sparse Autoencoder Feature Structure, October 2024.

Zhao Mandi, Shreeya Jain, and Shuran Song. RoCo: Dialectic Multi-Robot Collaboration with Large Language Models, July 2023.

Nicholas Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247):335–341, 1949. ISSN 0162-1459. doi: 10.2307/2280232.

Kouhei Nakaji, Lasse Bjørn Kristensen, Jorge A. Campos-Gonzalez-Angulo, Mohammad Ghazi Vakili, Haozhe Huang, Mohsen Bagherimehrab, Christoph Gorgulla, FuTe Wong, Alex McCaskey, Jin-Sung Kim, Thien Nguyen, Pooja Rao, and Alan Aspuru-Guzik. The generative quantum eigensolver (GQE) and its application for ground state search, January 2024.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, June 2025.

Luca Parisi, Daniel Neagu, Renfei Ma, and Felician Campean. Quantum ReLU activation for Convolutional Neural Networks to improve diagnosis of Parkinson's disease and COVID-19. *Expert Systems with Applications*, 187:115892, January 2022. ISSN 0957-4174. doi: 10.1016/j.eswa.2021.115892.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative Agents for Software Development, June 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model, December 2023.

Sita Rani, Piyush Kumar Pareek, Jaskiran Kaur, Meetali Chauhan, and Pankaj Bhambri. Quantum Machine Learning in Healthcare: Developments and Challenges. In *2023 IEEE International Conference on Integrated Circuits and Communication Systems (ICICACS)*, pp. 1–7, February 2023. doi: 10.1109/ICICACS57338.2023.10100075.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, January 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6.

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. Rethinking the Bounds of LLM Reasoning: Are Multi-Agent Discussions the Key?, February 2024.

Lin Wei, Haowen Liu, Jing Xu, Lei Shi, Zheng Shan, Bo Zhao, and Yufei Gao. Quantum machine learning in medical image analysis: A survey. *Neurocomputing*, 525:42–53, March 2023. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.01.049.

Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A Survey on Neural Architecture Search, June 2019.

Fu-Te Wong. *SQUEEZE ARTIFICIAL CONSCIOUSNESS FROM LLMMA*. July 2025. doi: 10.13140/RG.2.2.31501.45287.

Wenjie Wu, Ge Yan, Xudong Lu, Kaisen Pan, and Junchi Yan. QuantumDARTS: Differentiable Quantum Architecture Search for Variational Quantum Algorithms. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 37745–37764. PMLR, July 2023.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective Retrieval Augmented Generation, February 2024.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers, April 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models, December 2023.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building Cooperative Embodied Agents Modularly with Large Language Models, February 2024.

Michael R. Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using Large Language Models for Hyperparameter Optimization, December 2023.

A Appendix

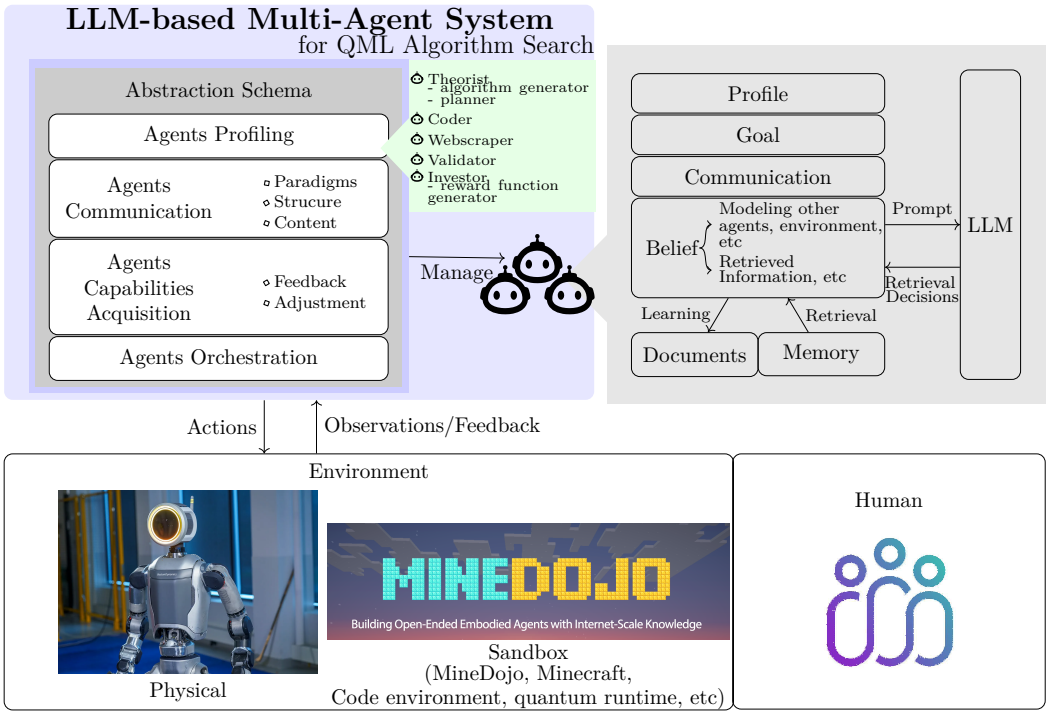


Figure A.1: The architecture of the LLM-based Multi-Agent system for QML search

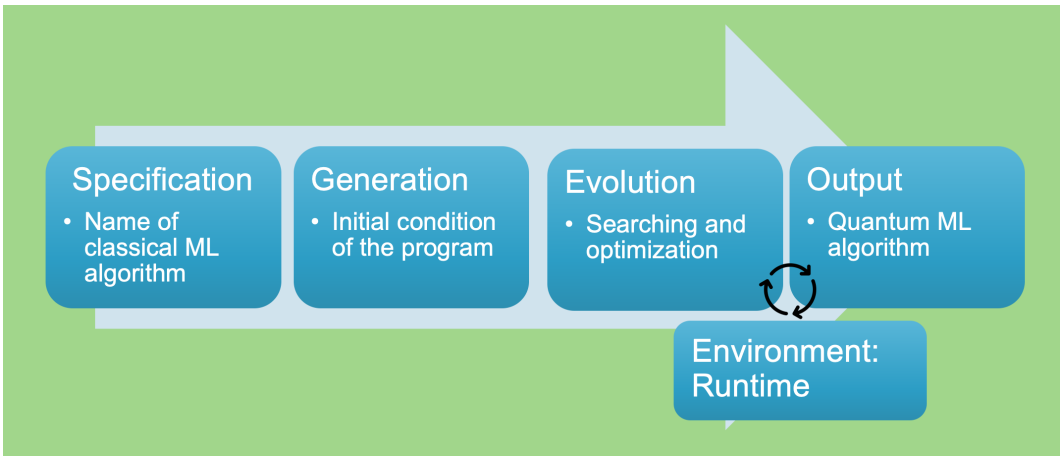
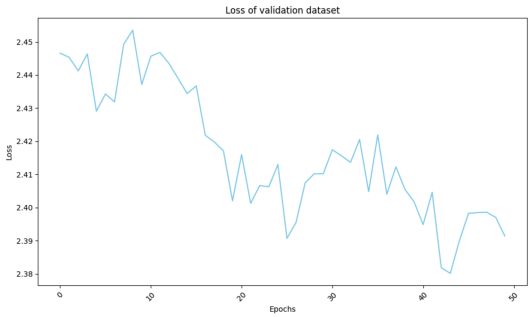


Figure A.2: Flow chart of the QML searching process.

```python
def quantum_forward(params, input_data):
    """Execute quantum circuit with given parameters and input data."""
    backend = Aer.get_backend('qasm_simulator')

    # Create circuit
    qc, circuit_params = create_quantum_circuit(n_qubits, n_layers)

    # Assign parameters
    parameter_dict = dict(zip(circuit_params, params))
    bound_circuit = qc.assign_parameters(parameter_dict)

    # Encode input
    bound_circuit = encode_input(input_data.flatten()[:n_qubits], bound_circuit, n_qubits)

    # Execute
    job = backend.run(bound_circuit, shots=n_shots)
    result = job.result()
    counts = result.get_counts()

    # Calculate expectation value
    expectation = 0
    total_shots = sum(counts.values())

    for bitstring, count in counts.items():
        # Convert bitstring to +1/-1 value
        value = (-1)**(bitstring.count('1') % 2)
        expectation += value * count / total_shots

    return expectation

def cost_function(params, input_data, target):
    """Calculate cost function for optimization with L2 regularization."""
    prediction = quantum_forward(params, input_data)
```

(a) A snippet of the Python implementation of the discovered Quantum MLP.



(b) The learning dynamics of the Quantum-integrated MLP, showing decreasing loss over training steps.

Figure A.3: The discovered Quantum-integrated MLP and its learning dynamics.

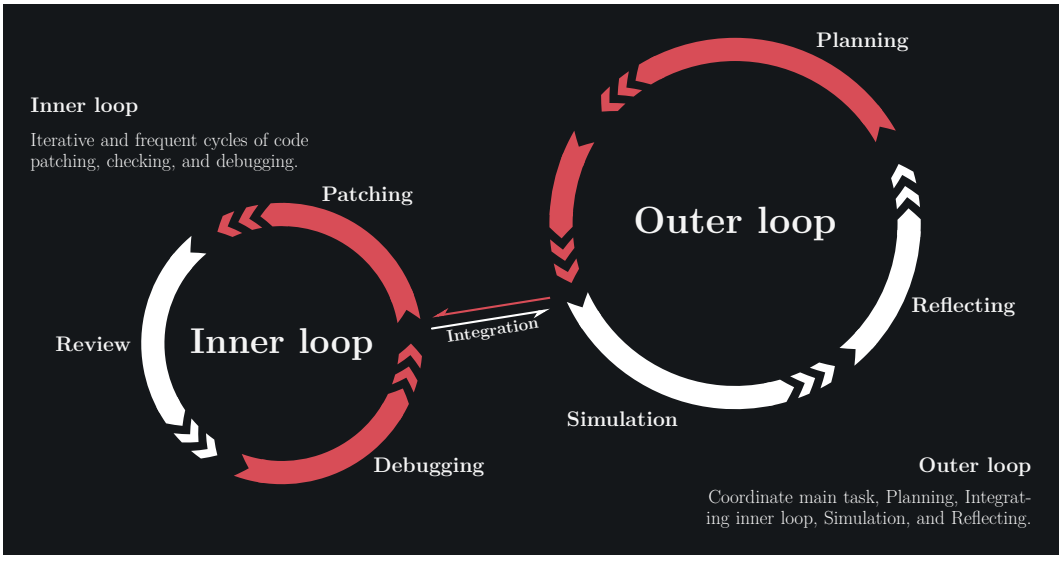


Figure A.4: The nested dual-loop procedure for robust code implementation and improvement.

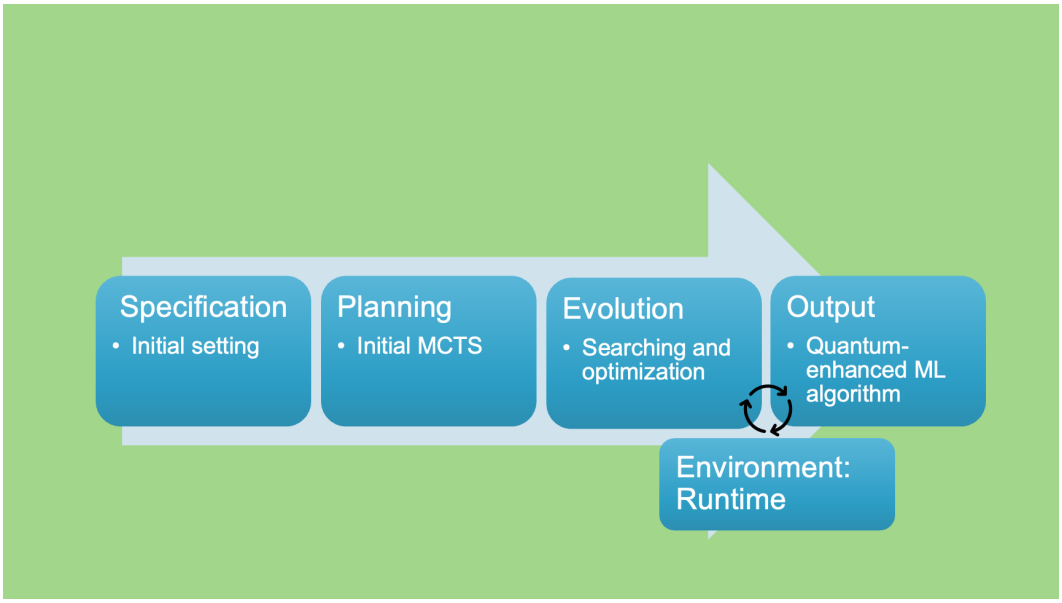


Figure A.5: The flowchart of the MCTS-guided search process in our LLM-based Multi-Agent system.

⓪Abstract

The quantum multilayer perceptron (QMLP) represents a revolutionary approach to neural network architectures that leverages quantum mechanical principles to enhance computational capabilities. This emerging field combines the structural foundation of classical multilayer perceptrons with quantum computing paradigms, including superposition, entanglement, and quantum interference. This comprehensive report examines the theoretical foundations, architectural implementations, current applications, and future prospects of quantum multilayer perceptrons in the context of quantum machine learning.

①1. Introduction

The intersection of quantum computing and artificial intelligence has given rise to quantum machine learning (QML), a field that promises to revolutionize computational approaches to pattern recognition, optimization, and data processing. Within this domain, the quantum multilayer perceptron stands as one of the most promising architectures, offering potential quantum advantages over classical neural networks.

Traditional multilayer perceptrons (MLPs) have been fundamental building blocks in classical machine learning, serving as universal function approximators capable of learning complex non-linear relationships. The quantum analog extends these capabilities by exploiting quantum mechanical phenomena, potentially providing exponential speedups for specific problem classes and enabling novel computational paradigms.

The development of quantum multilayer perceptrons addresses several key challenges in both quantum computing and machine learning: the efficient representation of high-dimensional data, the implementation of non-linear transformations in quantum systems, and the practical application of quantum algorithms to real-world problems. This report provides a comprehensive analysis of these aspects while examining the current state of research and future directions in this rapidly evolving field.

②2. Theoretical Foundations

③2.1 Quantum Computing Principles

Quantum multilayer perceptrons are built upon fundamental quantum mechanical principles that distinguish them from their classical counterparts. The primary quantum phenomena utilized include:

Superposition: Unlike classical bits that exist in definite states of 0 or 1, quantum bits (qubits) can exist in superposition states, representing both 0 and 1 simultaneously with complex probability amplitudes. This property allows quantum systems to process exponentially more information than classical systems with the same number of basic units.

Entanglement: Quantum particles can become entangled, creating correlations that persist regardless of spatial separation. In quantum neural networks, entanglement enables complex correlations between different layers and neurons that are impossible to achieve classically.

Figure A.6: An example summary produced by the Webscraper tool from online sources.

```python
class QuantumNeuralNetwork(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.input_dim = input_dim
        self.output_dim = output_dim

        # Classical pre-processing layers
        self.pre_net = nn.Sequential(nn.Linear(input_dim, 16), nn.ReLU(), nn.Linear(16, n_qubits))

        # Quantum layer weights
        self.q_weights = nn.Parameter(torch.randn(2, n_qubits, 2) * 0.1)

        # Classical post-processing
        self.post_net = nn.Sequential(nn.Linear(n_qubits, 16), nn.ReLU(), nn.Linear(16, output_dim))

    def forward(self, x):
        # Classical preprocessing
        x_processed = self.pre_net(x)

        # Quantum processing
        batch_size = x.shape[0]
        quantum_outputs = []

        for i in range(batch_size):
            # Convert to numpy for PennyLane (move to CPU first)
            input_data = x_processed[i].detach().cpu().numpy()
            weights = self.q_weights.detach().cpu().numpy()

            # Run quantum circuit
            q_out = quantum_circuit(input_data, weights)
            quantum_outputs.append(q_out)

        # Convert back to tensor and move to device
        quantum_tensor = torch.tensor(quantum_outputs, dtype=torch.float32, requires_grad=True, device=x.device)

        # Classical post-processing
        output = self.post_net(quantum_tensor)
        return output
```

Figure A.7: A code snippet illustrating the ”Intersection” model architecture.

```python
def forward(self, x):
    # Variational quantum branch
    quantum_x = x
    for i, layer in enumerate(self.quantum_layers):
        quantum_x = layer(quantum_x)
        quantum_x = self.activation(quantum_x)
        if i < len(self.quantum_layers) - 1:  # No dropout on last quantum layer
            quantum_x = self.dropout(quantum_x)

    # Classical branch for hybrid enhancement
    classical_x = self.classical_branch(x)

    # Fuse quantum and classical features
    fused_features = torch.cat([quantum_x, classical_x], dim=1)
    fused_output = self.fusion_layer(fused_features)
    fused_output = self.activation(fused_output)

    # Final classification
    output = self.output_layer(fused_output)
    return output
```

Figure A.8: A code snippet illustrating the ”Fusion-concatenate” model architecture.

```python
def forward(self, x):

    # Apply quantum-inspired rotations (element-wise)
    cos_theta = torch.cos(self.theta)
    sin_theta = torch.sin(self.theta)
    cos_phi = torch.cos(self.phi)
    sin_phi = torch.sin(self.phi)
    cos_psi = torch.cos(self.psi)
    sin_psi = torch.sin(self.psi)

    # Multi-rotation transformation (simulating multiple quantum gates)
    x_rot1 = x * cos_theta - x * sin_theta * cos_phi
    x_rot2 = x * sin_theta * sin_phi + x * cos_psi
    x_rotated = x_rot1 + x_rot2 * sin_psi

    # Interference pattern (quantum superposition-like)
    x_interference = torch.matmul(x_rotated, self.interference_weights)
    x_combined = x_rotated + 0.1 * x_interference  # Small interference effect

    # Entanglement-like operation with proper batch handling
    x_entangled = torch.matmul(x_combined, self.entangle_weights)

    # Apply layer normalization
    output = self.layer_norm(x_entangled)

    return output
```

Figure A.9: A code snippet illustrating the ”Quantum-inspired” model architecture.

```python
def forward(self, x):
    # Classical preprocessing
    classical_features = self.classical_pre(x)

    # Quantum-inspired processing
    quantum_features = self.quantum_layer1(classical_features)
    quantum_features = self.quantum_layer2(quantum_features)

    # Residual connection (hybrid approach)
    # Combine classical and quantum features
    if classical_features.size(1) != quantum_features.size(1):
        classical_adapted = torch.mean(
            classical_features.view(classical_features.size(0), -1, quantum_features.size(1)), dim=1
        )
    else:
        classical_adapted = classical_features

    hybrid_features = self.residual_weight * quantum_features + (1 - self.residual_weight) * classical_adapted

    # Classical post-processing for classification
    output = self.classical_post(hybrid_features)

    return output
```

Figure A.10: A code snippet illustrating the ”Quantum-inspired and superimposed” model.