

# Automating quantum feature map design via large language models

Kenya Sakka<sup>1</sup> Kosuke Mitarai<sup>1 2</sup> Keisuke Fujii<sup>1 2 3</sup>

## Abstract

Quantum feature maps are a key component of quantum machine learning, encoding classical data into quantum states to exploit the expressive power of high-dimensional Hilbert spaces. Despite their theoretical promise, designing quantum feature maps that offer practical advantages over classical methods remains an open challenge. In this work, we propose an agentic system that autonomously generates, evaluates, and refines quantum feature maps using large language models. The system consists of five components: Generation, Storage, Validation, Evaluation, and Review. Using these components, it iteratively improves quantum feature maps. Experiments on the MNIST dataset show that it can successfully discover and refine feature maps without human intervention. The best feature map generated outperforms existing quantum baselines and achieves competitive accuracy compared to classical kernels across MNIST, Fashion-MNIST, and CIFAR-10. Our approach provides a framework for exploring dataset-adaptive quantum features and highlights the potential of LLM-driven automation in quantum algorithm design.

## 1. Introduction

Quantum machine learning (QML) has gained attention in recent years due to its potential advantages over classical machine learning (Biamonte et al., 2017; Cerezo et al., 2022). By leveraging the principles of quantum mechanics, QML aims to enhance computational efficiency and improve learning performance across various tasks.

A central concept in quantum machine learning is that of

quantum features (Havlíček et al., 2019; Schuld & Killoran, 2019; Mitarai et al., 2018; Cerezo et al., 2022). This notion can be seen as a quantum counterpart of classical features in machine learning, where quantum states—represented as density operators on Hilbert spaces whose dimension grows exponentially with the number of qubits—are used as feature representations. By encoding classical data into such quantum states and leveraging the structure of these exponentially large Hilbert spaces, quantum features aim to provide enhanced expressive power for learning algorithms. Liu *et al.* have shown that, in a certain artificial task, quantum features can indeed offer rigorous advantages over classical counterpart (Liu et al., 2021).

Despite the theoretical promise of quantum features, designing quantum feature maps that provide practical advantages in real-world machine learning tasks remains an open challenge. In particular, for widely used datasets like MNIST and other benchmark tasks in classical machine learning, no quantum feature map has yet been found that consistently outperforms classical approaches or demonstrates a clear quantum advantage. This limitation has been highlighted in a recent study by Huang et al. (Huang et al., 2021), where various empirical quantum feature maps have been tested across standard benchmarks. The results so far suggest that quantum advantages are hard to observe under realistic conditions, pointing to a gap between theoretical potential and practical applicability.

One fundamental reason for this difficulty is that, in both classical and quantum settings, effective feature maps are often highly dependent on the structure of the dataset and the nature of the task. For instance, features suitable for image data may differ substantially from those effective for time-series or tabular data. Therefore, the practical realization of useful quantum features must consider dataset-specific design, ideally allowing for automated adaptation to the data at hand. Developing methods to generate such dataset-adaptive quantum features in a principled and scalable manner is an important and urgent challenge. Addressing this challenge is critical for advancing QML from a theoretical concept to a practical tool in modern machine learning.

Research on applying large language models (LLMs) and other AI techniques to quantum circuit design remains scarce, but existing work has begun to reveal the potential of

<sup>1</sup>Center for Quantum Information and Quantum Biology, Osaka University, 1-2 Machikaneyama, Toyonaka 560-0043, Japan  
<sup>2</sup>Graduate School of Engineering Science, Osaka University 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan  
<sup>3</sup>RIKEN Center for Quantum Computing, Wako Saitama 351-0198, Japan.  
 Correspondence to: Kenya Sakka <sakka.kenya.qiqb@osaka-u.ac.jp>, Kosuke Mitarai <mitarai.kosuke.es@osaka-u.ac.jp>, Keisuke Fujii <fujii.keisuke.es@osaka-u.ac.jp>.

automation in quantum algorithm development. For example, Nakaji (Nakaji et al., 2024) demonstrated an approach tailored to ground-state search problems, highlighting the feasibility of circuit generation by training task-specific models and validating the results through numerical experiments. In contrast, Ueda (Ueda & Matsuo, 2024) proposed a framework that leverages LLMs to select appropriate ansatz and incorporate experimental results as feedback for the LLMs. However, their work focuses primarily on conceptual design and does not include implementation results or quantitative evaluation. Nevertheless, these methods rely on fixed circuit templates, task-specific formulations that require additional model training or fine-tuning, and static internal knowledge, which can limit their flexibility and scalability across diverse tasks and rapidly evolving quantum software environments. In particular, key ingredients for realizing automated science for quantum computing such as the integration of code generation, empirical validation, and iterative improvement remains an open challenge.

In this work, we propose a prompt-based system to achieve autonomous improvement of quantum feature map design, which is an important but challenging task in quantum machine learning, without reliance on internal knowledge. Our system consists of five components—Generation, Storage, Validation, Evaluation, and Review—which together form a feedback loop designed to generate quantum circuits from scratch. As a prompt-based system, it does not require any additional model training or fine-tuning, making it easily adaptable to various tasks. In each iteration, the LLM generates candidate feature map ideas based on dataset characteristics and hardware constraints, evaluates them in terms of originality, feasibility, and versatility, and refines them using the latest relevant academic papers retrieved from a vector database. The resulting executable quantum circuits are then validated and evaluated on benchmark datasets using a kernel-based classification model, with implementation support based on documentation extracted from the latest source code of quantum libraries. The evaluation results are analyzed to identify successful design elements and areas for improvement, which in turn guide the next round of generation. This iterative loop enables the autonomous evolution of quantum feature maps through empirical feedback. An overview of the system developed in this work is shown in Fig. 1.

To the best of our knowledge, this is the first work in the quantum domain to automate the entire research workflow, ranging from idea generation to implementation, evaluation, and iterative refinement, using a LLM. Given the rapidly evolving nature of quantum computing, where new discoveries and major updates of software libraries are constantly emerging, it is difficult for an LLM to generate impactful ideas and executable programs based solely on its internal knowledge. Our system addresses this challenge by incor-

porating up-to-date external information and empirical feedback into a structured, iterative process. We demonstrate the effectiveness of this approach on widely used benchmark datasets, where the automatically generated quantum feature maps outperform existing ones widely adopted. In quantum machine learning, one of the key goals is to surpass the performance of classical machine learning models. In line with this objective, our system not only exceeds the accuracy of commonly used quantum feature maps but also outperforms a variety of classical kernels including linear and polynomial kernels, and achieves comparable performance to classical kernels known for their broad applicability and high accuracy. By integrating LLM-driven creativity with quantum program execution and performance-based refinement, our work highlights a new methodology for bridging the gap between theoretical quantum learning models and their practical deployment.

## 2. Preliminary

This section outlines the fundamental concepts necessary for the subsequent discussion. We begin by introducing the concept of quantum feature maps, which encode classical data into quantum states. We then describe the quantum kernel method, where similarities between quantum states are used for learning tasks. Following this, we provide a brief overview of LLMs, focusing on their generative capabilities and broad applicability. Finally, we discuss recent developments in automated science, highlighting the role of LLMs in automating various stages of the scientific discovery process, including applications in quantum computing.

### 2.1. Quantum Feature Map

Feature mapping is a transformation  $\phi : \mathcal{X} \rightarrow \mathcal{F}$  that maps data from the original input space  $\mathcal{X}$  to a higher-dimensional feature space  $\mathcal{F}$ , facilitating the identification of nonlinear patterns. For instance, given an input  $d$ -dimensional vector  $\mathbf{x} \in \mathbb{R}^d$ , an appropriately designed nonlinear transformation  $\phi(\mathbf{x})$  can allow linear models, such as support vector machines (SVMs), to effectively handle complex data distributions. However, increasing the complexity of the nonlinear transformation generally requires a higher-dimensional feature space, which leads to the curse of dimensionality. Kernel methods address this challenge by enabling computations in the high-dimensional space without explicitly constructing it, using a kernel function that computes inner products between mapped data points. This concept forms the basis of kernel-based learning algorithms and naturally motivates their extension to quantum settings.

Quantum feature mapping extends this concept to quantum computing by employing a quantum circuit to encode classical data into quantum states (Havlíček et al., 2019; Schuld & Killoran, 2019; Mitarai et al., 2018; Cerezo et al., 2022).

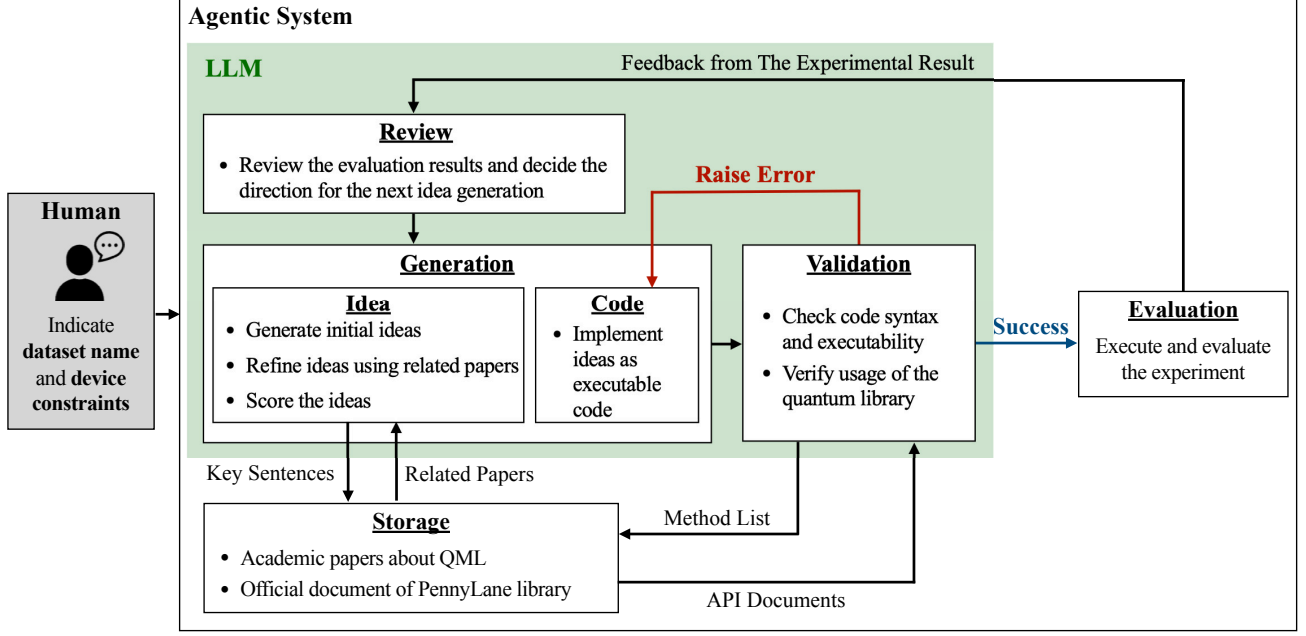


Figure 1. Overview of the agentic system for automatic generation of quantum feature maps. When a user provides task instructions to the system, five internal components work collaboratively to autonomously conduct experiments and improvements. As a result, the system generates an executable program that implements a quantum feature map capable of performing the task with high accuracy.

Given an input  $\mathbf{x}$ , the quantum feature map prepares the corresponding quantum state, or quantum feature, as

$$\rho(\mathbf{x}) = U(\mathbf{x})|0\rangle^{\otimes n}\langle 0|^{\otimes n}U(\mathbf{x})^\dagger, \quad (1)$$

where  $|0\rangle^{\otimes n}$  is the initial state of  $n$  qubits, and  $U(\mathbf{x})$  is a unitary transformation that depends on  $\mathbf{x}$ .

## 2.2. Quantum Kernel Method

The quantum kernel method applies classical kernel techniques to data embedded in a quantum Hilbert space via quantum feature maps. The quantum kernel function is defined as the Hilbert–Schmidt inner product between density operators:

$$k(\mathbf{x}, \mathbf{x}') = \text{Tr}[\rho(\mathbf{x})\rho(\mathbf{x}')] \quad (2)$$

This kernel function quantifies the similarity between data points in the quantum feature space and can be used with standard kernel-based learning algorithms such as support vector machines or kernel ridge regression (Havlíček et al., 2019; Schuld & Killoran, 2019; Cerezo et al., 2022).

An attractive property of the quantum kernel method is that it inherits the theoretical foundation of classical kernel methods, enabling one to find optimal solutions in the quantum feature space according to a well-defined loss function. In particular, learning remains convex in the feature space, allowing for efficient optimization and generalization analysis

(Havlíček et al., 2019; Schuld & Killoran, 2019; Cerezo et al., 2022). This is in contrast to other approaches that employ the quantum feature to construct models, such as quantum circuit learning framework (Mitarai et al., 2018) where we apply a trainable quantum circuit to  $\rho(\mathbf{x})$  to extract relevant information from the feature. These approaches gives lower prediction cost (Nakayama et al., 2024) and might be able to generalize more due to the restricted search space (Mitarai et al., 2018), the complex loss landscape challenges us to train the models (McClean et al., 2018). As such, researchers have often employed the quantum kernel methods to benchmark the quantum feature maps on real datasets (Huang et al., 2021; Haug et al., 2021).

## 2.3. Large Language Models

Large Language Models (LLMs) are a type of generative model and large-scale machine learning system trained on vast amounts of text data from the internet and other sources (Anthropic, 2024; DeepMind, 2023; Llama Team, 2024; OpenAI, 2023). Given an input sequence of tokens  $x_{1:t} = (x_1, x_2, \dots, x_t)$ , LLMs generate text by modeling the conditional probability distribution:

$$P(x_{t+1} | x_{1:t}) = \frac{\exp(s(x_{1:t}, x_{t+1}))}{\sum_{x' \in V} \exp(s(x_{1:t}, x'))} \quad (3)$$

where  $s(x_{1:t}, x')$  represents the unnormalized logit score assigned to token  $x'$ , and  $V$  denotes the vocabulary. By

sampling from this distribution, LLMs iteratively produce coherent and contextually relevant text.

Leveraging extensive training datasets and large-scale neural networks, such as the Transformer architecture (Vaswani et al., 2017), LLMs can perform not only text generation but also a wide range of tasks, including code generation (Li et al., 2022) and multi-modal processing involving images, videos, and audio (Yin et al., 2024). Furthermore, in recent years, models referred to as Reasoning Models (Huang & Chang, 2023; DeepSeek-AI et al., 2025), which are designed for deep and structured thinking, have been increasingly utilized in tasks requiring complex task.

## 2.4. Automated Science

The use of AI for scientific discovery has primarily been explored in domains that can be simulated, such as machine learning. In particular, the recent significant advancements in LLMs have greatly expanded their applicability, as computers can now understand instructions provided in natural language. For example, there have been efforts to use LLMs for generating objective functions in machine learning models (Lu et al., 2024a). In that research, LLMs were employed to automate the generation of objective functions and their subsequent refinement based on evaluation results. Additionally, research has focused on model merging algorithms, proposing an iterative improvement process driven by LLMs to develop algorithms autonomously, without relying on human expertise or predefined ideas (Ishibashi et al., 2024). In addition to work on automating specific algorithm development, there has also been research on the automation of scientific discovery in collaboration with humans (Wu et al., 2024), aiming to support and enhance human-led research processes. Beyond this, other studies have explored the complete automation of the scientific research process, encompassing idea generation, experimental execution, and academic paper writing (Lu et al., 2024b; Yamada et al., 2025).

Regarding the use of AI in the field of quantum computing, generative models based on Transformer architectures have been developed to generate quantum circuits (Nakaji et al., 2024). However, these models are primarily trained for the ground state search problem, and extending them to tasks in different domains requires the design of new objective functions and re-training, which can be both technically challenging and computationally expensive. Moreover, they cannot readily incorporate up-to-date external knowledge, and there is no established method for encoding classical data, limiting their applicability. Also, Ueda and Matsuo have proposed to use LLMs for optimizing quantum circuits in quantum generative adversarial networks (Ueda & Matsuo, 2024). In their approach, the LLM selects from a set of ansatz candidates predefined by the user, rather than gen-

erating code directly. The method depends on the model’s internal knowledge, and its practicality remains unclear, as no numerical experiments or implementation results are provided—it is presented primarily as a conceptual idea.

The role of AI in scientific automation is expanding, evolving from a mere assistive tool to an entity that actively participates in the fundamental process of scientific discovery. However, to truly realize this potential, new frameworks are needed to enable AI to engage efficiently in the pursuit of novel scientific insights. This includes not only supporting individual tasks but also empowering AI to operate across the entire research cycle—generating idea, designing and executing experiments, analyzing results, and iteratively refining its own processes—to achieve autonomous scientific advancement.

## 3. LLM-based system for automatic generation of quantum feature maps

Our agentic system for automatic generation of quantum feature maps consists of five components: “Generation”, “Storage”, “Validation”, “Evaluation” and “Review” (Figure 1). A single trial comprises processing by these five components, and by iteratively repeating this process while incorporating feedback from the results, the system improves the classification accuracy progressively. In this section, we describe the architecture of our system in detail.

### 3.1. Generation

In the “Generation” component, the system uses LLMs to propose, score, refine, and implement candidate ideas for quantum feature maps. This process begins with the generation of multiple candidate ideas, followed by a scoring phase to evaluate their potential. Based on the scores and other contextual signals, the system then performs a reflection step to refine ideas. These refined ideas are re-scored, and finally, the ideas are implemented as Python programs. Each prompt includes information such as the dataset name, the kernel function, the machine learning model, the input data format, hardware constraints of the quantum device, and other directives.

#### 3.1.1. IDEA GENERATION

The first process of the “Generation” component is generation of candidate ideas. Here, the system prompts an LLM to generate ideas for quantum feature maps. For the first trial, we designed a prompt that encourages the generation of broadly applicable, general ideas to create a foundation for future performance improvements. In the subsequent trials, the system prompt LLM to refine the design of the ansatz based on the review provided from the previous Review step. Throughout the trials, we prompt LLM to generate multiple



ideas simultaneously to encourage diversity. Notably, we prohibit the use of nonlinear transformation or trainable parameters within the quantum feature maps, since otherwise the system might output feature maps that depend on classical models, such as neural networks, to achieve high accuracy. The LLM structures each output into four components: an overview, a detailed explanation, corresponding mathematical expressions in TeX format, and a set of key sentences summarizing the idea. As examples, we provide the prompts used for this idea generation process in the experiments of this work (see Sec. 4) in Listing 1, 2 and 3 in Appendix A.

### 3.1.2. SCORING

The second process of the “Generation” is scoring of the generated ideas. The system prompts an LLM to score the overall direction of each idea based on three criteria: *Originality*, which assesses the novelty of the idea; *Feasibility*, which examines whether the idea can be implemented as a program; and *Versatility*, which evaluates broad applicability of the idea rather than being overly specialized for a particular task or dataset. For this step, we integrate a vector database storing relevant information within the system, which is described in detail in Sec. 3.2. It is notable that we assign the scores to the ideas solely to guide the LLM’s reasoning and do not explicitly use them in later stages to, e.g., select which direction to pursue. We eventually pass these scores to the user prompt of reflection for further refinement (Listing 10).

The detailed flow of the scoring process is as follows. The system first uses the key sentences generated during the idea generation phase as search queries to retrieve relevant academic papers from the database. The LLM generate additional search queries up to a fixed number of attempts if it judges the initial information to be insufficient. Note that naively inputting academic papers into the LLM can easily exceed its context window. To avoid this, we use a lightweight model to generate concise summaries that fit within a predefined word limit, focusing on key elements such as methodology, results, and areas for improvement. We provide the prompt used in the summarization process of this work in Listing 11 and 12, in Appendix A. The technique of retrieving necessary external information and incorporating it into the prompt of a generative model is known as Retrieval-Augmented Generation (RAG) (Lewis et al., 2020). After the retrieval process is completed, the LLM scores the generated idea on a scale of 0 to 10 using the retrieved information. To establish a scoring baseline, we also provide human-annotated scores of existing quantum feature maps to the LLM as few-shot examples (Brown et al., 2020). We use Listing 5 in Appendix A in our experiments presented in 4. To reduce the variations of scores across trials, the scoring results are incorporated into the prompt

for the next trial. As examples, we provide the prompts used for this idea scoring process in the experiments of this work (see Sec. 4) in Listing 4, 6, 7 and 8, in Appendix A.

### 3.1.3. REFLECTION

The third process is to let an LLM to reflect on the generated ideas (Madaan et al., 2023). The system does so using external information, which are retrieved from the database using key sentences incorporated with the ideas as search queries. Concretely, the system first retrieve the summaries of the academic papers in the same manner during the scoring process, and then prompts an LLM to reflect on the ideas using them. This process of searching for relevant papers and reflecting on the idea for further improvements continues until the LLM judges the reflection to be complete or the predefined maximum number of iterations is reached. As examples, we provide the prompts used for this reflection process in the experiments of this work (see Sec. 4) in Listing 9 and 10, in Appendix A.

### 3.1.4. IMPLEMENTATION

Upon completion of idea reflection and scoring, the system finally uses an LLM to generate a Python program which implements quantum feature maps. In this process, the LLM is given the names of available quantum gates and library functions, as well as the implementation template presented in Listing 15. As examples, we provide the prompts used for this implementation process in the experiments of this work (see Sec. 4) in Listing 13 and 14, in Appendix A.

## 3.2. Storage

The “Storage” component of the system is a vector database consisting of relevant papers and documentations of the program libraries used by the LLMs. The former is included to compensate for the knowledge cutoff of the LLMs, particularly in light of the rapid recent developments in quantum computing and quantum machine learning. The latter addresses the need for up-to-date documentation, given the frequent updates in quantum computing software libraries. This component stores these data as a vector database (VectorDB) (Johnson et al., 2017; Wang et al., 2021). A VectorDB accepts natural language sentences converted into vector representations as search queries, enabling context-aware retrieval that is more effective than traditional keyword-based search (Karpukhin et al., 2020).

## 3.3. Validation

The “Validation” component of the system iteratively refine the code until it became executable by leveraging syntax analysis and external documentation. It consists of three static validations and one dynamic validation.

As static validation, the system first checks whether the generated program can be compiled as a Python script using the `py_compile` library. Next, it uses the `ast` module to verify that the code complies with Python’s syntax rules. If both checks pass, the system extracts PennyLane method names from the code and queries the documentation database using exact keyword matches. The code and corresponding documentation are then provided as input to the LLM, which verifies the correctness of method usage, including function calls, argument names, and argument types.

After three static validations are passed, the system performs dynamic validation using dummy data to ensure that the code runs without errors. As examples, we provide the prompts used for this validation process in the experiments of this work (see Sec. 4) in Listing 16 and 17, in Appendix A.

If an error occurs at any stage of the validation process, the source code and error messages are fed back to the LLM, which attempts to correct and regenerate the source code. This validation process continues until all validations pass successfully or the maximum number of retries is exceeded. As examples, we provide the prompts used for this idea scoring process in the experiments of this work (see Sec. 4) in Listing 18 and 19, in Appendix A.

### 3.4. Evaluation

The “Evaluation” component assesses the performance of the quantum machine learning model using validated quantum feature maps. It follows a structured procedure based on standard machine learning practices.

The system first splits the dataset into training, validation, and test subsets. It transforms the training samples using the generated quantum feature map and computes pairwise kernel values to construct a kernel matrix. Using this matrix, it trains a support vector machine (SVM). The trained SVM is then used to compute accuracy, precision, recall, and F-measure on the validation and test sets. Note that the choice of the quantum-feature-based model is arbitrary; while we use an SVM in this work for demonstration purposes due to its ease of training once the kernel matrix is computed, other models such as quantum circuit learning (Mitarai et al., 2018) can also be employed. After the evaluation, the system feeds back the results to “Review” component which guide idea generation in the next iteration.

### 3.5. Review

The “Review” component evaluates the quantum feature map ideas generated in the previous trial based on the feedback information provided by the “Evaluation” component. This component uses an LLM to analyze the evaluation results of the most recent trial, listing multiple key factors

that contributed to success and areas requiring improvement. The prompt includes five key items: a text that guides the direction of the review, a textual description of the quantum feature map, its mathematical expression, the model’s training time, hardware information, and the performance metrics on the validation set, which includes an additional formatted string (e.g.,  $\text{idea}_1 > \text{idea}_2$ ) to indicate the ranking of generated ideas based on accuracy. The description and mathematical expression are the ones generated during the “Generation” step. The results of this review are used as supplementary information for the “Generation” component in the next trial.

Three remarks are in order. First, we include the training time in order to make the system to avoid quantum feature maps that are overly complex or rely on computationally expensive methods, such as amplitude encoding (Schuld et al., 2015). While such designs are not inherently undesirable from the perspective of achieving high accuracy, they can lead to problematically long experimental (in our experiments presented in Sec. 4, simulation) times. We encourage the system to be aware of computational costs from this concern.

Second, we include the hardware information in the prompt to prevent the system from deviating from the intended focus on quantum feature maps. In some cases, the generated ideas shifted toward broader challenges in quantum computing, such as noise reduction, while our main intension is to construct an effective quantum feature maps. For example, without this information, the “Review” component occasionally produced suggestions to reduce the number of gates or to introduce redundant procedures for mitigating noise. While such ideas are relevant in general, they fall outside the scope of evaluating quantum feature maps. By explicitly providing information about the target quantum device (in our experiments in Sec. 4, it is an simulator) in the prompt, we reinforce that the evaluation should remain focused on the quantum feature map itself.

Third, we choose the text for directing the review with the following process. The system first calculates the accuracy difference between the two most recent trials as  $\text{DiffMetric}$ . More specifically,  $\text{DiffMetric}$  at the  $t$ -th trial is calculated as  $\text{DiffMetric}(t) = A(t) - A(t - 1)$ , where  $A(t)$  denotes the validation accuracy at the  $t$ -th trial. Based on the  $\text{DiffMetric}$  value, it chooses five types of direction:

- $0.2 < \text{DiffMetric} \leq 1.0$ : The previous trial’s idea led to a significant improvement. The text tells the LLM to focus on identifying the successful aspects to ensure that the accuracy continues to improve.
- $0.0 < \text{DiffMetric} \leq 0.2$ : The previous trial’s idea led to a moderate improvement. The text tells the LLM to identify the positive aspects while also exploring poten-

tial improvements for further accuracy enhancement.

- $\text{DiffMetric} = 0.0$ : The previous trial’s idea resulted in no change. The text tells the LLM to identify bottlenecks or constraints in the idea and explore potential modifications.
- $-0.2 \leq \text{DiffMetric} < 0.0$ : The previous trial’s idea led to a decline in accuracy. The text refers to all past trials and tells the LLM to determine the cause of the accuracy degradation and aim to restore the accuracy level.
- $-1.0 \leq \text{DiffMetric} < -0.2$ : The previous trial’s idea led to a significant decline in accuracy. The text tells the LLM to conduct thorough analysis of all past trials, identify the root cause of accuracy deterioration, and consider major modifications to restore accuracy.

We provide the prompts used for the Review component in the experiments of this work (see Sec. 4) in Listing 20 and 21, in Appendix A.

## 4. Experiments

We evaluate the effectiveness of our LLM-based agentic system for generating quantum feature maps through a series of experiments on standard image classification datasets. We first run the system on the MNIST handwritten digits dataset (LeCun et al., 1998) to generate quantum feature maps. Then, we compare the generated quantum feature maps with classical and quantum baselines across other image datasets, specifically Fashion-MNIST (Xiao et al., 2017) and CIFAR-10 (Krizhevsky, 2009) to assess their generalizability. The detailed experimental conditions are described in the following sections. Hyperparameters within the system, such as the number of ideas generated in each trial, can be found in Appendix C.

### 4.1. Dataset and preprocessing

The MNIST dataset (LeCun et al., 1998) consists of 70,000 handwritten digit images, each of which is a  $32 \times 32$  grayscale image labeled with one of the digits from 0 to 9. MNIST is officially divided into 60,000 training images and 10,000 test images. Our agentic system uses a sampled subset of the training data and do not use any of the test set. When it finishes the iterative improvements of the feature maps, we evaluate the feature maps generated from the system on the entire test dataset. We also use the Fashion-MNIST (Xiao et al., 2017) and CIFAR-10 (Krizhevsky, 2009) datasets to evaluate the generalizability of the generated quantum feature map in this final evaluation process. Fashion-MNIST comprises 70,000 images related to fashion, each of which is a  $32 \times 32$  grayscale image categorized into one of ten classes, such as clothing and footwear.

CIFAR-10 is a dataset consisting of 60,000 color images, where each image is a  $32 \times 32$  RGB image. The dataset includes ten classification labels, such as automobiles, birds, and other object categories.

We apply principal component analysis (PCA) to reduce the dimensionality of the image data. This is a standard approach when constructing quantum feature map for relatively high-dimensional inputs such as images (Huang et al., 2021). In this work, we use the top 80 principal components as input features and normalize the data to the range of 0.0 to 1.0. This preprocessing is fixed, that is, the system never sees the original data at any point of the iterative process or allowed to modify this preprocessing.

To improve computational efficiency during the iterative quantum circuit generation process, we do not use the full official training set of 60,000 images within the MNIST dataset. Instead, we randomly sample 10,000 images while ensuring an equal distribution across all ten classes. We then split this sampled dataset into 6,000 images for training, 2,000 for validation, and 2,000 for testing.

### 4.2. Large language model

We use OpenAI’s LLMs for quantum feature map generation. Specifically, the system uses three models, “o3-mini-2025-01-31”, “gpt-4o-2024-11-20”, and “gpt-4o-mini-2024-07-18” depending on the specific tasks. Specifically, it uses the “o3-mini-2025-01-31” model, which excels in reasoning tasks, for review, idea-generation, idea-reflection, and code-generation. The *reasoning\_effort* parameter, which controls the depth of reasoning, of this model is set to its highest level, “high”. The parameter *temperature*, which controls the randomness of output, is not supported in the “o3-mini-2025-01-31” model, and thus the output exhibits a fixed level of randomness. The “gpt-4o-2024-11-20” model, a highly versatile general-purpose model, is utilized for scoring and validation tasks. The lightweight “gpt-4o-mini-2024-07-18” model is used for summary tasks. For the GPT-4o series, the temperature parameter is set to 0.0, the lowest value indicating the least randomness. We also perform experiments using other LLMs, whose results are discussed in Appendix F.

When storing data in the VectorDB, we use the “text-embedding-3-small” model provided by OpenAI as the embedding model. We segment texts into chunks of 1,024 tokens, and convert each chunk into a 1,536-dimensional vector representation before being stored in the database.

### 4.3. External knowledge

We use the arXiv API to retrieve PDF files of academic papers. We restrict the search criteria to the quant-ph category, which corresponds to quantum physics, and set the target

period from January 1, 2020, to December 31, 2024. We specify “Quantum Machine Learning” as the search keyword and find 998 papers. To store them in the database, we extract text from the retrieved PDFs and segment it into chunks of 1,024 tokens. We then convert these text segments into vector representations using an embedding model.

We use the source code of PennyLane version 0.39.0 as a reference for software documentation. Since PennyLane is open-source software, we can access both its source code and documentation publicly. However, the documentation is not always up-to-date and lacks sufficient detail for the purpose of this study. To address this, we create reference information for the LLM directly from the source code. We begin by extracting all classes related to quantum gates, along with their class names and the corresponding docstrings that describe their functionality and usage. We also add metadata that specifies how each class is invoked in a program. Finally, we segment the docstring texts into chunks of 1,024 tokens and convert them into vector representations using an embedding model before storing them in the database.

#### 4.4. Support vector machine settings

We evaluate the effectiveness of the generated quantum feature maps using an SVM as the downstream model. For each feature map, we compute the kernel values using the Hilbert–Schmidt inner product defined in Eq. 2, and construct the corresponding kernel matrix. We then train an SVM classifier using `scikit-learn` (Pedregosa et al., 2011) based on this matrix. We choose the SVM for its efficiency and reproducibility in kernel-based learning tasks.

We fix the SVM hyperparameters as follows: we set the regularization parameter  $C$  to 1.0, and set the kernel coefficient  $\gamma$  to ‘scale’, which automatically adjusts based on the variance of the input features. We perform all kernel evaluations using a noiseless simulator provided within PennyLane (Bergholm et al., 2018). We fix the number of qubits to 10 throughout the experiments.

## 5. Results

We first confirmed that our system can successfully generate executable quantum feature maps using an LLM. In all trials, the system generated Python code that passed the validation process without error, and we observed no execution failures after validation.

### 5.1. Behavior of the improvements

Figure 2, which shows two example trajectories, corresponding to the ones with high-performance and low-performance initial ideas, of the best validation/test accuracy across trials, demonstrates that the system progressively optimizes the

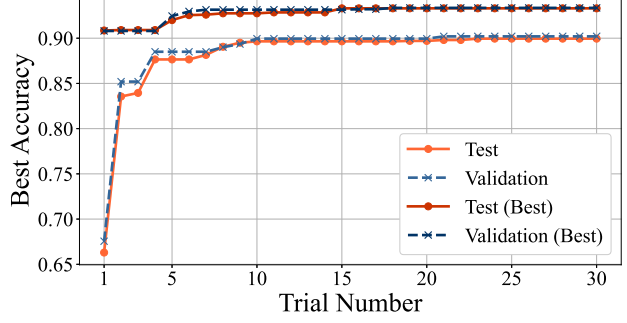


Figure 2. Trajectory of classification accuracy on the MNIST dataset using quantum feature maps generated by our agentic system. The curves shown in dark colors (red and blue) represent trials with high-performance initial ideas. The curves shown in lighter colors (orange and light blue) represent an example where low-performance initial ideas. The vertical axis of the figure represents the best validation accuracy up to that trial, defined as  $\max(\text{accuracy}(t), \text{accuracy}(t-1), \dots)$ , where  $\text{accuracy}(t)$  denotes the best validation accuracy in the  $t$ -th trial. The horizontal axis corresponds to the trial number.

quantum feature maps through iterative refinement. The validation/test accuracy here is the one computed with the 2000 images passed to the agent for the validation and test, and not with the official test set of MNIST. We can also examine the feature maps generated throughout the trials to observe in what manner the system improves feature maps.

Let us first discuss the case where the system generated a high-performance initial idea. In this particular case, the system initially generated a simple quantum circuit that first applies eight single-qubit rotations to each qubit, where each input dimension is embedded as the angle of a single-qubit rotation, and then applies  $\prod_{i=0}^9 \text{CNOT}_{i,i+1}$ , where  $\text{CNOT}_{i,j}$  denotes a controlled-NOT gate with control qubit  $i$  and target qubit  $j$ . This circuit, despite effectively being a quantum feature map consisting of single-qubit rotations only as the final CNOT gates cancels when taking the kernel values by Eq. (2), achieves over 90% validation/test accuracy. Note that this is not a surprising result; (Huang et al., 2021) has already shown that this type of entanglement-free feature map can achieve high-accuracy on a Fashion-MNIST dataset. Subsequent trials from this initial idea introduced several refinements. These include varying the type of rotation gates by layer index, adjusting scale parameters for rotation angles, and incorporating global features by aggregating all 80 input dimensions. A certain trial also added more advanced logic that independently tune scale parameters for each layer. The embedding method also evolved from assigning a single data dimension to each gate to combining multiple data dimensions for a single rotation angle. Ultimately, the system arrived at a complex circuit including



many two-qubit gates and complicated, however linear, embedding of the input features to angles, which we show as Listing 15 in Appendix D. This feature map has the best performance over the whole in this work, achieving over 95% in accuracy.

In contract, for cases where the system generated a low-performance initial idea, the performance tends to saturate at lower levels, as we will see later in Fig. 3. An example of such trajectory is shown in Figure 2 in a lighter color. In this particular case, the two initial ideas were (1) a simple circuit that resembles the circuit generated in the high-performance gate but with significantly lower performance due to the different order of assigning feature values to angles, and (2) a complex circuit that involves controlled rotations with angles determined by multiple feature values but shows higher performance than (1). We observe that the system pursued improvements in the idea (2) since it performed better than (1). However, due to the complexity of the initial idea, the system fails to improve the feature maps to the level obtained in the high-performance case.

Finally, we show all trajectories of accuracy improvement across 45 experiments in Figure 3. The same data but plotted against trial number can be found in Fig. 5 of Appendix E. The figure clearly shows the dependence of the obtained accuracies on the initial idea; the lower initial accuracy tends to result in lower accuracies in subsequent trials. This result suggests that, if human professionals can provide good initial ideas, the system might be able to obtain feature maps that exceed the best result obtained in this work. We leave developing such a system as an interesting future direction to explore.

## 5.2. Generalizability of the best feature map

Next, we evaluate the generalizability of the generated quantum feature maps on Fashion-MNIST and CIFAR-10 datasets. The purpose of this evaluation is two-fold. First, we wish to assure that the system has not designed a feature map that only works on MNIST dataset, which is used in the iterative improvement process. Knowledge about the MNIST dataset that is almost surely included in detail within LLMs can potentially lead them to design such a feature map. Second objective is to compare the performance of the generated quantum feature maps among the ones that are widely used as baselines. For classical feature maps to compare against, we select four kernels commonly used in SVMs: the linear kernel, the Radial basis function (RBF) kernel, the polynomial kernel, and the sigmoid kernel. For quantum feature maps, we selected the so-called ZZ feature map (Havlíček et al., 2019), the so-called NPQC feature map and YZCX feature map (Haug et al., 2021).

Each method that we compare the generated feature map against involves multiple hyperparameters. Therefore, we

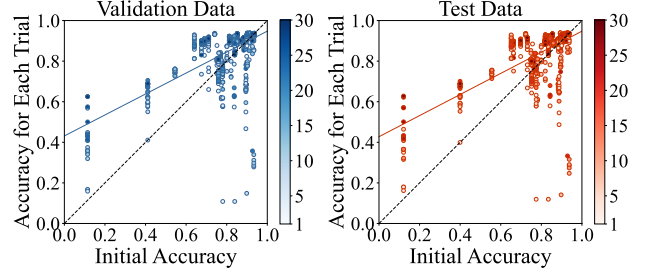


Figure 3. Trajectory of classification accuracy over the course of all 45 experiments. The values on the Y-axis represent the accuracy obtained at each trial, rather than the best accuracy. Therefore, the highest value along the Y-axis corresponds to the final best accuracy. The color bars on the right side show the trial index. The number of trials is represented by the intensity of the plotted points, with darker colors indicating later trials. The regression line represented by the solid line in the validation data is  $y = 0.5164x + 0.4315$ , and in the test data, it is  $y = 0.5200x + 0.4276$ . The black dotted line indicates the baseline; points above this line represent trials that exceeded the initial accuracy, while points below represent trials that fell short of it.

employed Optuna (Akiba et al., 2019), a hyperparameter optimization framework, to search for optimal parameter configurations based on the sampled training dataset, followed by evaluation on the corresponding test dataset. Since the optimization process involves inherent randomness, we conducted five independent optimization runs for each dataset.

We report the results of each method obtained in this manner in Table 1. It shows the mean and standard deviation of classification accuracy obtained using the models configured with the best parameters from each run of the hyperparameter tuning. From Table 1, we can confirm that the generated quantum feature map achieved consistent classification accuracy across different datasets, indicating that they did not overfit to the MNIST dataset. For all datasets, the generated feature map outperformed other widely used quantum feature maps in terms of classification accuracy. On the other hand, when compared to classical machine learning approaches, the generated maps outperformed the linear, polynomial and sigmoid kernels. However, they slightly underperformed compared to the RBF Kernel, which is the most commonly used and effective kernel in classical settings.

## 6. Conclusion

In this study, we proposed an agentic system that autonomously performs iterative refinement by using a LLM to generate ideas for quantum feature maps along with executable code, followed by reviewing the evaluation results. Although the generated quantum feature map does not

Type	Method	Accuracy		
		MNIST (10,000 sample)	Fashion-MNIST (10,000 sample)	CIFAR-10 (10,000 sample)
Classical	Linear Kernel	0.9385 $\pm$ 0.0002	0.8437 $\pm$ 0.0009	0.4087 $\pm$ 0.0011
	Polynomial Kernel	0.9667 $\pm$ 0.0058	0.8702 $\pm$ 0.0030	0.5375 $\pm$ 0.0014
	Sigmoid Kernel	0.9343 $\pm$ 0.0002	0.8189 $\pm$ 0.0120	0.4079 $\pm$ 0.0006
	RBF Kernel	<b>0.9765 <math>\pm</math> 0.0005</b>	<b>0.8864 <math>\pm</math> 0.0014</b>	<b>0.5669 <math>\pm</math> 0.0085</b>
Quantum	ZZ FeatureMap	0.9255 $\pm$ 0.0009	0.8252 $\pm$ 0.0023	0.3907 $\pm$ 0.0016
	NPQC FeatureMap	0.9644 $\pm$ 0.0028	0.8749 $\pm$ 0.0026	0.4903 $\pm$ 0.0188
	YZCX FeatureMap	0.9727 $\pm$ 0.0006	0.8778 $\pm$ 0.0049	0.4753 $\pm$ 0.0341
	Generated Feature Map (Best)	<b>0.9731 <math>\pm</math> 0.0008</b>	<b>0.8835 <math>\pm</math> 0.0021</b>	<b>0.5290 <math>\pm</math> 0.0030</b>

Table 1. Generated Quantum Feature Map Performance on Different Datasets

outperform classical machine learning models, it achieves higher accuracy than several existing quantum feature maps across multiple datasets.

In future work, extending the system to support quantum machine learning models with trainable parameters—such as quantum circuit learning—and improving the logic of the iterative refinement process to enhance exploration efficiency may lead to the discovery of quantum feature maps that outperform classical machine learning models.

Beyond quantum feature map design, our agentic framework can potentially be extended to a broad class of variational quantum algorithms such as the variational quantum eigensolver (VQE) (Peruzzo et al., 2014) and the quantum approximate optimization algorithm (QAOA) (Farhi et al., 2014). By enabling automatic circuit generation and refinement through empirical feedback, the system may help to discover more efficient or interpretable ansatz structures tailored to specific quantum tasks. Furthermore, adapting the system to generate and optimize circuits for quantum error correction, or to assist in the construction of novel quantum algorithms, presents a promising direction for future research at the intersection of quantum computing and autonomous scientific discovery.

## Acknowledgements

This work is supported by MITOU Target Program, organized by Information-technology Promotion Agency (IPA). This work is also supported by MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) Grant Nos. JPMXS0120319794 and JPMXS0118067394, and JST COI-NEXT Grant No. JPMJPF2014. K.M. is supported by JST FOREST Grant No. JPMJFR232Z and JSPS KAKENHI Grant No. 23H03819.

## References

- Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD '19)*, pp. 2623–2631, 2019. doi: 10.1145/3292500.3330701. URL <https://doi.org/10.1145/3292500.3330701>.
- Anthropic. Claude 3 model card, 2024. Available at <https://assets.anthropic.com/m/61e7d27f8c8f5919/original/Claude-3-Model-Card.pdf>.
- Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Ahmed, S., others, and Killoran, N. PennyLane: Automatic differentiation of hybrid quantum-classical computations. *arXiv*, 1811.04968, 2018. URL <https://arxiv.org/abs/1811.04968>. Version 4, last revised 29 Jul 2022.
- Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., and Lloyd, S. Quantum machine learning. *Nature*, 549:195–202, 2017. doi: 10.1038/nature23474. URL <https://www.nature.com/articles/nature23474>.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.*, 33:1877–1901, 2020. URL <https://arxiv.org/abs/2005.14165>.
- Cerezo, M., Verdon, G., Huang, H.-Y., Cincio, L., and Coles, P. J. Challenges and opportunities in quantum machine learning. *Nat. Comput. Sci.*, 2: 567–576, 2022. doi: 10.1038/s43588-022-00311-3. URL <https://www.nature.com/articles/s43588-022-00311-3>.
- DeepMind, G. Gemini: A family of highly capable multi-modal models. *arXiv*, 2312.11805, 2023. URL <https://arxiv.org/abs/2312.11805>.

- DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., et al. Deepseek-rl: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv*, 2501.12948, 2025. URL <https://arxiv.org/abs/2501.12948>.
- Farhi, E., Goldstone, J., and Gutmann, S. A quantum approximate optimization algorithm. *arXiv:1411.4028 [quant-ph]*, 2014. URL <https://arxiv.org/abs/1411.4028>.
- Haug, T., Self, C. N., and Kim, M. S. Quantum machine learning of large datasets using randomized measurements. *arXiv*, 2108.01039, 2021. URL <https://arxiv.org/abs/2108.01039>.
- Havlíček, V., Córcoles, A. D., Temme, K., Harrow, A. W., Kandala, A., Chow, J. M., and Gambetta, J. M. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019. doi: 10.1038/s41586-019-0980-2.
- Huang, H.-Y., Broughton, M., Mohseni, M., Babush, R., Boixo, S., Neven, H., and McClean, J. R. Power of data in quantum machine learning. *Nature Communications*, 12(1):2631, 2021. doi: 10.1038/s41467-021-22539-9. URL <https://www.nature.com/articles/s41467-021-22539-9>.
- Huang, J. and Chang, K. C.-C. Towards reasoning in large language models: A survey. *Findings of the Association for Computational Linguistics: ACL 2023*, 2023: 1049–1065, 2023. doi: 10.18653/v1/2023.findings-acl.67. URL <https://aclanthology.org/2023.findings-acl.67/>.
- Ishibashi, Y., Yano, T., and Oyamada, M. Can large language models invent algorithms to improve themselves? *arXiv*, 2410.15639, 2024. URL <https://arxiv.org/abs/2410.15639>.
- Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017. URL <https://arxiv.org/abs/1702.08734>.
- Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and tau Yih, W. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.550. URL <https://aclanthology.org/2020.emnlp-main.550/>.
- Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, Univ. of Toronto, 2009. URL <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>.
- LeCun, Y., Cortes, C., and Burges, C. J. C. MNIST handwritten digit database. ATT Labs [Online], 1998. URL <http://yann.lecun.com/exdb/mnist>.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL <https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf>.
- Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL <https://www.science.org/doi/10.1126/science.abq1158>.
- Liu, Y., Arunachalam, S., and Temme, K. A rigorous and robust quantum speed-up in supervised machine learning. *Nat. Phys.*, 17:1013–1017, 2021. doi: 10.1038/s41567-021-01287-z. URL <https://www.nature.com/articles/s41567-021-01287-z>.
- Llama Team, A. . M. The llama 3 herd of models. *arXiv*, 2407.21783, 2024. URL <https://arxiv.org/abs/2407.21783>.
- Lu, C., Holt, S., Fanconi, C., Chan, A. J., Foerster, J., van der Schaar, M., and Lange, R. T. Discovering preference optimization algorithms with and for large language models. *arXiv*, 2406.08414, 2024a. URL <https://arxiv.org/abs/2406.08414>.
- Lu, C., Lu, C., Lange, R. T., Foerster, J., Clune, J., and Ha, D. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv*, 2408.06292, 2024b. URL <https://arxiv.org/abs/2408.06292>.
- Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegrefe, S., Alon, U., Dziri, N., Prabhunoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL <https://arxiv.org/abs/2303.17651>.
- McClean, J. R., Boixo, S., Smelyanskiy, V. N., Babush, R., and Neven, H. Barren plateaus in quantum neural network training landscapes. *Nature*

- Communications*, 9(1):4812, 2018. doi: 10.1038/s41467-018-07090-4. URL <https://www.nature.com/articles/s41467-018-07090-4>.
- Mitarai, K., Negoro, M., Kitagawa, M., and Fujii, K. Quantum circuit learning. *Phys. Rev. A*, 98:032309, 2018. doi: 10.1103/PhysRevA.98.032309. URL <https://doi.org/10.1103/PhysRevA.98.032309>.
- Nakaji, K., Kristensen, L. B., Campos-Gonzalez-Angulo, J. A., Vakili, M. G., Huang, H., Bagherimehrab, M., et al. The generative quantum eigensolver (gqe) and its application for ground state search. *arXiv*, 2401.09253, 2024. URL <https://arxiv.org/abs/2401.09253>.
- Nakayama, A., Morisaki, H., Mitarai, K., Ueda, H., and Fujii, K. Explicit quantum surrogates for quantum kernel models. *arXiv:2408.03000 [quant-ph]*, 2024. URL <https://arxiv.org/abs/2408.03000>.
- OpenAI. Gpt-4 technical report. *CoRR*, abs/2303.08774, 2023. URL <https://arxiv.org/abs/2303.08774>.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.-H., Zhou, X.-Q., Love, P. J., Aspuru-Guzik, A., and O’Brien, J. L. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:4213, 2014. doi: 10.1038/ncomms5213. URL <https://doi.org/10.1038/ncomms5213>.
- Schuld, M. and Killoran, N. Quantum machine learning in feature hilbert spaces. *Phys. Rev. Lett.*, 122:040504, 2019. doi: 10.1103/PhysRevLett.122.040504. URL <https://doi.org/10.1103/PhysRevLett.122.040504>.
- Schuld, M., Sinayskiy, I., and Petruccione, F. An introduction to quantum machine learning. *Contemp. Phys.*, 56(2):172–185, 2015. doi: 10.1080/00107514.2014.964942. URL <https://doi.org/10.1080/00107514.2014.964942>.
- Ueda, K. and Matsuo, A. Optimizing ansatz design in quantum generative adversarial networks using large language models. *arXiv preprint arXiv:2405.13196*, 2024. IBM Quantum, IBM Research - Tokyo, Tokyo, Japan.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL <https://arxiv.org/abs/1706.03762>.
- Wang, J., Yi, X., Guo, R., Jin, H., Xu, P., Li, S., Wang, X., Guo, X., Li, C., Xu, X., Yu, K., Yuan, Y., Zou, Y., Long, J., Cai, Y., Li, Z., Zhang, Z., Mo, Y., Gu, J., Jiang, R., Wei, Y., and Xie, C. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD ’21)*, pp. 2614–2627. Association for Computing Machinery, 2021. doi: 10.1145/3448016.3457550. URL <https://doi.org/10.1145/3448016.3457550>.
- Wu, Y., Shi, K., Burda, Y., Edwards, H., Kay, J., Zahavy, T., Such, F. P., Freeman, C. D., Goh, G., Hesse, C., Sutskever, I., Saunders, W., Chen, M., Tang, Q., Li, Y., Zhang, J., Kramár, J., Baker, B., Jain, S., Hernandez, G., Sun, Y., Yang, E. Z., Ha, D., Steinhardt, J., Brown, T. B., Amodei, D., Olah, C., Schmidt, L., and Clune, J. Towards an ai co-scientist. *arXiv*, 2502.18864, 2024. URL <https://arxiv.org/abs/2502.18864>.
- Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv*, 1708.07747, 2017. URL <https://arxiv.org/abs/1708.07747>.
- Yamada, Y., Lange, R. T., Lu, C., Hu, S., Forster, J., Clune, J., and Ha, D. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. Sakana AI Blog Post. Available online: <https://pub.sakana.ai/ai-scientist-v2/paper>, April 2025.
- Yin, S., Fu, C., Zhao, S., Li, K., and Sun, X. A survey on multimodal large language models. *Natl. Sci. Rev.*, 2024. doi: 10.1093/nsr/nwad123. URL <https://doi.org/10.1093/nsr/nwad123>.



## A. Prompts

In the developed agentic system, a total of eight types of prompts are used to instruct the LLM. These prompts are explained separately by component. At runtime, they may receive multiple placeholders (represented in red color text), the details of which are provided in the [Placeholder in Prompts](#) section. In the prompt, the terms “Trial” and “Round” both refer to repetitions, but they denote different levels of iteration: a “Trial” represents a complete cycle of the agentic system—including Generation, Validation, Evaluation, and Review—while a “Round” refers to repeated processes within specific components of a single Trial, such as Scouring or Reflection.

### A.1. Generation

In the [Generation component](#), five types of prompts are used: idea generation, idea scoring, idea reflection, summary generation, and code generation. The corresponding prompt files are shown in Listings 1–15.

- Idea generation: Listings 1–3
- Idea scoring: Listings 4–8
- Idea reflection: Listings 9–10
- Summary generation: Listings 11–12
- Code generation: Listings 13–15

*Listing 1. The developer prompt for the idea generation*

```

1 """
2 You are a quantum computing expert specializing in designing Quantum Feature Maps for
  classification tasks using a Quantum Support Vector Machine (QSVM).
3
4 Your task is to create a quantum feature map that will serve as the kernel function in a
  QSVM classifier applied to MNIST data. The ultimate goal is to design a feature map
  that enables the classifier to achieve high accuracy in classification.
5
6 This task will follow an iterative improvement process, where the feature map design is
  refined based on review comments provided after each iteration. Use the feedback to
  enhance the design while maintaining alignment with the defined objectives and
  constraints.
7
8 # Task Definition
9 Develop multiple ideas for quantum feature maps that satisfy the following criteria. The
  feature maps will be used to compute the quantum kernel  $K(x, x') = |\langle \Phi(x) | \Phi(x') \rangle|^2$ 
  for a QSVM. Ensure the designs are tailored to this kernel computation.
10 1. Design Considerations:
11   - Define combinations of quantum gates.
12   - Define an entanglement pattern for the features.
13   - Specify the method for embedding input data and quantum states as rotation angles
     of quantum gates.
14   - Ensure the 80-dimensional input data is utilized effectively, minimizing any loss
     of information.
15   - Avoid excessive feature compression that may lead to information loss (e.g.,
     simple feature averaging, summing, etc.).
16 2. Restrictions on Encoding and Embedding:
17   - Only linear functions are allowed for encoding and embedding.
18   - All parameters in the encoding and embedding must be non-trainable.
19
20 ## Key Context
21 - The input data, originally represented as 784-dimensional image data, has been
  compressed to 80 dimensions using PCA and each value normalized to the range [0.0,
  1.0].
22 - Propose a quantum feature map that is independent of the number of qubits in the
  quantum device.

```

```

23
24 ## Iterative Design
25 - You will refine this feature map over {max-trial-num} total trials.
26 - Each trial, you'll receive evaluation feedback to help evolve the design.
27 - In subsequent trials, the primary goal is to improve classification accuracy based on
    the feedback provided, while maintaining or improving the feature map's fidelity and
    computational feasibility.
28
29 ## Output Format
30 results: [idea_1, idea_2, ..., idea_n]
31 Each idea should be structured as follows:
32     - explanation: A detailed explanation of the proposed feature map. Include design
        rationale, expected outcomes, and quantum gates used.
33     - formula: A concise TeX-formatted mathematical representation of your idea.
34     - summary: A 100-300 word summary highlighting the core innovation.
35     - feature_map_name: A descriptive name for your feature map.
36     - key_sentences: Up to 5 key sentences, each 50-100 words, that describe the
        essential aspects of your design for subsequent vector-based searching.
37
38 ## Important Notes
39 - Clarity is paramount. Reiterate points if necessary to ensure understanding. There is
    no length restriction on the explanation-ensure all relevant details are provided.
40 - Evaluation is performed using an ideal quantum simulator without noise, so hardware
    noise does not need to be considered.
41 """

```

Listing 2. The user prompt for the idea generation (first trial)

```

1 """
2 Trial {current-trial}/{max-trial-num}.
3
4 This is the first trial of the quantum feature map design task.
5 For this initial trial:
6 - Focus on designing high-accuracy quantum feature maps while exploring diverse
    approaches to feature map design.
7 - Ensure the designs align with the following constraints:
8     1. The idea itself must be a feature map that is independent of the number of qubits
        in the quantum device, but evaluation will be conducted using an
        {device-n-qubit}-qubit simulator.
9     2. The input data consists of 80-dimensional PCA-reduced MNIST data, with each
        value normalized to the range [0.0, 1.0].
10    3. The encoding method is restricted to non-trainable parameters and linear
        functions.
11    4. Ensure effective utilization of the 80-dimensional input data while minimizing
        information loss.
12        - Avoid excessive feature compression that may lead to information loss (e.g.,
        simple feature averaging, summing, etc.).
13
14 ### Key Objective
15 Create {idea-num} high-accuracy quantum feature map ideas that explore diverse
    directions and can serve as strong foundations for refinement in future trials.
16 """

```

Listing 3. The user prompt for the idea generation (subsequent trials)

```

1 """
2 Trial {current-trial}/{max-trial-num}.
3
4 ### Feedback from the Previous Trial (Trial {previous-trial})
5 {review-comment}
6
7 ### Task for This Trial
8 Based on the feedback provided above, refine your quantum feature map ideas and generate
    a total of {idea-num} improved ideas. These ideas should aim to enhance the

```

```

classification accuracy.
9
10 ### Key Guidelines
11 - The feedback includes the following:
12   - **Keep Points:** Aspects of the previous design that should be retained.
13   - **Suggestions:** Areas for improvement or new directions to explore.
14
15 - Variety in Approaches:
16   - You are not required to address all feedback points in a single idea.
17   - Ensure that the multiple ideas generated based on the review incorporate different
18     directions of improvement, maintaining diversity.
19
20 ### Primary Objective
21 The primary goal in this trial is to boost classification accuracy through iterative
    refinements while ensuring the designs align with the constraints.
22 """

```

Listing 4. The developer prompt for the idea scoring

```

1 """
2 As a reviewer for a scientific journal, you are tasked with evaluating new scientific
3   ideas from multiple perspectives while adhering to specific evaluation criteria.
4   Your evaluation should be thorough, objective, and based on the guidelines provided
5   below.
6
7 # Evaluation Criteria
8 Each criterion is scored on a scale of 0.0 to 10.0, in increments of 0.1, where 0.0
9   represents the lowest score and 10.0 represents the highest score:
10 - Originality: Assess how the idea differs from existing research. Does it make a novel
11   contribution?
12 - Feasibility: Evaluate the practicality of implementing the idea.
13 - Versatility: Consider how broadly the idea can be applied.
14
15 # Steps for Evaluation
16 1. Understand the Idea:
17   - Carefully read and comprehend the proposed idea.
18   - Organize the information needed to make a well-informed evaluation.
19 2. Assess Information Sufficiency:
20   - First, confirm the round number provided by the user. If it is the final round,
21     skip Step 2 and proceed to Step 3.
22   - Otherwise, assess whether the "# Related Work" section provides sufficient
23     information to evaluate the idea.
24   - If the information is insufficient for scoring, <is_lack_information> tag set to
25     True, and list up to 5 necessary information key sentences as a comma-separated
26     list within <additional_key_sentences> tags. The search will be conducted by
27     embedded vector for academic paper; therefore, ensure the <
28     additional_key_sentences> are specific and relevant. Each sentence length should
29     be between 50 to 100 words.
30   - In this case, terminate the scoring process and set all scores to 0.0.
31 3. Provide Reasoning:
32   - If the information is sufficient, proceed to evaluate the idea based on the
33     specified criteria.
34   - Enclose the rationale behind the evaluation results of each indicator in <reason>
35     tags and explain it in text.
36 4. Assign Scores:
37   - Based on the evaluation results and their rationale, assign a score to each
38     indicator.
39 5. Terminating the Evaluation:
40   - Once all scores and reasoning have been assigned, the evaluation is complete. <
41     is_lack_information> tag set to False.
42
43 # Baseline
44 {few_shot_examples}
45 """

```

Listing 5. The few shots example for idea scoring

```

1 """
2 - ZZFeatureMap:
3   - explanation:
4     - The **ZZFeatureMap** is a quantum circuit that encodes classical data into
      quantum states through a combination of data-dependent rotations and
      entangling gates. First, the classical data is encoded using single-qubit
      rotation gates, such as  $R_Z$  or  $R_Y$ , which rotate each qubit by
      an angle proportional to the input data. Then, entangling gates,
      specifically  $ZZ$ -type interactions, are applied between pairs of qubits
      to introduce quantum correlations. These entangling gates are represented
      by  $e^{-i\theta Z_i Z_j}$ , where  $Z_i Z_j$  denotes the tensor
      product of Pauli- $Z$  operators on two qubits, and  $\theta$  is a
      tunable parameter. The circuit can be extended with multiple layers of
      encoding and entanglement to increase its expressivity, capturing more
      complex patterns in the data.
5   - scores:
6     - Originality: 5.0
7     - Feasibility: 9.0
8     - Versatility: 6.0
9
10 - YZCX:
11   - explanation:
12     - The **YZCX feature map** is particularly well-suited for handling high-
      dimensional data, such as images, due to its ability to efficiently encode
      complex features and relationships into quantum states. The use of  $R_Y$ 
      and  $R_Z$  rotation gates allows the map to transform multi-dimensional
      input data into quantum states, capturing intricate patterns through the
      variation of rotation angles. Following these rotations, the inclusion of
      controlled-X ( $C_X$ ) gates introduces entanglement, enabling the circuit
      to represent correlations between features, such as the dependencies
      between pixels in images. The parameterized nature of the  $R_Y$  and  $R_Z$ 
      gates, combined with the adaptable structure of the controlled-X
      operations, provides the flexibility to optimize the feature map for
      specific data types. This makes the YZCX feature map an effective tool for
      processing high-dimensional datasets, leveraging quantum resources to
      capture complex relationships and enhance learning in data-intensive tasks.
13
14   - scores:
15     - Originality: 7.0
16     - Feasibility: 8.0
17     - Versatility: 7.0
18
19 - Chunked Angle Embedding:
20   - explanation:
21     - This feature map encodes the 80-dimensional PCA features on individual qubits
      as rotational angles around the Y-axis, chunking the data into 10 groups of
      8 features each. Each of the 10 qubits receives a single base rotation  $R_Y(\alpha)$ 
      derived from the mean of those 8 features. Specifically, if the group
      assigned to qubit  $j$  contains values  $(x_1, \dots, x_8)$ , we define  $\alpha_j = a \times (x_1 + \dots + x_8)/8$ ,
      where  $a$  is a fixed scaling factor used to ensure angles remain within
       $[0, \pi]$ . This ensures that each qubit's rotation angle captures the averaged
      local structure of the features assigned to it, while not overfitting or
      requiring trainable gates. After the initial embedding, we add a layer of
      controlled-Z (CZ) gates arranged in a cyclic pattern among qubits 1, ..., 10,
      introducing entanglement based on these assigned angles. These CZ gates
      preserve single-qubit phase information while correlating qubits, making the
      final state sensitive to multi-qubit interactions. We expect this approach
      to facilitate capturing relevant patterns across different parts of the
      image (as compressed by PCA). By encoding the average magnitude of each
      chunk rather than each feature individually, the map remains relatively
      sparse, making it easier to simulate and interpret. The hope is to highlight
      medium-scale data correlations from the PCA transformation, while leaving
      space for additional classical or quantum post-processing. This design

```



```

    trades some fine-grained detail for a more stable and robust representation
    that could be well-suited for classification in quantum kernel-based methods
    .
22 - socres:
23     - Originality: 7.5
24     - Feasibility: 8.0
25     - Versatility: 6.5
26
27 - Multi Axis Repeated Encoding:
28     - explanation:
29         - This design relies on the idea of repeated angle embeddings along the X, Y,
            and Z axes to generate a richer quantum feature map without introducing
            trainable parameters. We first split the 80-dimensional PCA data into five
            sets of 16 features each. For each set, we map these features to a 10-qubit
            system by distributing them among qubits, ensuring each qubit gets a portion
            of the data. The embedding comprises three distinct layers: (1)  $R_x(\theta)$ 
            rotations for each qubit, where  $\theta$  is proportional to the assigned data slice
            ; (2)  $R^y(\phi)$  rotations, likewise determined by the same data slices; and (3)
             $R_z(\psi)$  rotations, ensuring we incorporate different axes for amplitude shifts
            . These layers are repeated twice in sequence without any trainable
            parameters, but with carefully selected scaling to keep angles within  $[0, 2\pi]$ 
            . Because each qubit's embedding is repeated, the final state includes
            multiple nonlinear transformations of the same data, which can better
            separate points in the Hilbert space. This multi-axis repetition is designed
            to amplify relevant distinctions in the data representation. While the
            circuit might appear more complex compared to single-layer encodings, its
            repeated structure ensures interpretability, as each repeated block injects
            additional nuance into the quantum state. The interleaving of X, Y, and Z
            rotations is especially potent at highlighting subtle variations in data
            because each axis naturally imparts a different influence on the qubit's
            Bloch sphere representation.
30     - socres:
31         - Originality: 7.5
32         - Feasibility: 8.0
33         - Versatility: 6.5
34
35 - Polynomial Interaction Embedding:
36     - explanation:
37         - This feature map aims to emulate polynomial kernel expansions in a quantum
            circuit, capturing second-order interactions among the 80 PCA features. We
            first partition the 80 features into 10 sets of 8 features. Each set is
            mapped onto two qubits, resulting in five pairs covering the entire 10-qubit
            register. For each pair  $(q^i, q_j)$ , we apply an encoding that simulates  $(x^{i^2},$ 
 $\sqrt{2} x^i x_j, x_j^2)$  relationships. Concretely, the circuit starts by applying  $R_x$ 
 $(\theta^i)$  on  $q^i$  and  $R_x(\theta_j)$  on  $q_j$ , where  $\theta^i$  and  $\theta_j$  are scaled from the mean of the
            assigned 8 features per qubit. Then, a CNOT gate from  $q^i$  to  $q_j$  is applied,
            adding an entanglement aspect that encodes a cross-term. We repeat a second
            rotation layer on both qubits, culminating in a final Y rotation  $R^y(\alpha)$  that
            further accentuates the polynomial-like mixing. The fixed angles for each
            operation are derived from the feature set's values, ensuring no trainable
            parameters are involved. The advantage of this approach lies in its direct
            simulation of polynomial expansions that classical kernel methods employ,
            only here it is done in the Hilbert space. By capturing cross-terms through
            entanglement, the circuit might better differentiate images with subtle
            shape variations present in the PCA features. The expected outcome is a
            structured quantum state where second-order interactions and individual
            contributions combine in a well-defined, stable pattern, aligning with the
            performance improvements seen in polynomial kernels.
38     - socres:
39         - Originality: 7.5
40         - Feasibility: 8.0
41         - Versatility: 6.5
42
43 - Quantum Radial Sphere Map:

```

```

44 - explanation:
45   - In this design, we treat each of the 80 PCA features as defining a radial
distance in a 10-dimensional spherical coordinate system, which we then map
onto 10 qubits. We begin by normalizing each feature vector  $x = (x_1, \dots, x_{80})$ 
so that  $\sum_j x_j \leq 10$ . Interpreting  $x_j$  as radial components, each qubit  $j$ 
receives  $R^\gamma(\beta_j)$  and  $R_z(\gamma_j)$  gates, where  $\beta_j \propto x_j$  and  $\gamma_j \propto (x_j)^2$  to highlight
nonlinearity. A subsequent multi-qubit entangling layer of controlled-Y
gates (one for each pair  $(j, j+1)$ ) encloses the radial arcs in a correlated
structure. Conceptually, we are mapping features onto a high-dimensional "
sphere" by layering single-qubit rotations that reflect radius-like
expansions. The  $R_z(\gamma_j)$  gates add a second-order nuance, capturing
distinguishable curvature for each feature channel. Because the radial-based
encoding forces data to lie on a quantum Bloch-sphere submanifold, small
differences in radial displacement can become noticeable in the entangled
Hilbert space. This approach helps those digits that appear similar in
amplitude but differ in curvature or squared intensity. Overall, the design
is purely fixed-angles are direct functions of the input feature magnitudes-
letting the quantum circuit act as a robust spherical mapping mechanism
devoid of trainable gates. The hope is that radial representations better
capture local data variations, especially for curved digit features in MNIST
, and that these subtle arcs, once entangled, magnify classification
boundaries more effectively than linear embeddings alone.

46 - socres:
47   - Originality: 7.5
48   - Feasibility: 8.0
49   - Versatility: 6.5
50
51 - Block Mixing Feature Map:
52   - explanation:
53   - This method segments the 80 PCA features into 10 separate 'blocks,' each
corresponding to one qubit. Within each block, the 8 features are encoded
sequentially using fixed single-qubit rotations, but interspersed with multi-
qubit gates to preserve partial intermediate states and allow them to
interact. Specifically, for qubit  $j$ , we define 8 angles  $\{\theta_{j1}, \theta_{j2}, \dots, \theta_{j8}\}$ ,
each proportional to one of the 8 features in the  $j$ -th block. We apply  $R^\gamma(\theta_{j1}) \rightarrow R^\gamma(\theta_{j2}) \rightarrow \dots \rightarrow R^\gamma(\theta_{j8})$  in order, but after each rotation, we use a ring
of iSWAP gates across all qubits. The iSWAP ring ensures that the partial
quantum states from each qubit get 'swapped around', introducing cross-block
correlations. Because the iSWAP gate swaps amplitude and phase information,
each qubit's state after a single step partially depends on the states of
the other qubits, building a collective representation. This approach is
reminiscent of 'block encoding plus mixing' and might help the classifier
better discern subtle correlations across different image regions compressed
by PCA. The final result is a 10-qubit state that progressively merges
local embedding information across blocks. No parameters are learned; the
angles depend directly on the data. The repeated interleaving of local
embedding steps with global iSWAP mixing yields a more entangled encoding
than purely local maps, but the structure remains simple enough to be
executed quickly in simulation. We anticipate that this layering of partial
embeddings can capture nonlinear correlations while avoiding large circuit
depths or complicated parameter tuning.

54 - socres:
55   - Originality: 7.5
56   - Feasibility: 8.0
57   - Versatility: 6.5
58
59 - Random Supremacy Embedding:
60   - explanation:
61   - This approach leverages the concept of 'quantum supremacy circuits'-random but
well-controlled entangling gate patterns-to embed the 80-dimensional
features into a highly non-trivial superposition. In detail, each qubit  $j$  is
initialized with a rotation  $R^\gamma(\theta_j)$  based on the average of 8 features from
the 80, ensuring coverage of all features. Then, a randomly generated
entangling pattern is unleashed: for example, apply a layer of 2-qubit gates

```

(like CZ or iSWAP) between qubits (1,2), (3,4), (5,6), (7,8), (9,10), and then a second layer for qubits (2,3), (4,5), (6,7), (8,9), while skipping pairs that might lead to excessive depth. We follow this random pattern with an additional single-qubit  $R_z(\phi_j)$  rotation, using a feature-based, statically assigned  $\phi_j \propto (x_j)^2$ . By applying random entangling gates, we effectively spread the features throughout the Hilbert space in unpredictable ways, which sometimes leads to highly expressive states for classification tasks. The design remains parameter-free, with randomness pre-selected offline (the same random pattern is used for every data point). Because random circuits are known to sample from complex distributions, they may help highlight fine details across digits. The goal is to push the encoding's representational capacity to the limit, effectively exploring data separation in a wide portion of the state space. This technique can be repeated or simulated at moderate depths without major hardware constraints. We expect it to yield interesting classification benefits from a structure that is partially reminiscent of chaotic classical transformations, yet still harnessing quantum entanglement for better expressiveness.

62 - socres:

63 - Originality: 7.5

64 - Feasibility: 6.0

65 - Versatility: 7.0

66

67 - Pairwise Phase Correlation Map:

68 - explanation:

69 - This design encodes pairwise interactions more explicitly by segmenting the 80 features into 40 pairs. Each pair  $(x^i, x_j)$  is then mapped onto a single qubit using a phase-encoding approach: we initialize each qubit in  $|0\rangle$ , apply  $R^\gamma(\alpha^i)$  with  $\alpha^i \propto x^i$ , then apply a controlled-Z gate to add a phase shift proportional to  $x_j$ , and finally conclude with a further  $R^\gamma(\beta^i)$  to incorporate a second pass of  $x^i$ . We do this for 10 qubits, each assigned 4 pairs to process in sequence. Because we only have 10 qubits, we cycle through the 40 pairs in small batches, reusing the same qubits for multiple pairs. After each pair's encoding, a SWAP gate with an ancillary buffer qubit can help preserve older encodings' contributions, though the design must remain mindful of circuit depth. The emphasis is on capturing explicit synergy: if  $x^i$  and  $x_j$  are both large, the resulting phase shift is more pronounced, highlighting that dimension pair. The circuit stays fixed, with all angles assigned from the data. Since each qubit eventually encodes multiple pairs, the final state is an intricate overlap of numerous phase interactions. We expect that digits which share certain pairwise feature patterns in the PCA space will be recognized by the circuit's structure. As the approach directly writes pairwise correlations onto single qubits (augmented by entangling gates), it may help uncover second-order relationships among local aspects of the images.

70 - socres:

71 - Originality: 7.5

72 - Feasibility: 6.5

73 - Versatility: 7.0

74

75 - Frequency Fourier Embedding:

76 - explanation:

77 - We embed each of the 80 PCA features in frequency space by applying fixed-phase Fourier transforms on single qubits, effectively turning each qubit's state into a small local frequency domain representation. Concretely, we segment the 80 features into 10 groups. Each group is loaded onto one qubit in the form of  $n \geq 8$  discrete frequency components. We accomplish this by applying a precomputed set of single-qubit gates that approximate a discrete Fourier transform of the group's features. Each qubit's amplitude distribution thereafter mirrors the frequency spectrum. Then we apply a small set of cross-qubit entangling gates (for instance, controlled-S or controlled-phase shifts) to align frequencies across qubits, effectively correlating local frequency bands. Inspired by classical signal processing, this approach aims to transform localized pixel intensities (as captured by PCA) into frequency-like components that may separate digit structural

```

patterns. The resulting multi-qubit state, which is effectively a block of
frequency domain embeddings cross-correlated by entangling gates, might
highlight cyclical or repetitive patterns in digit shapes. This method is
parameter-free, with all transformations determined by a standard discrete
Fourier basis. We expect that for digits, especially those with repeating
strokes or patterns, frequency-based encoding might yield better separation
in the resulting quantum kernel space, as Fourier modes can represent
repeated shapes more distinctly than raw amplitude-based methods.
78 - socres:
79   - Originality: 7.5
80   - Feasibility: 6.5
81   - Versatility: 6.0
82
83 - Modulated Sine Phase Encoding:
84   - explanation:
85     - This method uses sinusoidal modulation of a single qubit phase for each PCA
      dimension, then distributes these phases across a 10-qubit system in a
      layered manner. First, split the 80 features into 10 partitions. For each
      partition's qubit j, we apply  $R_z(\phi_j)$  where  $\phi_j = k^i \sin(2\pi x^i)$  for each
      feature  $x^i$  in that partition, summed over i, with a small constant  $k^i$  to
      keep angles in  $[0, 2\pi]$ . The sine function introduces a natural periodicity,
      potentially capturing repeated shapes inherent in certain digits. We then
      apply an entangling scheme using a pattern of controlled swaps: specifically
      , each qubit j performs a C-SWAP with qubit j+1, transferring partial
      amplitude in a controlled manner if  $x^i$  surpasses a certain threshold. That
      threshold is also a fixed function of the data distribution (e.g., the
      median data value). The repeated layering of sinusoidal phase shifts with
      controlled swaps fosters a dynamic resonant structure that might highlight
      repeating strokes or loops in digit images. Since everything is specified a
      priori, no training is required. The central idea is that sine-based
      transformations can leverage periodic patterns within handwriting, and
      combining those with controlled swaps injects further entanglement sensitive
      to data thresholds. The final quantum state contains multiple harmonic
      components entangled across qubits, which might help classification in a
      kernel-based quantum SVM or similar method.
86 - socres:
87   - Originality: 7.5
88   - Feasibility: 6.0
89   - Versatility: 6.5
90 """

```

Listing 6. The user prompt for the idea scoring (first round)

```

1 """
2 Round {current_round} / {max_scoring_round} .
3
4 You are tasked with evaluating the following "# Proposed Idea" using the specified
  criteria and providing scores for each criterion. The idea represents a newly
  proposed quantum feature map for the quantum kernel method. Your evaluation and
  scoring should consider multiple perspectives and adhere to the strictest possible
  standards. Finally, after all the scores have been assigned, summarize the rationale
  for each score.
5
6 # Proposed Idea
7 {idea}
8
9 # Related Work
10 {related_work}
11 """

```

Listing 7. The user prompt for the idea scoring (intermediate rounds)

```

1 """

```



```

2 Round {current_round} / {max_scoring_round} .
3
4 Additional Related Work has been provided for further context. Please evaluate the
   proposed idea using the specified criteria and provide scores for each criterion.
5
6 # Related Work
7 {related_work}
8 """

```

Listing 8. The user prompt for the idea scoring (final rounds)

```

1 """
2 Round {current_round} / {max_scoring_round} (Final Round).
3
4 Additional Related Work has been provided for further context.
5
6 This marks the final opportunity to request additional information. Based on the
   provided details, conduct a comprehensive evaluation and ensure scores are assigned
   to each criterion without exception.
7 This round DOES NOT return "is_lack_information"=True. You must provide scores for each
   criterion.
8
9 # Related Work
10 {related_work}
11 """

```

Listing 9. The developer prompt for the idea reflection

```

1 """
2 You are a professor with extensive expertise in quantum computing and machine
   learning, particularly in scientific research.
3
4 Your task is to evaluate quantum machine learning ideas from multiple perspectives,
   incorporating insights from recent academic papers and best practices. Focus on
   refining each idea to improve its accuracy and effectiveness based on the latest
   advancements, while preserving the idea's core structure.
5
6 ### Notes
7 - Incorporate recent advancements: Utilize relevant developments in quantum
   technology and machine learning where applicable to enhance the evaluation.
8 - Provide balanced evaluations: While not all perspectives will apply equally to
   every idea, strive to deliver a thorough and well-rounded assessment.
9 - Design considerations: Restrict encoding to non-trainable parameters and 
   linear functions.
10 - Focus on research-supported improvements: Suggest refinements supported by related
   research, without making fundamental changes to the core concept of the idea.
11 - Retain original content when no changes are required: If no modifications are
   needed, keep the content of each tag unchanged and include it as-is in the output.
12 """

```

Listing 10. The user prompt for the idea reflection

```

1 """
2 Round {current_round} / {max_reflection_round} .
3
4 Carefully review the idea provided in the "# Previous Idea" section, along with its
   score from the "# First Round Score" section. When conducting your evaluation, take
   into account the relevant academic papers and insights listed in the "# Related Work
   " section.
5
6 After your analysis and evaluation:
7 - Refine and improve the idea for high-accuracy where appropriate, ensuring that the 
   core concept remains unchanged.

```

```

8 - If no modifications are required, retain the following tags as-is: 'feature_map_name',
  'summary', 'explanation', 'formula', and 'key_sentences'.
9 - In cases where no changes are necessary, set the 'is_completed' tag to **True** in
  your output.
10
11 # Previous Idea
12 {previous_idea}
13
14 # First Round Score:
15 {previous_score}
16
17 # Related Work
18 {related_work}
19 """

```

Listing 11. The developer prompt for the summary generation

```

1 """
2 You are an academic journal editor. Your task is to thoroughly understand the full
  content of the paper provided by the user and summarize it. When summarizing, rely
  solely on the information from the provided paper and avoid referencing external
  sources. Ensure that the summary accurately reflects the authors' arguments and
  claims. Write a detailed summary of approximately {max_summary_words} words.
3 """

```

Listing 12. The user prompt for the summary generation

```

1 """
2 Follow these steps to create a summary of the paper:
3 1. The full text of the paper is provided in the section titled "## Full content of
  paper." Carefully read and understand its content.
4 2. Create a detailed summary with approximately {max_summary_words} words, focusing on
  the following aspects:
5     - Key findings
6     - Methodology
7     - Results
8     - Future works or potential areas for improvement
9
10 ## Full content of paper
11 {raw_content}
12 """

```

Listing 13. The developer prompt for the code generation

```

1 """
2 You are an expert Python programmer specializing in quantum computing with extensive
  knowledge of the PennyLane library.
3
4 # Task Definition
5 Your task is to implement ideas for quantum feature maps in Python code using the
  PennyLane library. The available operations in PennyLane are listed under the "#
  Available PennyLane Operations" section.
6
7 # Input Data
8 The 'feature_map' method receives the input data 'x', which represents one sample of the
  dataset. This input is an 80-dimensional NumPy array with the shape '(80,)'. Each
  value already normalized to the range [0.0, 1.0].
9
10 # Key Guidelines:
11 1. **Base Code Format**:
12     - Use the provided base code format as the template for all implementations.
13     - Do not change function names or the overall structure of the base code.

```

```

14 - Do not include any operations for measuring quantum states, such as qml.exp or qml
    .measure, etc
15 - Ensure the code adheres to the format specified below:
16     ```python
17     {code}
18     ```
19 - All hyperparameters (e.g., reps, c, etc.) must be defined as arguments in the
    __init__ method of the FeatureMap class.
20
21 2. Initialization Parameters:
22     - When defining hyperparameters other than "self" and "n_qubits", always specify
        default values.
23     - Default values should be derived from the user-provided idea.
24
25 3. Imports and Libraries:
26     - Include all necessary import libraries.
27     - If external libraries are required, ensure they are properly included in the code.
28
29 4. Code Quality:
30     - Use clear and consistent argument names and include type hints for all
        methods.
31     - Add relevant comments to explain the purpose and functionality of the code.
32     - Ensure that the generated code is executable with PennyLane.
33
34 5. PennyLane Operations:
35     - Limit your implementation to the operations listed under the "# Available
        PennyLane Operations" section.
36     - Each PennyLane operation should be explicitly assigned argument names.
37
38 # Available PennyLane Operations
39 {pennylane_operations}
40
41 ---
42
43 # Output Format
44 Implement result should be structured as follows:
45     - class_name: Name of the generated feature map class
46     - code: generated feature map code
47
48 # Notes:
49 - The quantum feature maps you design must strictly follow the base code schema provided
    .
50 - Focus on creating efficient and clear implementations that align with best practices
    in quantum computing and software development.
51 """

```

Listing 14. The user prompt for the code generation

```

1 """
2 Please implement a quantum feature map based on the designs described in the "# Idea"
    section by extending the 'BaseFeatureMap' class using PennyLane code. The idea is
    based on the assumption that it does not depend on the number of qubits. However,
    the generated code will be executed on a 10-qubit simulator.
3
4 # Idea
5 {idea}
6 """

```

Listing 15. The base code for implementing generated feature map idea

```

1 import numpy as np
2 import pennylane as qml
3 from qxmt.constants import PENNYLANE_PLATFORM
4 from qxmt.feature_maps import BaseFeatureMap

```

```

5
6 # above are the default imports. DO NOT REMOVE THEM.
7 # new imports can be added below this line if needed.
8
9
10 class SeedFeatureMap(BaseFeatureMap):
11     """Seed feature map class.
12
13     Args:
14         BaseFeatureMap (_type_): base feature map class
15
16     Example:
17         """
18
19     def __init__(self, n_qubits: int) -> None:
20         """ Initialize the Seed feature map class.
21
22         Args:
23             n_qubits (int): number of qubits
24         """
25         super().__init__(PENNYLANE_PLATFORM, n_qubits)
26         # hyperparameters
27         self.n_qubits: int = n_qubits
28
29     def feature_map(self, x: np.ndarray) -> None:
30         """Create quantum circuit of feature map.
31         The input data is a sample of MNIST image data. It is decomposed into 80
32         features by PCA.
33
34         Args:
35             x (np.ndarray): input data shape is (80,).
36         """
37         # define your quantum feature map here
38         pass

```

## A.2. Validation

In the [Validation component](#), the prompts were created to check whether the generated code appropriately uses the PennyLane library methods, and to assist in correcting any errors. The relevant listings are shown in Listings 16–19.

- Code validation: Listings 16–17
- Error correcting: Listings 18–19

*Listing 16. The developer prompt for the code validation*

```

1 """
2 You are an expert quantum software engineer especially skilled in the PennyLane library.
3
4 Your Task is to extracted argments from the user provided PennyLane function and
5   PennyLane documentation. The Details are as follows:
6 # Task
7 For each function or class in the provided input, follow these steps:
8
9 1. **Extract the Class Name**:
10    - Identify the function or class name from the user-provided code in the "# User
11      Provided PennyLane Class and Method" section.
12    - The class name starts with 'qml.' and ends before the first '(''.
13    - Do not include the parentheses or any characters after them.
14
15 2. **Extract User-Defined Argument Names**:
16    - Identify argument names in the user code based on the patterns described in the "#
17      Expected User Arguments Pattern" section.

```



```

14 - For **Pattern 1**, arguments are values only (no 'arg_name' or '='). This pattern
15   is excluded from validation and should be ignored in this step.
16 - For **Pattern 2** and **Pattern 3**, extract the 'arg_name' portion only. If
17   multiple arguments are present, separate them by commas and exclude type hints,
18   values, and assignment operators ('=').
19 3. **Extract Reference Argument Names from Documentation**:
20   - Refer to the corresponding "Class Name" section in the "# PennyLane Documentation"
21   part.
22   - The arguments are listed as '"name" ("type)": "description"'. Only extract the '
23     name' portion.
24   - If multiple arguments exist, separate them by commas.
25 # Expected User Arguments Pattern
26 - Pattern 1:
27   - qml.Hoge('value') => Expected: ignore
28   - qml.Hoge('value_1', 'value_2') => Expected: ignore
29 - Pattern 2:
30   - qml.Hoge('arg_name = value') => Expected: arg_name
31   - qml.Hoge('arg_name=value') => Expected: arg_name
32   - qml.Hoge('arg_name_1=value_1', 'arg_name_2=value_2') => Expected: arg_name_1,
33     arg_name_2
34   - qml.Hoge('value_1', 'arg_name_2=value_2') => Expected: arg_name_2
35 - Pattern 3:
36   - qml.Hoge('arg_name: arg_type = value') => Expected: arg_name
37   - qml.Hoge('arg_name:arg_type=value') => Expected: arg_name
38   - qml.Hoge('arg_name_1:arg_type=value_1', 'arg_name_2:arg_type=value_2') => Expected
39     : arg_name_1, arg_name_2
40   - qml.Hoge('value_1', 'arg_name_2:arg_type=value_2') => Expected: arg_name_2
41 # Output JSON Format
42 ``` json
43 [
44   {{
45     "class_name": "Name of the user provided PennyLane class (step1)",
46     "user_args_name": "Argument name list that extracted user code (step2)",
47     "docs_args_name": "Argument name list that extracted pennylane documentation (
48       step3)"
49   }},
50   // Repeat for each case
51 ]
52 ```
53 """

```

Listing 17. The user prompt for the code validation

```

1 """
2 PennyLane method and class list is provided in "# User Provided PennyLane Class and
3   Method" section. PennyLane documentation is provided in "# PennyLane Documentation"
4   section. Please extract the arguments from the user provided PennyLane method and
5   PennyLane documentation.
6
7 Make sure the output format is defined in "# Output JSON Format" section.
8
9 # User Provided PennyLane Class and Method
10 {methods}
11
12 # PennyLane Documentation
13 {references}
14 """

```

Listing 18. The developer prompt for the error correcting

```

1 """

```

```

2 You are an expert qunatum computing software engineer skilled in designing quantum
  feature maps using the PennyLane library in Python.
3
4 ## Task Definition
5 You are tasked with fixing the errors and warnings in the quantum feature map code
  provided by the user.
6 Your task is only fix errors abd warnings. Do not change the logic or structure of the
  code.
7 """

```

Listing 19. The user prompt for the error correcting

```

1 """
2 Please correct the following errors in the quantum feature map code.
3
4 ## Code:
5 {code}
6
7 ## Errors
8 {error_messages_string}
9
10 ## Warnings
11 {warning_messages_string}
12
13 # Output Format
14 Make sure the code follows the same structure as the base code provided and is formatted
  in the following JSON format:
15 - class_name: Name of the generated feature map class
16 - params: dictionary of parameters for the feature map class
17 - code: generated feature map code
18 """

```

### A.3. Review

In the [Review component](#), prompts are used to determine the next direction based on the ideas and the execution results. The related listings are shown in Listings 20 and 21.

Listing 20. The developer prompt for the review

```

1 """
2 You are an expert in quantum physics and quantum machine learning, specializing in
  quantum feature map design.
3
4 # Task
5 Your task is to review past ideas on quantum feature maps and their evaluation results,
  and propose improvements to enhance accuracy through continuous refinement. Restrict
  your review to the quantum feature map design itself; do not propose changes to the
  overall model, evaluation metrics, or other workflows.
6
7 # Quantum Feature Map Definition
8 Quantum feature maps ( $\Phi(x)$ ) will be used to compute the quantum kernel  $K(x, x') = |\langle \Phi(x) | \Phi(x') \rangle|^2$  for a QSVM (Quantum Support Vector Machine).
9 1. Design Considerations:
10    - Define combinations of quantum gates.
11    - Define an entanglement pattern for the features.
12    - Specify the method for embedding input data and quantum states as rotation angles
      of quantum gates.
13    - Ensure the 80-dimensional input data is utilized effectively, minimizing any loss
      of information.
14    - Avoid excessive feature compression that may lead to information loss (e.g.,
      simple feature averaging, summing, etc.).
15 2. Restrictions on Encoding and Embedding:
16    - Only linear functions are allowed for encoding and embedding.

```

```

17     - All parameters in the encoding and embedding must be **non-trainable**.
18
19 # Output Format
20 1. Keep Points:
21     - Identify factors that contributed to improved accuracy.
22     - Analyze the most accurate idea in detail.
23     - Review multiple ideas to identify common elements that contributed to improving
      accuracy.
24 2. Suggestions:
25     - Limit the number of suggestions to {max.suggestion.num} or fewer.
26     - Ensure each suggestion includes only a single proposal.
27     - Prioritize the most impactful suggestions.
28     - If no suggestions for improvement are identified, return suggestions: ["COMPLETED
      "].
29 3. Output Schema: {{
30     "keep_points": ["point 1", "point 2", ..., "point n"],
31     "suggestions": ["suggestion 1", "suggestion 2", ..., "suggestion n"]
32 }}.
33
34 # Notes
35 - Input Data: The input data, originally represented as 784-dimensional image data, has
      been compressed to 80 dimensions using PCA and each value normalized to the range
      [0.0, 1.0]
36 - Simulation: Use an ideal quantum simulator without noise for evaluation
37 - Evaluation Metric: Classification accuracy is the primary metric. The goal is to
      achieve the highest possible accuracy.
38 """

```

Listing 21. The user prompt for the review

```

1 """
2 The previous idea and experimental results are provided below in the "# Previous Trial
  Idea and Results" section. These reflect iterative adjustments based on past trial
  review comments.
3
4 Review the trial results to design a quantum feature map for a more accurate QSVM.
  Identify the factors that contributed to accuracy improvement and areas for further
  enhancement. Finally, check whether the review results comply with the design rules
  for the quantum feature map.
5
6 # Previous Trial Idea and Results (Trial Number: {last-trial.num})
7 {last-trial.results}
8 """

```

## B. Placeholder in Prompts

The prompts contain multiple placeholders. The definition of each value is shown in Table 2. These values are dynamically set at runtime, substituted into the prompt, and passed to the LLMs.

Table 2. Placeholder for our agent system

Component	Name	Description
Generation	{max_trial_num}	The maximum number of experimental trial
Generation	{current_trial}	The number of the current trial
Generation	{device_n_qubit}	The number of qubits available on quantum device
Generation	{idea_num}	The maximum number of ideas generated simultaneously per trial
Generation	{previous_trial}	The number of the previous trial
Generation	{review_comment}	The review comment of the previous trial
Generation	{few_shot_examples}	The few shot examples for idea scoring (Detailed in Listing 5)
Generation	{current_round}	The number of current round for idea scoring
Generation	{max_scoring_round}	The maximum number of rounds for idea scoring
Generation	{related_work}	The summary of papers retrieved from the database
Generation	{max_summary_words}	The maximum number of words in the summary per paper
Generation	{raw_content}	The full text of paper to be summarized
Generation	{max_reflection_round}	The maximum number of rounds for idea reflection
Generation	{previous_idea}	The generated idea of the previous trial
Generation	{previous_score}	The idea score of the previous trial
Generation	{code}	The template for the implementation code
Generation	{idea}	The generated ideas in the current trial
Generation	{pennylane_operations}	The methods available in PennyLane and their descriptions
Validation	{methods}	The PennyLane methods used in generated code
Validation	{references}	The documentation of the methods retrieved from the database
Validation	{error_messages_string}	The error messages encountered during program validation
Validation	{warning_messages_string}	The warning messages encountered during program validation
Review	{max_suggestion_num}	The maximum number of suggestions
Review	{last_trial_num}	The trial number of the idea under review
Review	{last_trial_results}	The generated ideas and their evaluation result
Review	{quantum_gate_list}	The list of quantum gates available in the program

### C. Hyperparameters

In our developed agent system, all seven types of hyperparameters are set at runtime. Table 3 summarizes their overview and the values used in this study.

Table 3. Hyperparameter for our agent system

Component	Name	Description	Value
All	n_qubits	The number of qubits available on quantum device	10
All	max_trial_num	The maximum number of experimental trial	30
Generation	max_idea_num	The maximum number of ideas generated simultaneously per trial	2
Generation	max_scoring_round	The maximum number of scoring rounds per idea	3
Generation	max_reflection_round	The maximum number of reflection rounds per trial	3
Generation	max_paper_per_query	The maximum number of papers retrieved per search query	3
Review	max_suggestion_num	The maximum number of improvement suggestions during idea review	3

### D. Generated Feature Map Code

In the developed agent system, executable Python code is automatically generated and validated by an LLM for each trial. The generated code that achieved the highest accuracy on the Validation dataset is shown in Listing 22. The corresponding quantum circuit diagram is shown in Figure 4.

Listing 22. Generated feature map code by our agentic system

```

1 import numpy as np
2 import pennylane as qml
3 from qxmt.constants import PENNYLANE_PLATFORM
4 from qxmt.feature_maps import BaseFeatureMap
5
6 # new imports can be added below this line if needed.
7
8
9 class AdaptiveSingleAxisWithMidRangeCRotAndISWAPFusionFeatureMap(BaseFeatureMap):
10     """
11     Adaptive Single-Axis with Mid-Range CRot and ISWAP Fusion Feature Map.
12
13     This feature map partitions the 80-dimensional PCA-reduced input into 5 layers (each
14     with 16 features).
15     For each layer  $l$  ( $l = 1, \dots, 5$ ):
16
17     - Local Encoding:
18         The first 10 features are encoded on a 10-qubit register using fixed  $RY(\pi * x)$ 
19         rotations,
20         ensuring efficient local encoding.
21
22     - Stage 1 (Immediate Neighbor Entanglement):
23         For each qubit  $j$ , two designated entanglement features are selected from the
24         block starting at index  $16*l + 10$ :
25          $x_a = x[16*l + 10 + (j \bmod 6)]$ 
26          $x_b = x[16*l + 10 + ((j+1) \bmod 6)]$ 
27         The rotation angle is computed as:
28          $angle = \pi * (0.5*x_a + 0.5*x_b + 0.1*(x_a - x_b))$ 
29         A CRX gate is applied between qubit  $j$  and  $(j+1) \bmod n\_qubits$ .
30
31     - Stage 2 (Next-Nearest Neighbor Entanglement):
32         For each qubit  $j$ , three features are selected at indices  $(j \bmod 6)$ ,  $((j+2) \bmod 6)$ ,
33         and  $((j+4) \bmod 6)$ 
34         from the same block. Their average is used to compute the rotation angle ( $\pi$ 
35         times the average),
36         and a CRY gate is applied between qubit  $j$  and  $(j+2) \bmod n\_qubits$ .
37
38     - Stage 3 (Mid-Range Entanglement):
39         For each qubit  $j$ , the average of the same two features as in Stage 1 ( $x_a$  and
40          $x_b$ ) is computed,
41         scaled by an adaptive factor  $\lambda_\ell$  (provided via lambda_factors), yielding an
42         angle:
43          $angle = \pi * \lambda_\ell * (0.5*x_a + 0.5*x_b)$ 
44         A CRot gate (with theta=0.0 and omega=0.0) is applied between qubit  $j$  and  $(j+3) \bmod n\_qubits$ .
45
46     - Stage 4 (ISWAP Fusion):
47         To integrate non-nearest neighbor correlations, an ISWAP-like interaction is
48         applied using a parameterized
49         IsingXY gate. For  $j = 0, \dots, n\_qubits/2 - 1$  (to avoid duplication), the gate is
50         applied on qubits  $j$  and  $j + n\_qubits/2$ 
51         with rotation angle  $\pi * \gamma$ , where  $\gamma$  is a scaling factor.
52
53     - Global Entanglement:
54         A MultiRZ gate aggregates features from all layers with a rotation angle
55         computed as:
56          $global\_angle = \pi * \sum_\ell [\Delta_\ell * x[16*l+10]]$ 
57         where  $\Delta_\ell$  are fixed non-uniform weights.
58
59     Note: The input  $x$  is expected to have shape (80,).
60
61     Parameters:
62         n_qubits (int): Number of qubits (ideally 10).

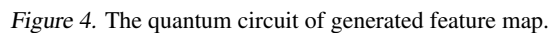
```



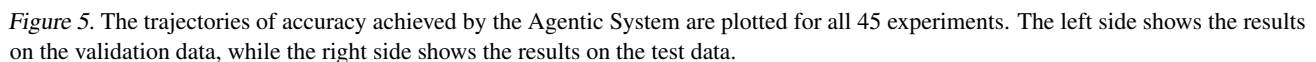
```

53     lambda_factors (list): A list of 5 adaptive scaling factors for Stage 3. Default
54         is [0.3, 0.3, 0.3, 0.3, 0.3].
55     delta_weights (list): A list of 5 weights for global entanglement. Default is
56         [0.15, 0.25, 0.35, 0.15, 0.10].
57     gamma (float): Scaling factor for the ISWAP-like fusion interaction. Default is
58         0.5.
59 """
60 def __init__(self, n_qubits: int, lambda_factors: list = None, delta_weights: list =
61     None, gamma: float = 0.5) -> None:
62     super().__init__(PENNYLANE_PLATFORM, n_qubits)
63     self.n_qubits = n_qubits
64
65     # Adaptive scaling factors for mid-range entanglement (Stage 3)
66     if lambda_factors is None:
67         self.lambda_factors = [0.3, 0.3, 0.3, 0.3, 0.3]
68     else:
69         self.lambda_factors = lambda_factors
70
71     # Global entanglement weights
72     if delta_weights is None:
73         self.delta_weights = [0.15, 0.25, 0.35, 0.15, 0.10]
74     else:
75         self.delta_weights = delta_weights
76
77     # Scaling factor for the ISWAP fusion stage (implemented via IsingXY gate)
78     self.gamma = gamma
79
80 def feature_map(self, x: np.ndarray) -> None:
81     expected_length = 5 * 16
82     if len(x) != expected_length:
83         raise ValueError(f"Input data dimension must be {expected_length}, but got {
84             len(x)}")
85
86     # Process each of the 5 layers
87     for l in range(5):
88         base = 16 * l
89
90         # Local Encoding: Apply RY rotations for qubits 0 through 9
91         for j in range(self.n_qubits):
92             angle_ry = np.pi * x[base + j]
93             qml.RY(phi=angle_ry, wires=j)
94
95         # Stage 1: Immediate Neighbor Entanglement using CRX gates
96         for j in range(self.n_qubits):
97             idx_a = base + 10 + (j % 6)
98             idx_b = base + 10 + ((j + 1) % 6)
99             x_a = x[idx_a]
100             x_b = x[idx_b]
101             angle_immediate = np.pi * (0.5 * x_a + 0.5 * x_b + 0.1 * (x_a - x_b))
102             qml.CRX(phi=angle_immediate, wires=[j, (j + 1) % self.n_qubits])
103
104         # Stage 2: Next-Nearest Neighbor Entanglement using CRY gates
105         for j in range(self.n_qubits):
106             idx1 = base + 10 + (j % 6)
107             idx2 = base + 10 + ((j + 2) % 6)
108             idx3 = base + 10 + ((j + 4) % 6)
109             avg_triple = (x[idx1] + x[idx2] + x[idx3]) / 3.0
110             angle_cry = np.pi * avg_triple
111             qml.CRY(phi=angle_cry, wires=[j, (j + 2) % self.n_qubits])
112
113         # Stage 3: Mid-Range Entanglement using CRot gates
114         for j in range(self.n_qubits):
115             idx_a = base + 10 + (j % 6)
116             idx_b = base + 10 + ((j + 1) % 6)
117             pair_avg = 0.5 * (x[idx_a] + x[idx_b])

```



All 45 experiments were conducted using the same architecture and prompt. For each experiment, the trajectory of the best accuracy at each trial is plotted in Figure 5. The method for calculating best accuracy, which is shown on the Y-axis, is the same as in Figure 2. The system is evaluated using validation and test dataset that described in Section 4.1.



## F. Supplementary Evaluation on Alternative LLMs

In this study, we employed a LLM developed by OpenAI within our system. However, a wide range of LLMs is available regardless of their form of release, including open-source models and closed models accessible via APIs provided by developers. Furthermore, the development of LLMs specialized for specific domains or tasks is actively progressing. To the best of our knowledge, there is currently no publicly available LLM specifically designed for the quantum domain. Therefore, for comparative purposes, we selected two general-purpose LLMs that are conceptually aligned with the OpenAI model: Gemini developed by Google (DeepMind, 2023) and Claude developed by Anthropic (Anthropic, 2024). It should be noted that the prompts used in this study were optimized for the OpenAI model and have not been sufficiently tuned for Gemini or Claude. Accordingly, the results presented in this section are intended to demonstrate that the proposed agentic system can function across different LLMs, rather than to provide a quantitative comparison of performance among LLMs. The hyperparameters used in the experiment are the same as those described in Appendix C, except that `max_iter` was reduced to 15, and `idea_num` was increased to 3.

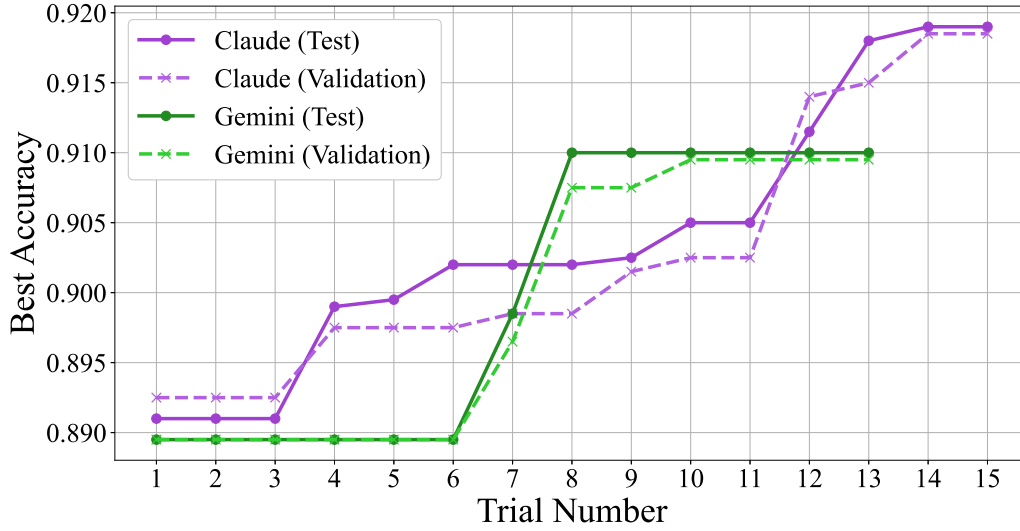


Figure 6. Trajectory of classification accuracy on the MNIST dataset. The green line represents using the Gemini model as the LLM, while the purple line represents using the Claude model.

### F.1. Gemini by Google

The Gemini series includes multiple models designed for different types of tasks. To maintain consistency with the model-task correspondence used for OpenAI’s models, we assigned specific Gemini models to each task category. In particular, “gemini-2.0-pro-exp-02-05” was used for review, idea-generation, idea-reflection, and code-generation tasks; “gemini-2.0-flash-001” was used for scoring and validation; and “gemini-2.0-flash-lite-001” was used for summary tasks.

We confirmed that the proposed agentic system is also capable of generating and iteratively improving quantum feature maps when using the Gemini models (Figure 6, green line). In the case of the Gemini model, the termination of the improvement process was determined by the LLM during the “Review” component following Trial 13. It was also observed that, when using Gemini, the system tends to produce relatively simple quantum feature maps and implementations (Listing 23). In addition, the average processing time per trial was found to be 14 minutes, which is faster compared to other models (OpenAI model is 32 minutes, Claude model is 44 minute). Although all models were accessed via APIs and thus subject to network conditions and latency, the inference response time— even when using reasoning models— was fast overall, even when accounting for such factors.

Listing 23. Generated feature map code by Gemini model

```
1 import numpy as np
2 import pennylane as qml
3 from qxmt.constants import PENNYLANE_PLATFORM
```

```

4 from qxmt.feature_maps import BaseFeatureMap
5
6 # above are the default imports. DO NOT REMOVE THEM.
7 # new imports can be added below this line if needed.
8
9
10 class LEFM_QPERY_b5_2FeatureMap(BaseFeatureMap):
11     """LEFM with Quadratic Post-Entanglement RY (LEFM-QPERY-b5.2) class.
12
13     Args:
14         BaseFeatureMap (_type_): base feature map class
15
16     Example:
17     """
18
19     def __init__(self, n_qubits: int, a_coeffs: np.ndarray = None) -> None:
20         """Initialize the LEFM with Quadratic Post-Entanglement RY feature map class.
21
22         Args:
23             n_qubits (int): number of qubits
24             a_coeffs (np.ndarray): Coefficients for RX rotations. Default is np.ones(
25                 n_qubits).
26         """
27         super().__init__(PENNYLANE_PLATFORM, n_qubits)
28         # hyperparameters
29         self.n_qubits: int = n_qubits
30         self.a_coeffs: np.ndarray = (a_coeffs if a_coeffs is not None else np.ones(
31             n_qubits))
32
33     def feature_map(self, x: np.ndarray) -> None:
34         """Create quantum circuit of feature map.
35         The input data is a sample of MNIST image data. It is decomposed into 80
36         features by PCA.
37
38         Args:
39             x (np.ndarray): input data shape is (80,).
40         """
41         n_features = x.shape[0]
42         n_layers = n_features // (2 * self.n_qubits) # Each layer uses 2*n_qubits
43         features
44
45         for layer in range(n_layers):
46             # RX Encoding Layer
47             for i in range(self.n_qubits):
48                 qml.RX(phi=self.a_coeffs[i] * x[layer * 2 * self.n_qubits + i], wires=i)
49
50             # Entanglement Layer
51             for i in range(self.n_qubits - 1):
52                 qml.CNOT(wires=[i, i + 1])
53                 qml.CNOT(wires=[self.n_qubits - 1, 0])
54
55             # Quadratic RY Rotation Layer
56             for i in range(self.n_qubits):
57                 qml.RY(phi=5.2 * x[layer * 2 * self.n_qubits + self.n_qubits + i] ** 2,
58                     wires=i)

```

## F.2. Claude by Anthropic

The Claude series includes multiple models designed for different types of tasks. To maintain consistency with the model-task correspondence used for OpenAI’s models, we assigned specific Gemini models to each task category. In particular, “claude-3-7-sonnet-20250219” was used for review, idea-generation, idea-reflection, and code-generation tasks; “claude-3-5-haiku-20241022” was used for scoring, validation and summary tasks. The “claude-3-7-sonnet-20250219” model allows for adjusting the depth of reasoning by specifying the number of tokens allocated for the reasoning process. In

this study, we allocated 20,000 tokens for this purpose.

We confirmed that the proposed agentic system also functions effectively when using the Claude model. Compared to other LLMs, the Claude model tended to produce more complex quantum feature maps and generate code of larger scale (Listing 22, 23, 24). On the other hand, some of the generated ideas included nonlinear transformations, which were explicitly prohibited in the prompts. This suggests the need for careful design of the reasoning process and prompt engineering tailored to the specific characteristics of the model.

Listing 24. Generated feature map code by Claude model

```

1 import numpy as np
2 import pennylane as qml
3 from qxmt.constants import PENNYLANE_PLATFORM
4 from qxmt.feature_maps import BaseFeatureMap
5
6
7 class DualPhasePatternEncoder(BaseFeatureMap):
8     """Dual-Phase Pattern Quantum Encoder feature map.
9
10    This feature map implements different phase patterns in first versus second
11    repetition
12    while preserving complementary structures.
13
14    Args:
15        BaseFeatureMap (_type_): base feature map class
16
17    Example:
18        >>> feature_map = DualPhasePatternEncoder(n_qubits=10)
19        """
20
21    def __init__(
22        self,
23        n_qubits: int,
24        scale_factor: float = 0.94 * np.pi,
25        offset: float = np.pi / 3.35,
26        rep1_phase1_even: float = np.pi / 3,
27        rep1_phase1_odd: float = np.pi / 2,
28        rep1_phase2_even: float = np.pi / 4,
29        rep1_phase2_odd: float = np.pi / 2,
30        rep1_phase3_all: float = np.pi / 4,
31        rep2_phase1_even: float = np.pi / 2,
32        rep2_phase1_odd: float = np.pi / 3,
33        rep2_phase2_even: float = np.pi / 2,
34        rep2_phase2_odd: float = np.pi / 4,
35        rep2_phase3_all: float = np.pi / 3,
36        h_mod_phase1: float = np.pi / 6,
37        h_mod_phase2: float = np.pi / 2,
38        h_mod_phase3: float = 5 * np.pi / 6,
39        reps: int = 2
40    ) -> None:
41        """Initialize the Dual-Phase Pattern Quantum Encoder feature map.
42
43    Args:
44        n_qubits (int): number of qubits
45        scale_factor (float, optional): Scaling factor for feature angles. Defaults
46        to 0.94*np.pi.
47        offset (float, optional): Offset for feature angles. Defaults to np.pi/3.35.
48        rep1_phase1_even (float, optional): Phase for even qubits after layer 1 in
49        rep 1. Defaults to  $\pi/3$ .
50        rep1_phase1_odd (float, optional): Phase for odd qubits after layer 1 in rep
51        1. Defaults to  $\pi/2$ .
52        rep1_phase2_even (float, optional): Phase for even qubits after layer 2 in
53        rep 1. Defaults to  $\pi/4$ .
54        rep1_phase2_odd (float, optional): Phase for odd qubits after layer 2 in rep

```



```

1. Defaults to  $\pi/2$ .
rep1_phase3_all (float, optional): Phase for all qubits after layer 3 in rep
1. Defaults to  $\pi/4$ .
rep2_phase1_even (float, optional): Phase for even qubits after layer 1 in
rep 2. Defaults to  $\pi/2$ .
rep2_phase1_odd (float, optional): Phase for odd qubits after layer 1 in rep
2. Defaults to  $\pi/3$ .
rep2_phase2_even (float, optional): Phase for even qubits after layer 2 in
rep 2. Defaults to  $\pi/2$ .
rep2_phase2_odd (float, optional): Phase for odd qubits after layer 2 in rep
2. Defaults to  $\pi/4$ .
rep2_phase3_all (float, optional): Phase for all qubits after layer 3 in rep
2. Defaults to  $\pi/3$ .
h_mod_phase1 (float, optional): Phase for mod 4 = 1 qubits before Hadamard.
Defaults to  $\pi/6$ .
h_mod_phase2 (float, optional): Phase for mod 4 = 2 qubits before Hadamard.
Defaults to  $\pi/2$ .
h_mod_phase3 (float, optional): Phase for mod 4 = 3 qubits before Hadamard.
Defaults to  $5\pi/6$ .
reps (int, optional): Number of repetitions. Defaults to 2.
"""
super().__init__(PENNYLANE_PLATFORM, n_qubits)
# hyperparameters
self.n_qubits: int = n_qubits
self.scale_factor: float = scale_factor
self.offset: float = offset

# Phase pattern for first repetition
self.rep1_phase1_even: float = rep1_phase1_even
self.rep1_phase1_odd: float = rep1_phase1_odd
self.rep1_phase2_even: float = rep1_phase2_even
self.rep1_phase2_odd: float = rep1_phase2_odd
self.rep1_phase3_all: float = rep1_phase3_all

# Phase pattern for second repetition
self.rep2_phase1_even: float = rep2_phase1_even
self.rep2_phase1_odd: float = rep2_phase1_odd
self.rep2_phase2_even: float = rep2_phase2_even
self.rep2_phase2_odd: float = rep2_phase2_odd
self.rep2_phase3_all: float = rep2_phase3_all

# Hadamard pattern phases
self.h_mod_phase1: float = h_mod_phase1
self.h_mod_phase2: float = h_mod_phase2
self.h_mod_phase3: float = h_mod_phase3
self.reps: int = reps

# Define triplets for controlled-Z gates
self.cz_triplets = [
    (0, 3, 6), (1, 4, 7), (2, 5, 8), (3, 6, 9), (4, 7, 0),
    (5, 8, 1), (6, 9, 2), (7, 0, 3), (8, 1, 4), (9, 2, 5)
]

def _encode_features_first_rep(self, x: np.ndarray) -> None:
    """Apply feature encoding for the first repetition.

    First repetition (30 features):
    * Features 1-6 → Rx rotations on qubits 1-6
    * Features 7-24 → Ry rotations on qubits 7-10 and 1-14 (exactly 18 Ry gates)
    * Features 25-30 → Rz rotations on qubits 5-10

    Args:
        x (np.ndarray): Input data of shape (80,)
    """
    # Features 1-6 → Rx rotations on qubits 1-6 (0-5 in 0-indexed)

```

```

105     for i in range(min(6, len(x))):
106         angle = self.scale_factor * x[i] + self.offset
107         qml.RX(phi=angle, wires=i % self.n_qubits)
108
109     # Features 7-24 → Ry rotations on qubits 7-10 and 1-14 (exactly 18 Ry gates)
110     for i in range(6, min(24, len(x))):
111         angle = self.scale_factor * x[i] + self.offset
112         # Map to qubits 7-10 (6-9 in 0-indexed) and then 1-14 (0-13 in 0-indexed)
113         if i < 10: # For features 7-10 map to qubits 7-10 (6-9 in 0-indexed)
114             wire_idx = i
115         else: # For features 11-24 map to qubits 1-14 (0-13 in 0-indexed)
116             wire_idx = (i - 10) % self.n_qubits
117         qml.RY(phi=angle, wires=wire_idx % self.n_qubits)
118
119     # Features 25-30 → Rz rotations on qubits 5-10 (4-9 in 0-indexed)
120     for i in range(24, min(30, len(x))):
121         angle = self.scale_factor * x[i] + self.offset
122         wire_idx = (i - 24 + 4) % self.n_qubits # Map to qubits 5-10 (4-9 in 0-
123             indexed)
124         qml.RZ(phi=angle, wires=wire_idx)
125
126 def _encode_features_second_rep(self, x: np.ndarray) -> None:
127     """Apply feature encoding for the second repetition.
128
129     Second repetition (30 features):
130     * Features 31-36 → Rx rotations on qubits 5-10
131     * Features 37-54 → Ry rotations on qubits 1-18 (exactly 18 Ry gates)
132     * Features 55-60 → Rz rotations on qubits 1-6
133
134     Args:
135         x (np.ndarray): Input data of shape (80,)
136     """
137     # Features 31-36 → Rx rotations on qubits 5-10 (4-9 in 0-indexed)
138     for i in range(30, min(36, len(x))):
139         angle = self.scale_factor * x[i] + self.offset
140         wire_idx = (i - 30 + 4) % self.n_qubits # Map to qubits 5-10 (4-9 in 0-
141             indexed)
142         qml.RX(phi=angle, wires=wire_idx)
143
144     # Features 37-54 → Ry rotations on qubits 1-18 (0-17 in 0-indexed)
145     for i in range(36, min(54, len(x))):
146         angle = self.scale_factor * x[i] + self.offset
147         wire_idx = (i - 36) % self.n_qubits # Map to qubits 1-18 (0-17 in 0-indexed)
148         qml.RY(phi=angle, wires=wire_idx)
149
150     # Features 55-60 → Rz rotations on qubits 1-6 (0-5 in 0-indexed)
151     for i in range(54, min(60, len(x))):
152         angle = self.scale_factor * x[i] + self.offset
153         wire_idx = (i - 54) % self.n_qubits # Map to qubits 1-6 (0-5 in 0-indexed)
154         qml.RZ(phi=angle, wires=wire_idx)
155
156 def _encode_final_layer(self, x: np.ndarray) -> None:
157     """Apply feature encoding for the final layer.
158
159     Final encoding layer (20 features):
160     * Features 61-64 → Rx rotations on qubits 7-10
161     * Features 65-80 → Ry rotations on qubits 1-16 (exactly 16 Ry gates)
162
163     Args:
164         x (np.ndarray): Input data of shape (80,)
165     """
166     # Features 61-64 → Rx rotations on qubits 7-10 (6-9 in 0-indexed)
167     for i in range(60, min(64, len(x))):
168         angle = self.scale_factor * x[i] + self.offset

```

```

167         wire_idx = (i - 60 + 6) % self.n_qubits # Map to qubits 7-10 (6-9 in 0-
168             indexed)
169         qml.RX(phi=angle, wires=wire_idx)
170
171     # Features 65-80 → Ry rotations on qubits 1-16 (0-15 in 0-indexed)
172     for i in range(64, min(80, len(x))):
173         angle = self.scale_factor * x[i] + self.offset
174         wire_idx = (i - 64) % self.n_qubits # Map to qubits 1-16 (0-15 in 0-indexed)
175         qml.RY(phi=angle, wires=wire_idx)
176
177     def _apply_local_entanglement(self) -> None:
178         """Apply CNOT gates between adjacent qubits (Layer 1)."""
179         for i in range(self.n_qubits):
180             qml.CNOT(wires=[i, (i + 1) % self.n_qubits])
181
182     def _apply_medium_entanglement(self) -> None:
183         """Apply CNOT gates between qubits separated by distance 2 (Layer 2)."""
184         for i in range(self.n_qubits):
185             qml.CNOT(wires=[i, (i + 2) % self.n_qubits])
186
187     def _apply_global_entanglement(self) -> None:
188         """Apply CNOT gates between qubits separated by distance n/3 (Layer 3)."""
189         distance = max(1, self.n_qubits // 3) # Ensure distance is at least 1
190         for i in range(self.n_qubits):
191             qml.CNOT(wires=[i, (i + distance) % self.n_qubits])
192
193     def _apply_phase1_rep1(self) -> None:
194         """Apply Phase pattern after Layer 1 in repetition 1:
195         Rz( $\pi/3$ ) to even-indexed qubits and Rz( $\pi/2$ ) to odd-indexed qubits."""
196         for i in range(self.n_qubits):
197             if i % 2 == 0: # even
198                 qml.RZ(phi=self.rep1_phase1_even, wires=i)
199             else: # odd
200                 qml.RZ(phi=self.rep1_phase1_odd, wires=i)
201
202     def _apply_phase2_rep1(self) -> None:
203         """Apply Phase pattern after Layer 2 in repetition 1:
204         Rz( $\pi/4$ ) to even-indexed qubits and Rz( $\pi/2$ ) to odd-indexed qubits."""
205         for i in range(self.n_qubits):
206             if i % 2 == 0: # even
207                 qml.RZ(phi=self.rep1_phase2_even, wires=i)
208             else: # odd
209                 qml.RZ(phi=self.rep1_phase2_odd, wires=i)
210
211     def _apply_phase3_rep1(self) -> None:
212         """Apply Phase pattern after Layer 3 in repetition 1:
213         Rz( $\pi/4$ ) to all qubits."""
214         for i in range(self.n_qubits):
215             qml.RZ(phi=self.rep1_phase3_all, wires=i)
216
217     def _apply_phase1_rep2(self) -> None:
218         """Apply Phase pattern after Layer 1 in repetition 2:
219         Rz( $\pi/2$ ) to even-indexed qubits and Rz( $\pi/3$ ) to odd-indexed qubits."""
220         for i in range(self.n_qubits):
221             if i % 2 == 0: # even
222                 qml.RZ(phi=self.rep2_phase1_even, wires=i)
223             else: # odd
224                 qml.RZ(phi=self.rep2_phase1_odd, wires=i)
225
226     def _apply_phase2_rep2(self) -> None:
227         """Apply Phase pattern after Layer 2 in repetition 2:
228         Rz( $\pi/2$ ) to even-indexed qubits and Rz( $\pi/4$ ) to odd-indexed qubits."""
229         for i in range(self.n_qubits):
230             if i % 2 == 0: # even

```

```

230         qml.RZ(phi=self.rep2_phase2_even, wires=i)
231     else: # odd
232         qml.RZ(phi=self.rep2_phase2_odd, wires=i)
233
234 def _apply_phase3_rep2(self) -> None:
235     """Apply Phase pattern after Layer 3 in repetition 2:
236     Rz( $\pi/3$ ) to all qubits."""
237     for i in range(self.n_qubits):
238         qml.RZ(phi=self.rep2_phase3_all, wires=i)
239
240 def _apply_cz_triplets(self) -> None:
241     """Apply controlled-Z gates to strategic triplets."""
242     for a, b, c in self.cz_triplets:
243         if a < self.n_qubits and b < self.n_qubits and c < self.n_qubits:
244             qml.CZ(wires=[a, b])
245             qml.CZ(wires=[b, c])
246             qml.CZ(wires=[c, a])
247
248 def _apply_fourier_hadamard(self) -> None:
249     """Apply Fourier-Inspired Hadamard Pattern:
250     - Qubit index mod 4 = 0: Apply H gate
251     - Qubit index mod 4 = 1: Apply Rz( $\pi/6$ ) followed by H gate
252     - Qubit index mod 4 = 2: Apply Rz( $\pi/2$ ) followed by H gate
253     - Qubit index mod 4 = 3: Apply Rz( $5\pi/6$ ) followed by H gate
254     """
255     for i in range(self.n_qubits):
256         mod4 = i % 4
257         if mod4 == 0:
258             qml.Hadamard(wires=i)
259         elif mod4 == 1:
260             qml.RZ(phi=self.h_mod_phase1, wires=i)
261             qml.Hadamard(wires=i)
262         elif mod4 == 2:
263             qml.RZ(phi=self.h_mod_phase2, wires=i)
264             qml.Hadamard(wires=i)
265         elif mod4 == 3:
266             qml.RZ(phi=self.h_mod_phase3, wires=i)
267             qml.Hadamard(wires=i)
268
269 def feature_map(self, x: np.ndarray) -> None:
270     """Create quantum circuit of feature map.
271     The input data is a sample of MNIST image data. It is decomposed into 80
272     features by PCA.
273
274     Args:
275         x (np.ndarray): input data shape is (80,).
276     """
277     # First repetition with its specific phase pattern
278     self._encode_features_first_rep(x)
279
280     # Apply entanglement layers with phase pattern for first repetition
281     self._apply_local_entanglement()
282     self._apply_phase1_rep1()
283
284     self._apply_medium_entanglement()
285     self._apply_phase2_rep1()
286
287     self._apply_global_entanglement()
288     self._apply_phase3_rep1()
289
290     # Apply controlled-Z triplet pattern
291     self._apply_cz_triplets()
292
293     # Second repetition with its specific phase pattern
294     self._encode_features_second_rep(x)

```

```
294
295     # Apply entanglement layers with phase pattern for second repetition
296     self._apply_local_entanglement()
297     self._apply_phase1_rep2()
298
299     self._apply_medium_entanglement()
300     self._apply_phase2_rep2()
301
302     self._apply_global_entanglement()
303     self._apply_phase3_rep2()
304
305     # Apply controlled-Z triplet pattern
306     self._apply_cz_triplets()
307
308     # Apply final encoding layer
309     self._encode_final_layer(x)
310
311     # Apply Fourier-Inspired Hadamard Pattern
312     self._apply_fourier_hadamard()
```