QUEST: Quantum-Enhanced Response Selection for Frozen Fine-Tuned LLMs in Real-World Programming Tutoring

Anonymous ACL submission

Abstract

Fine-tuned large language models (LLMs) are increasingly used for programming tutoring, yet their generated responses are not always the most pedagogically useful for novice learners. We frame this issue as a response selection problem: given multiple candidate tutoring answers from a fine-tuned LLM, the system should select the response that best supports student understanding. We study this problem in a real multilingual Python tutoring setting, using question–answer data collected from approximately 1,500 students enrolled in an introductory programming course. We propose **QUEST** (**QU**antum-**E**nhanced **S**election for programming **T**utoring), a compact quantum-enhanced reranker that operates on frozen query–candidate representations from a fine-tuned Qwen2.5 backbone. QUEST models nonlinear interactions between student questions and candidate answers without updating the full LLM. Experiments show that QUEST achieves the highest mean reranking performance among the compared compact rerankers, and complementary automatic and human evaluations suggest that it tends to select more helpful tutoring responses. Our results show that response selection is a parameter-efficient way to improve fine-tuned LLMs for real-world multilingual programming tutoring.

1 Introduction

Fine-tuned large language models (LLMs) are increasingly used as programming tutors for novice learners (Ho et al., 2023; Peng et al., 2025). In realistic classroom settings, students ask short, informal, and often underspecified questions involving both natural language and code. To study this setting (Lee et al., 2022), we collect multilingual question–answer data from approximately 1,500 students enrolled in an introductory Python programming course and fine-tune an LLM to generate tutoring responses.

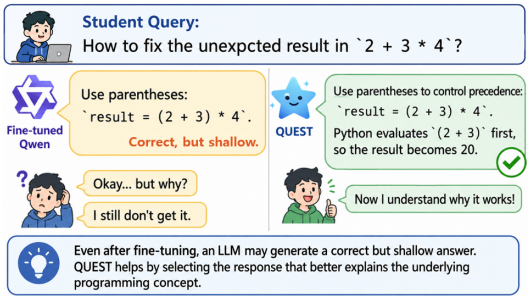

Figure 1: Even after fine-tuning, a tutoring LLM may produce a correct but shallow answer. QUEST reranks candidates to select the response that better supports student understanding.

Yet fine-tuning alone does not guarantee that the first generated answer is the best answer for learning (Pal Chowdhury et al., 2025; Gupta et al., 2025; Kochmar et al., 2025). Fig. 1 shows a representative case. For the query asking how to fix `result = 2 + 3 * 4` when the result is unexpected, a fine-tuned LLM correctly suggests using parentheses: `result = (2 + 3) * 4`. This answer is not incorrect. However, it may miss the student's actual need: understanding why parentheses change the result (Shalyminov et al., 2018). A more useful tutoring response explains that Python evaluates multiplication before addition, and that parentheses force the enclosed expression to run first (Tao et al., 2023; Hu et al., 2024).

The central challenge is that a fine-tuned LLM may produce a correct response that still fails to address the student's underlying conceptual confusion. Programming tutoring, therefore, requires more than fluent generation. It requires selecting the response that best connects the student's question to the relevant programming concept (Puech et al., 2025). We frame this as a response selection problem: given multiple candidate answers from a fine-tuned LLM, the system should select the candidate that is most pedagogically useful (Su et al., 2021; Glass et al., 2022).

We propose **QUEST** (**QU**antum-**E**nhanced **S**election for programming **T**utoring), a compact quantum-enhanced reranker for programming tutoring with fine-tuned LLMs. QUEST operates on frozen query–candidate representations from a fine-tuned Qwen2.5 (Yang et al., 2024) backbone and models non-linear interactions between student questions and candidate answers without updating the full LLM. Across reranking metrics, automatic quality evaluation, and human preference studies with students from a subsequent course offering, QUEST shows positive evidence for selecting more helpful tutoring responses under the same frozen-representation setting.

Our contributions are as follows:

1. **A real multilingual tutoring reranking setting.** We construct response selection tasks from anonymized question–answer data collected from over 1,500 students in an introductory Python course.

2. **A post-generation view of LLM tutoring.** We show that even fine-tuned LLMs require response selection, since correct generated answers may still fail to address students' underlying conceptual needs.

3. **A compact quantum-enhanced response selector.** We introduce a lightweight reranker that models non-linear query–answer interactions over frozen LLM representations, and evaluate it against classical, PEFT-style, and quantum baselines.

2 Deployment Background

2.1 Motivation: Scaling Programming Tutoring in a Large Course

We study tutoring response selection in a real large-scale introductory Python course. The course enrolled 1,508 first-year undergraduate students from diverse majors and was delivered through weekly online lectures, TA-led practice sessions, and bi-weekly programming assignments. The course was primarily delivered in the institution's A-language, but the classroom also included 74 international students. As a result, student support needed to handle A-language queries, English queries, and code-mixed interactions involving programming terms, Python code, and informal descriptions of confusion. Table 1 summarizes the deployment setting. This multilingual context motivated us to construct tutoring data from original A-language and English queries rather than relying on translated-only examples.

Table 1: Summary of the course deployment setting used to construct our tutoring response selection data.

Item	Value
Course type	Python programming
Deployment period	One semester
Course format	Weekly online lectures
Practice format	TA-led practice sessions
Enrollment	1,508 students
International students	74 students
Primary course language	A-language
Additional supported language	English
Tutoring data language	A-language and English
Identifiable user tracking	No

Note. The institution name and original A-language are temporarily anonymized to avoid potential deanonymization during double-blind review.

This setting created a substantial support bottleneck. A limited number of TAs had to support more than 1,500 novice programmers, many of whom asked informal, incomplete, multilingual, or code-mixed questions (Mulryan-Kyne, 2010). In the same deployment context, the AI tutor processed 2,608 student queries, with 53.3% resulting in multi-turn sessions. Student queries were dominated by conceptual questions, assignment clarification, and code debugging, indicating that students needed explanatory tutoring support rather than simple factual lookup.

To address this scale, we trained a course-specific open-source tutoring model under privacy and deployment constraints. The fine-tuning data preserved the multilingual nature of the course and covered both Python concept questions and code-level debugging questions, with answers structured to be concise, beginner-friendly, and aligned with the course context.

2.2 Problem Formulation: Response Selection after Fine-tuning

Fine-tuning improved the model's classroom suitability, but it did not remove the need for response selection. A fine-tuned LLM may produce a response that is correct but still not the most helpful for a novice learner (Lodge et al., 2018). A better tutoring response explains that multiplication has higher precedence than addition and that parentheses force the enclosed expression to run first.

We therefore frame tutoring as a post-generation response selection problem. For each student query, the fine-tuned LLM may generate several plausible answers, and the system must select the one

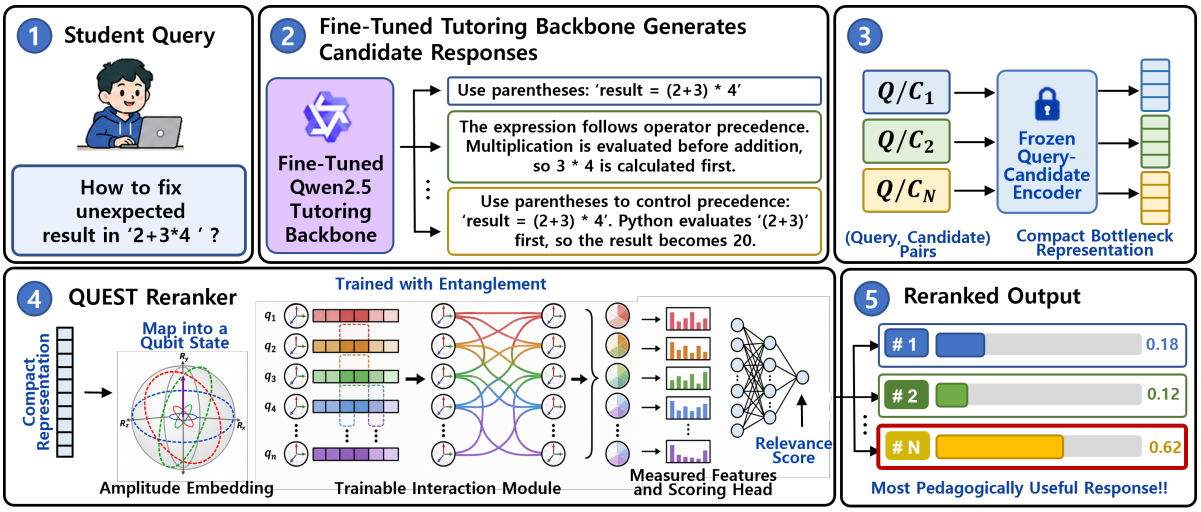

Figure 2: **Overview of QUEST.** A fine-tuned Qwen2.5 tutoring backbone generates multiple candidate responses for a student query, and each query–candidate pair is encoded into a frozen bottleneck representation.

that best supports learning. This selection cannot be based only on surface correctness or fluency. It must also consider whether the response addresses the student's intent, explains the relevant programming concept, and provides enough guidance for a novice learner to understand the underlying rule.

This reflects our deployment observation: the bottleneck is often not generation itself, but choosing the most pedagogically useful response among plausible alternatives. We therefore introduce QUEST, a compact quantum-enhanced reranker over frozen Qwen2.5 query–candidate representations.

3 QUEST

Figure 2 presents an overview of QUEST. Our goal is not to replace the fine-tuned tutoring LLM, but to improve how its outputs are selected. Given a student query, a fine-tuned Qwen2.5 tutoring backbone first generates multiple plausible candidate responses. These candidates are often all reasonable at a surface level, but they differ in pedagogical usefulness: some are brief and correct, while others are more explanatory and better aligned with student understanding. QUEST addresses this selection problem by reranking the candidates using a compact quantum-enhanced scoring module over frozen query–candidate representations.

3.1 From Candidate Generation to Frozen Representations

For each student query, we first use a fine-tuned Qwen2.5 tutoring model to generate a small pool of candidate responses. This candidate pool reflects a realistic tutoring scenario: the backbone can produce several plausible answers, but the system still needs to determine which one is the most helpful for the student. Each student query is paired with each candidate response, forming a set of query–candidate pairs. These pairs are then encoded by a frozen query–candidate encoder derived from the fine-tuned tutoring backbone. Instead of updating the full LLM during reranking, we keep this encoder fixed and extract a compact bottleneck representation for each pair. This representation serves as a shared input space for all reranking methods. As a result, performance differences come from how the reranker models interactions inside the representation, rather than from differences in the underlying language model. This design is important for two reasons. First, it makes the reranking problem modular and parameter-efficient. Second, it isolates the response selection problem itself: all methods see the same query, the same candidate pool, and the same frozen representation, so improvements can be attributed to better response selection.

3.2 QUEST Reranker

The QUEST reranker takes the compact bottleneck representation of a query–candidate pair and converts it into a score indicating how pedagogically useful the candidate response is. As shown in Figure 2, QUEST consists of three stages: amplitude encoding, a trainable quantum interaction module, and a measured-feature scoring head.

Amplitude Embedding. QUEST first normalizes the bottleneck representation and uses it as the amplitudes of a quantum state. Intuitively, this

maps the full query–candidate representation into a shared Hilbert space, where information from the candidate is distributed across a superposed state rather than stored as independent feature coordinates. This representation is useful for tutoring response selection because pedagogical usefulness depends on combinations of signals: a response can be correct and relevant, but still less helpful if it does not explain the student's underlying misconception.

Trainable Interaction Module. After embedding, QUEST applies a parameterized quantum circuit to transform the state. The circuit can be understood as a compact interaction module that learns how to rotate and mix the encoded representation. In our implementation, trainable parameters control rotations around the y- and z-axes. The y-axis rotations adjust the relative contribution of encoded components, while the z-axis rotations adjust their phase-like relationships. The circuit also uses entangling operations, which make the transformation of one qubit depend on another. These operations introduce non-linear dependencies among dimensions of the original bottleneck representation, allowing QUEST to model cases where response quality depends on coupled factors such as correctness, relevance, completeness, and conceptual clarity.

Measured Features and Scoring Head. The transformed state is then measured to produce a small set of numerical readout features. These features summarize the learned query–answer interactions induced by the circuit. A lightweight scoring head maps the measured features, optionally together with the original bottleneck representation, to a candidate score. At inference time, QUEST assigns scores to all candidate responses and selects the highest-scoring one as the final tutoring response.

Compared with an MLP-style reranker, QUEST explicitly introduces a structured interaction step: the representation is first placed into a superposed state, transformed through trainable rotations and entanglement, and then read out as compact interaction features.

3.3 Why QUEST Fits Programming Tutoring

QUEST targets the final decision point in a tutoring pipeline: selecting which generated candidate should be shown to the student. Because the Qwen2.5 backbone and query–candidate encoder remain frozen, the reranker learns only to distinguish among plausible candidate answers. This makes the approach modular and suitable for deployment, improving the final response choice without retraining the full LLM or changing the generation pipeline.

Table 2: Composition of student query types in the deployment logs. The logs were collected from 1,508 students enrolled in an introductory Python programming course.

Query Type	Count	%
Conceptual (course content)	942	36.1
Assignment clarification	602	23.1
Code debugging	571	21.9
Post-hoc assignment explanation	241	9.2
Course scheduling	82	3.1
Attendance-related	67	2.7
Off-topic / non-academic	103	3.8
Total	2,608	100.0

4 Tutoring Response Dataset

4.1 Student Query Collection

We construct the tutoring response selection dataset from anonymized interaction logs collected during the deployment described in Section 2. The logs contain 2,608 student queries from a real introductory Python programming course. Unlike general-purpose programming QA data, these queries reflect how novice students ask for help in an actual classroom: they include conceptual questions, assignment-related confusion, debugging requests, code snippets, error messages, and informal descriptions of misunderstanding.

Table 2 shows the composition of student query types. Most queries are directly related to learning support rather than administrative requests. Conceptual questions account for 36.1% of the logs, followed by assignment clarification (23.1%), code debugging (21.9%), and post-hoc assignment explanation (9.2%). Together, these tutoring-relevant categories cover over 90% of the collected queries. This distribution motivates our response selection setting: the system must choose answers that are not only correct, but also explanatory, context-aware, and appropriate for novice learners.

The dataset also preserves the multilingual nature of the classroom. As shown in Table 3, 62.1% of queries are A-language dominant, 6.0% are English dominant, and 31.9% are code-mixed. We define code-mixed queries as those that combine natural language with Python code, operators, variable names, error messages, or English programming terms. This high proportion of code-mixed queries reflects the linguistic structure of program-

Table 3: Language composition of student queries in the tutoring logs. The dataset preserves the original multilingual classroom setting, including A-language, English, and code-mixed programming queries.

Language Type	Count	%
A-language dominant	1,620	62.1
Code-mixed	832	31.9
English dominant	156	6.0
Total	2,608	100.0

ming help-seeking, where even A-language questions often contain English keywords or executable code.

4.2 Candidate Response Pool Construction

We convert the collected tutoring queries into a response selection task by building a shared candidate pool for each query. For every student query, the fine-tuned Qwen2.5 tutoring backbone generates four plausible candidate responses under the same tutoring instruction. The candidates are not designed as trivial negatives; instead, they represent realistic alternatives that may differ in clarity, completeness, conceptual grounding, or alignment with the student's intent.

All reranking methods select from the same four-candidate pool for each query. This isolates response selection from response generation: the task is not to produce a new answer, but to choose the candidate that best supports student understanding among plausible alternatives. Preference labels are assigned after pool construction using the GPT-4o-based rubric described in Section 4.3, so the candidate generation process is independent of the reranking models. Each query–candidate pair is encoded using the frozen fine-tuned Qwen2.5 backbone and compressed into a compact bottleneck representation. The backbone remains fixed for all methods, allowing us to compare rerankers under the same candidate pool and representation setting.

4.3 Preference Label Construction

After constructing the four-candidate pool for each student query, we assign preference labels using GPT-4o as an automatic pedagogical-quality judge. This label-construction step is performed before training any reranker and is independent of all reranking models. GPT-4o is given the student query and the four candidate responses, but it is not given any information about QUEST, baseline rerankers, model scores, or the identity of the candidate generator.

We use a fixed rubric with four dimensions: correctness, relevance to the student's question, conceptual clarity, and instructional completeness. Correctness measures whether the response gives a technically valid Python explanation or fix. Relevance measures whether the response addresses the specific intent of the student query. Conceptual clarity measures whether the response explains the underlying programming rule rather than only giving a final answer. Instructional completeness measures whether the response gives enough guidance for a novice learner to apply the concept in similar cases.

For each query q_i and its four candidate responses $\{c_{i1}, \ldots, c_{i4}\}$, GPT-4o assigns each candidate a rubric score on each dimension. We normalize each rubric score to $[0, 1]$ and compute a scalar pedagogical-quality score:

$$s_{ij} = w_{\text{cor}} r_{ij}^{\text{cor}} + w_{\text{rel}} r_{ij}^{\text{rel}} + w_{\text{clar}} r_{ij}^{\text{clar}} + w_{\text{comp}} r_{ij}^{\text{comp}},$$

where r_{ij}^{cor}, r_{ij}^{rel}, r_{ij}^{clar}, and r_{ij}^{comp} denote the normalized rubric scores for candidate c_{ij}. Unless otherwise stated, we use equal weights for the four rubric dimensions. The candidate with the highest total score is treated as the preferred response for Acc@1 and MRR. For NDCG, we use the normalized scalar quality score as the graded relevance value.

If two candidates receive the same highest total score, we treat both as relevant for ranking metrics. For the single-label training objective, we break ties by selecting the candidate with the higher conceptual-clarity score, since conceptual explanation is central to our tutoring setting. If the tie remains unresolved after this step, we select the candidate with the higher instructional-completeness score; any remaining ties are broken deterministically by candidate order.

Although GPT-4o is used for preference-label construction, this step is distinct from the GPT-4o-based G-Eval reported in Section 6.2. The label-construction procedure evaluates all candidates within the same candidate pool and produces rubric-level relevance scores used for training and ranking evaluation. In contrast, G-Eval is performed only after all rerankers have selected their final responses; it uses a separate prompt, evaluates one selected response at a time, and is used only as an auxiliary response-quality analysis. Thus, GPT-4o-based labels define the response-selection benchmark, whereas GPT-4o-based G-Eval evaluates the

Table 4: Reranking candidate tutoring responses generated by a full fine-tuned Qwen2.5 backbone. All rerankers use the same frozen query–candidate representations, and results are averaged over three fixed random seeds.

Method	Params	Val Acc@1	Val MRR	Val NDCG	Test Acc@1	Test MRR	Test NDCG
MLP	289	0.586 ± 0.063	0.744 ± 0.049	0.808 ± 0.037	0.602 ± 0.126	0.756 ± 0.082	0.818 ± 0.062
Adapter	297	0.464 ± 0.141	0.661 ± 0.111	0.746 ± 0.085	0.424 ± 0.121	0.635 ± 0.095	0.726 ± 0.072
LoRA	273	0.359 ± 0.056	0.572 ± 0.036	0.678 ± 0.027	0.307 ± 0.025	0.546 ± 0.014	0.659 ± 0.011
DoRA	289	0.174 ± 0.043	0.498 ± 0.016	0.625 ± 0.012	0.174 ± 0.043	0.487 ± 0.028	0.616 ± 0.021
AdaLoRA	281	0.357 ± 0.060	0.572 ± 0.037	0.678 ± 0.027	0.307 ± 0.025	0.545 ± 0.013	0.658 ± 0.010
SoRA	281	0.361 ± 0.054	0.552 ± 0.053	0.661 ± 0.028	0.314 ± 0.021	0.564 ± 0.012	0.643 ± 0.013
QAA	325	0.432 ± 0.053	0.701 ± 0.029	0.779 ± 0.022	0.404 ± 0.036	0.678 ± 0.023	0.761 ± 0.017
QPA	341	0.323 ± 0.059	0.559 ± 0.053	0.669 ± 0.041	0.318 ± 0.027	0.550 ± 0.005	0.662 ± 0.004
QUEST (Ours)	273	$\mathbf{0.693 \pm 0.036}$	$\mathbf{0.827 \pm 0.029}$	$\mathbf{0.871 \pm 0.034}$	$\mathbf{0.628 \pm 0.025}$	$\mathbf{0.786 \pm 0.007}$	$\mathbf{0.840 \pm 0.008}$

quality of method-selected outputs under a separate protocol.

5 Related Work

LLM-based programming tutoring and response selection. LLMs are increasingly used for programming tutoring, including concept explanation, debugging help, assignment clarification, and course-specific question answering (Kazemitabaar et al., 2024). Fine-tuning and retrieval augmentation improve course alignment, but most systems still return a single generated response (Lin et al., 2020). Our deployment setting exposes a remaining post-generation challenge: a fine-tuned tutor may produce a correct but shallow answer that does not resolve the student's conceptual confusion (Gao et al., 2020). We therefore frame programming tutoring as response selection, where a reranker must choose the most pedagogically useful answer from multiple plausible candidates by modeling query–answer interactions beyond surface relevance (Sonkar et al., 2024).

Reranking baselines for frozen LLM representations. We compare QUEST with compact rerankers that test complementary capabilities under the same frozen representation setting. An MLP tests whether standard feed-forward non-linear transformations are sufficient. Adapter (Houlsby et al., 2019), LoRA (Hu et al., 2022), DoRA (Liu et al., 2024), AdaLoRA (Zhang et al., 2023), and SoRA (Ding et al., 2023) test whether strong PEFT-style modules can solve the same response selection problem. QAA (Roh and Kim, 2025) and QPA (Liu et al., 2025) serve as quantum baselines, separating generic amplitude-based or parameterized quantum readout from QUEST's response-selection design. These comparisons distinguish QUEST from compact neural, PEFT-style, and quantum-style rerankers.

Table 5: GPT-4o-based G-Eval (Liu et al., 2023) of selected tutoring responses. GPT-4o scores each reranker-selected answer on four quality dimensions using a 0–5 scale, and Win Rate reports how often each method obtains the highest overall score.

Method	Help.	Comp.	Conc.	Overall	Win Rate
MLP	2.64	2.72	4.31	3.22	15%
Adapter	2.58	2.65	4.33	3.20	12%
LoRA	2.55	2.65	4.28	3.16	10%
DoRA	2.61	2.70	4.29	3.20	13%
AdaLoRA	2.50	2.62	4.29	3.14	8%
SoRA	2.52	2.63	4.27	3.14	9%
QAA	2.44	2.61	4.21	3.09	7%
QPA	2.49	2.64	4.25	3.13	8%
QUEST	**2.92**	**2.98**	**4.36**	**3.41**	**28%**

6 Experiment Results

We evaluate QUEST along four axes: reranking performance under the same frozen fine-tuned LLM representations, GPT-4o-based response quality, student human preference, and controlled boundary diagnostics for non-linear query–answer interactions. Detailed settings for candidate generation, reranker training, evaluation protocols, and baseline implementations are provided in Appendix A.

6.1 Main Reranking Performance

Table 4 reports the main tutoring response reranking results. All methods select from the same candidate pools and use the same frozen query–candidate representations extracted from the fine-tuned Qwen2.5 backbone. Thus, performance differences reflect the reranking module rather than differences in generation or representation extraction.

QUEST achieves the best average performance across all validation and test ranking metrics. On the test set, QUEST obtains the highest Acc@1, MRR, and NDCG, improving over the strongest classical baseline, MLP, from 0.602 to 0.628 in Acc@1, from 0.756 to 0.786 in MRR, and from 0.818 to 0.840 in NDCG. This indicates that

Table 6: Student human preference evaluation of QUEST-selected tutoring responses. Annotators compare QUEST with each baseline and choose the response that is more helpful for learning; columns report the proportions of QUEST wins, ties, and losses.

Comparison	Win	Tie	Lose
QUEST vs. GPT-4o	**0.66**	0.11	0.23
QUEST vs. Qwen2.5	**0.67**	0.08	0.25
QUEST vs. MLP	**0.64**	0.12	0.24

QUEST more effectively ranks pedagogically useful responses near the top of the candidate pool.

The PEFT-style baselines, including Adapter, LoRA, DoRA, AdaLoRA, and SoRA, do not consistently improve over the MLP baseline under the same frozen representation setting. This suggests that simply adding a compact trainable adaptation module is not sufficient for this response selection task. The quantum baselines, QAA and QPA, also underperform QUEST, indicating that the improvement does not come merely from using a quantum-style parameterization. Rather, the results support the importance of QUEST's interaction-aware reranking design.

6.2 Automatic Response Quality Evaluation

Ranking metrics evaluate agreement with the GPT-4o-based preference labels in Section 4.3, but do not directly assess the quality of the selected response. We therefore conduct a separate GPT-4o-based G-Eval after reranker training and model selection. Although both label construction and G-Eval use GPT-4o, they use different prompts, inputs, and outputs: label construction scores all four candidates to define training and ranking labels, whereas G-Eval scores only each method's final selected response and is not used for training, validation, checkpoint selection, or hyperparameter selection.

For each test query, GPT-4o rates the selected response on helpfulness, completeness, conciseness, and overall quality using a 0–5 scale. We also report Win Rate, the fraction of examples for which a method receives the highest overall G-Eval score.

As shown in Table 5, QUEST obtains the highest score across all G-Eval dimensions, with the highest overall score of 3.41 and the highest Win Rate of 0.28. This suggests that QUEST tends to select responses that are not only preferred by the ranking rubric, but also judged as more complete, helpful, and pedagogically appropriate under a separate response-quality evaluation.

Table 7: Inference example comparing a full fine-tuned Qwen2.5 response with the response selected by QUEST. Additional examples are provided in Appendix C.

Student Query (Original):
[Original A-language text is temporarily omitted to avoid potential deanonymization during double-blind review.]
Student Query (English):
How many times does the loop `for number in range(5):` run?

Fine-Tuned Qwen (English):
"`for number in range(5):` repeats 5 times."
→ **Correct but minimal; it does not explain why the loop runs five times.**

QUEST (Ours) (English):
"`range(5)` contains the numbers 0 through 4, so the loop runs 5 times."
→ **Adds the missing boundary explanation, helping novice students understand Python's zero-based range behavior.**

6.3 Human Preference Evaluation

We conduct a human preference evaluation with students from a subsequent offering of the course. The goal of this evaluation is to test whether the responses selected by QUEST are perceived as more helpful by the target learner population, beyond the automatic ranking and G-Eval results. We sample 50 student queries from the test set and construct pairwise comparisons between the response selected by QUEST and a response from one of three baselines: GPT-4o, the fine-tuned Qwen2.5 tutoring backbone, or the strongest classical reranker, MLP.

Thirty student annotators participate in the evaluation. For each comparison, annotators are shown two anonymized tutoring responses for the same student query and are asked which response would be more helpful for a student learning Python. They can choose the QUEST-selected response, the baseline response, or Tie if the two responses are comparable. Annotators are instructed to consider correctness, helpfulness, conceptual clarity, completeness, and whether the response addresses the student's likely confusion. The order of the two responses is randomized, and annotators are not shown the system identity.

Table 6 reports the proportions of QUEST wins, ties, and losses. QUEST is preferred more often than each baseline. It obtains a win rate of 0.66 against GPT-4o, 0.67 against the fine-tuned Qwen2.5 backbone, and 0.64 against MLP. The strongest preference margins are observed against

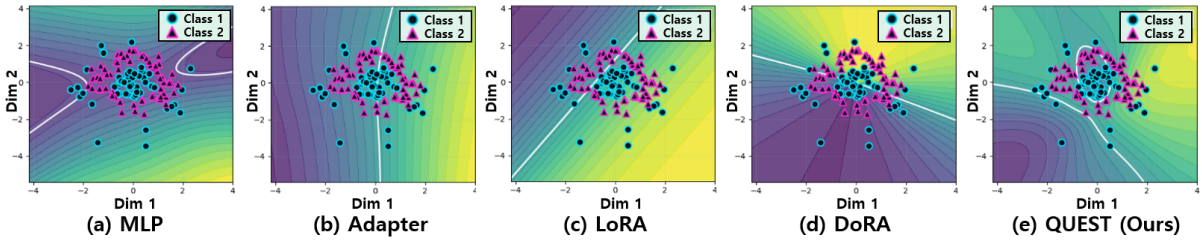

Figure 3: Decision-boundary visualization of the top-performing setting in Table 8. QUEST learns a more flexible non-linear boundary over tutoring query–answer representations than MLP and PEFT-style baselines.

GPT-4o and the fine-tuned backbone, suggesting that post-generation response selection can improve the learner-perceived usefulness of tutoring responses even when the underlying model is already strong or course-adapted. The margin over MLP is smaller, indicating that the strongest classical reranker also captures part of the pedagogical response-selection signal. These results complement the automatic evaluations by showing that students often prefer the responses selected by QUEST.

6.4 Qualitative Analysis

Table 7 provides a representative inference example that illustrates why response selection remains useful after fine-tuning. The student asks how many times `for number in range(5):` runs. The fine-tuned Qwen2.5 response gives the correct answer, stating that the loop repeats five times. However, it does not explain the boundary condition behind this answer. For novice Python learners, this missing explanation is important because `range(5)` is a common source of off-by-one confusion.

In contrast, the response selected by QUEST explains that `range(5)` contains the values 0 through 4. This adds the conceptual reason for the answer, rather than only stating the final count. The example therefore highlights a key distinction in programming tutoring: a response can be correct but still pedagogically incomplete. The fine-tuned backbone response is not wrong, but it omits the reasoning that helps students transfer the concept to future loop and indexing problems.

This qualitative pattern is consistent with the automatic and human evaluations. QUEST tends to prefer responses that connect surface-level answers to the underlying programming rule, such as iteration boundaries, operator precedence, mutability, or conditional syntax. This supports our interpretation that the reranker improves response selection not by generating new content, but by choosing the candidate that better addresses the student's conceptual need. Additional qualitative examples are provided in Appendix C.

Table 8: Controlled radial-ring boundary diagnostic on tutoring query–answer representations.

Method	Params	Val Acc.	Test Acc.	F1	AUC
MLP	129	$.69 \pm .04$	$.62 \pm .07$	$.73 \pm .03$	$.62 \pm .06$
Adapter	125	$.61 \pm .15$	$.55 \pm .18$	$.61 \pm .14$	$.53 \pm .15$
LoRA	99	$.48 \pm .07$	$.49 \pm .08$	$.48 \pm .07$	$.52 \pm .04$
DoRA	101	$.62 \pm .05$	$.59 \pm .04$	$.67 \pm .04$	$.63 \pm .01$
AdaLoRA	123	$.48 \pm .06$	$.51 \pm .07$	$.49 \pm .08$	$.52 \pm .02$
SoRA	123	$.48 \pm .05$	$.48 \pm .09$	$.47 \pm .08$	$.51 \pm .06$
QAA	77	$.32 \pm .01$	$.45 \pm .00$	$.32 \pm .00$	$.56 \pm .05$
QPA	93	$.44 \pm .21$	$.48 \pm .06$	$.24 \pm .41$	$.61 \pm .05$
QUEST	93	$\mathbf{.81 \pm .00}$	$\mathbf{.77 \pm .02}$	$\mathbf{.80 \pm .01}$	$\mathbf{.78 \pm .01}$

6.5 Controlled Interaction Diagnostics

We also conduct a controlled radial-ring diagnostic to test whether compact rerankers can model non-linear boundaries over tutoring query–answer representations. As shown in Table 8, QUEST achieves the highest diagnostic performance, with 0.77 test accuracy, 0.80 F1, and 0.78 AUC, outperforming MLP while using fewer parameters. These results suggest that the learned interaction module provides a useful inductive bias for non-linear response-selection structure.

7 Conclusion

We introduced QUEST, a compact quantum-enhanced reranker for selecting tutoring responses from a frozen fine-tuned Qwen2.5 backbone. Motivated by a real multilingual Python course deployment, we framed tutoring as a post-generation response selection problem, where the goal is to choose the candidate that best supports student understanding. Across reranking metrics, GPT-4o-based quality evaluation, student preference evaluation, and controlled diagnostics, QUEST shows positive evidence for compact interaction-aware response selection without updating the full LLM.

Limitations

This study is limited to a single large introductory Python course at one institution, so the results may not directly generalize to all courses or educational domains. Nevertheless, the deployment includes over 1,500 students, multilingual queries, and diverse programming support needs, providing a realistic testbed for tutoring response selection.

QUEST also depends on the quality of the candidate responses generated by the fine-tuned Qwen2.5 backbone. If no useful candidate is generated, reranking cannot recover an ideal response. However, this matches our intended use case: improving the final choice among plausible tutoring answers without retraining the full LLM.

Finally, our evaluations measure reranking performance, automatic response quality, and student preference, but not long-term learning gains. Future work should study whether QUEST-selected responses improve retention or assignment performance over time. The consistency across our evaluation settings nonetheless supports the usefulness of QUEST as a post-generation selector for programming tutoring.

Ethical Considerations

Human subjects and consent. The tutoring logs and student preference data were collected from an introductory Python course deployment with institutional approval. Students were informed that anonymized interaction data could be used for research, and participation in the preference study was voluntary and did not affect course standing.

Privacy and data handling. All logs were de-identified before analysis by removing direct identifiers such as names, contact information, and student IDs. We do not link queries to individual students in our experiments. Original A-language examples are temporarily omitted or translated for double-blind review to reduce deanonymization risk. Upon acceptance, we plan to release code and de-identified experimental artifacts when permitted by institutional privacy constraints.

Intended use and risks. QUEST is intended to support programming tutoring by selecting more pedagogically useful responses from candidates generated by a fine-tuned tutor. It is not intended to replace instructors or teaching assistants, make grading decisions, or provide administrative judgments. Although response selection can reduce shallow answers, residual risks remain, including incorrect model outputs, incomplete explanations, or overreliance by students. In deployment, the system should be used as assistive support with appropriate instructor oversight.

Use of AI systems. GPT-4o is used for two controlled purposes: automatic preference-label construction and GPT-4o-based G-Eval response-quality evaluation. These two uses follow separate prompts and protocols, as described in Sections 4.3 and 6.2. ChatGPT was used only for language editing, formatting, and proofreading support. All research design, analysis, experiments, and final claims remain the authors' responsibility.

References

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023. Sparse low-rank adaptation of pre-trained language models. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4133–4145, Singapore.

Xiang Gao, Yizhe Zhang, Michel Galley, Chris Brockett, and Bill Dolan. 2020. Dialogue response ranking training with large-scale human feedback data. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 386–395, Online.

Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Naik, Pengshan Cai, and Alfio Gliozzo. 2022. Re2G: Retrieve, rerank, generate. In *Proc. 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2701–2715, Seattle, United States.

Vansh Gupta, Sankalan Pal Chowdhury, Vilém Zouhar, Donya Rooein, and Mrinmaya Sachan. 2025. Are large language models for education reliable across languages? In *Proc. Workshop on Innovative Use of NLP for Building Educational Applications (BEA)*, pages 612–631, Vienna, Austria.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large language models are reasoning teachers. In *Proc. Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14852–14882, Toronto, Canada.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proc. International Conference on Machine Learning*, volume 97, pages 2790–2799.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang,

Weizhu Chen, and 1 others. 2022. LoRA: Low-rank adaptation of large language models. In *Proc. International Conference on Learning Representations (ICLR)*, Virtual.

Songbo Hu, Ivan Vulić, Fangyu Liu, and Anna Korhonen. 2024. Reranking overgenerated responses for end-to-end task-oriented dialogue systems. In *Proc. Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING)*, pages 13970–13991, Torino, Italia.

Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs. In *Proc. CHI Conference on Human Factors in Computing Systems (CHI)*, pages 1–20, Honolulu HI, USA.

Ekaterina Kochmar, Kaushal Maurya, Kseniia Petukhova, KV Aditya Srivatsa, Anaïs Tack, and Justin Vasselli. 2025. Findings of the BEA 2025 shared task on pedagogical ability assessment of AI-powered tutors. In *Proc. Workshop on Innovative Use of NLP for Building Educational Applications (BEA)*, pages 1011–1033, Vienna, Austria.

Changyoon Lee, Yeon Seonwoo, and Alice Oh. 2022. CS1QA: A dataset for assisting code-based question answering in an introductory programming course. In *Proc. Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2026–2040, Seattle, United States.

Zibo Lin, Deng Cai, Yan Wang, Xiaojiang Liu, Haitao Zheng, and Shuming Shi. 2020. The world is not binary: Learning to rank with grayscale data for dialogue response selection. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9220–9229, Online.

Chen-Yu Liu, Chao-Han Huck Yang, Hsi-Sheng Goan, and Min-Hsiu Hsieh. 2025. A quantum circuit-based compression perspective for parameter-efficient learning. In *Proc. International Conference on Learning Representations (ICLR)*, Singapore, Singapore.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. DoRA: Weight-decomposed low-rank adaptation. In *Proc. International Conference on Machine Learning (ICML)*, Vienna, Austria.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2511–2522, Singapore.

Jason M Lodge, Gregor Kennedy, Lori Lockyer, Amael Arguel, and Mariya Pachman. 2018. Understanding difficulties and resulting confusion in learning: An integrative review. In *Frontiers in Education*, volume 3, page 49.

Catherine Mulryan-Kyne. 2010. Teaching large classes at college and university level: challenges and opportunities. *Teaching in Higher Education*, 15(2):175–185.

Sankalan Pal Chowdhury, Terry Jingchen Zhang, Donya Rooein, Dirk Hovy, Tanja Käser, and Mrinmaya Sachan. 2025. Educators' perceptions of large language models as tutors: Comparing human and AI tutors in a blind text-only setting. In *Proc. Workshop on Innovative Use of NLP for Building Educational Applications (BEA)*, pages 356–374, Vienna, Austria.

Xian Peng, Pan Yuan, Dong Li, Junlong Cheng, Qin Fang, and Zhi Liu. 2025. KELE: A multi-agent framework for structured socratic teaching with large language models. In *Proc. Findings of the Association for Computational Linguistics: EMNLP*, pages 16342–16362, Suzhou, China.

Romain Puech, Jakub Macina, Julia Chatain, Mrinmaya Sachan, and Manu Kapur. 2025. Towards the pedagogical steering of large language models for tutoring: A case study with modeling productive failure. In *Proc. Findings of the Association for Computational Linguistics: ACL*, pages 26291–26311, Vienna, Austria.

Emily Jimin Roh and Joongheon Kim. 2025. Quantum-amplitude embedded adaptation for parameter-efficient fine-tuning in large language models. In *Proc. International Conference on Information and Knowledge Management (CIKM)*, Seoul, Korea.

Igor Shalyminov, Ondřej Dušek, and Oliver Lemon. 2018. Neural response ranking for social conversation: A data-efficient approach. In *Proc. EMNLP Workshop SCAI: The International Workshop on Search-Oriented Conversational AI*, pages 1–8, Brussels, Belgium.

Shashank Sonkar, Kangqi Ni, Sapana Chaudhary, and Richard Baraniuk. 2024. Pedagogical alignment of large language models. In *Proc. Findings of the Association for Computational Linguistics: EMNLP*, pages 13641–13650, Miami, Florida, USA.

Yixuan Su, Deng Cai, Qingyu Zhou, Zibo Lin, Simon Baker, Yunbo Cao, Shuming Shi, Nigel Collier, and Yan Wang. 2021. Dialogue response selection with hierarchical curriculum learning. In *Proc. Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1740–1751, Online.

Chongyang Tao, Jiazhan Feng, Tao Shen, Chang Liu, Juntao Li, Xiubo Geng, and Daxin Jiang. 2023. CORE: Cooperative training of retriever-reranker for

effective dialogue response selection. In *Proc. Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3102–3114, Toronto, Canada.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *Proc. International Conference on Learning Representations (ICLR)*, Kigali, Rwanda.

Appendix

A Experimental Settings

This appendix provides additional details for the experimental settings used in the main paper. We use publicly available model and software artifacts, including Qwen2.5, PyTorch, and PennyLane, in accordance with their respective licenses and terms of use. De-identified tutoring data will be released only when permitted by institutional privacy constraints. All experiments were run on NVIDIA H200 NVL GPUs. The main reranking experiments required approximately 1 GPU-hours in total, including hyperparameter selection and three-seed evaluation. The Qwen2.5 backbone was frozen during reranker training, so computational cost was dominated by representation extraction and lightweight reranker optimization. For each method, we select the checkpoint and hyperparameter configuration with the best validation ranking performance and evaluate it on the test set. Table 11 summarizes the hyperparameters considered for all reranking modules.

A.1 Tutoring Response Selection Dataset

We construct the tutoring response selection dataset from 2,608 anonymized student queries collected during the deployment of an introductory Python programming course. The logs include conceptual questions, assignment clarification, code debugging, post-hoc assignment explanation, and a small number of administrative or off-topic queries. We retain the full set of 2,608 queries in the main reranking benchmark to preserve the distribution encountered in deployment. However, our analysis focuses on tutoring-relevant categories, which account for over 90% of the logs.

All splits are performed at the query level. Each query appears in exactly one of the train, validation, or test splits, and all four candidate responses for the same query remain in the same split. This prevents candidate-level leakage across splits. The same query-level splits, candidate pools, frozen Qwen2.5 representations, and bottleneck projections are used for all reranking methods.

A.2 Candidate Response Pool Construction

For each student query, the fine-tuned Qwen2.5 tutoring backbone generates four plausible candidate responses under the same tutoring instruction. These candidates are intended to reflect realistic alternatives that a deployed tutoring model could return. They are not manually designed as trivial negatives. Instead, they may differ subtly in pedagogical quality: one response may be concise but underspecified, another may include a code fix without sufficient explanation, and another may better connect the answer to the underlying programming concept.

All reranking methods select from the same four-candidate pool for each query. This shared-pool design isolates response selection from response generation. The task is therefore not to generate a new answer, but to choose the candidate that best supports student understanding among plausible alternatives. Preference labels are assigned after candidate pool construction using the GPT-4o-based rubric described in Section 4.3.

Each query–candidate pair is encoded using the frozen fine-tuned Qwen2.5 backbone. The resulting representation is compressed into a compact bottleneck vector and used as the input to all reranking methods. Since the candidate pool, backbone, and representation extractor are fixed across methods, performance differences reflect the reranking module itself.

A.3 Main Reranking Experiment

The main reranking experiment evaluates how well each method selects the preferred tutoring response from the four-candidate pool. Each method assigns a scalar score to every candidate response for a given student query. The candidate responses are then ranked by this score. We report Acc@1, MRR, and NDCG.

Acc@1 evaluates whether the top-ranked candidate matches the preferred response. MRR measures whether the preferred response is ranked near the top. NDCG evaluates ranking quality when can-

Table 9: Comparison protocol for the main experiments.

Experiment	Compared Methods	Controlled Factors	Evaluation Output
Main reranking	MLP, Adapter, LoRA, DoRA, AdaLoRA, SoRA, QAA, QPA, QUEST	Same query set, same four-candidate pools, same frozen Qwen2.5 representations	Acc@1, MRR, NDCG
G-Eval	Responses selected by each reranker	Same query set and selected response format	Helpfulness, completeness, conciseness, overall score, win rate
Human preference	QUEST vs. GPT-4o, Qwen2.5, MLP	Same pairwise comparison protocol	QUEST win, tie, QUEST lose
Boundary diagnostic	Same reranker families with compact diagnostic modules	Same low-dimensional tutoring representations and seeds	Accuracy, F1, AUC, decision boundaries

didate responses have graded relevance or quality signals. All results are averaged over three fixed random seeds, and the same data splits are used across all methods.

A.4 GPT-4o-based Response Quality Evaluation

In addition to ranking metrics, we evaluate the quality of the response selected by each reranker using GPT-4o-based G-Eval. This evaluation is protocol-separated from preference-label construction. Although both procedures use GPT-4o, they use different prompts, input formats, and output targets. Label construction scores all four candidates for each query to define training and ranking labels, whereas G-Eval scores only the final response selected by each reranker and is used only for auxiliary response-quality analysis.

For each student query, we provide GPT-4o with the query and one selected response at a time. GPT-4o scores the response on helpfulness, completeness, conciseness, and overall quality using a 0–5 scale. The evaluation prompt instructs GPT-4o to judge whether the response is correct, addresses the student's likely confusion, explains the relevant programming concept, and remains concise enough for a novice learner. We use the same prompt and decoding configuration for all methods.

A.5 Human Preference Evaluation

We conduct a student human preference evaluation to test whether QUEST-selected responses are preferred by learners. Thirty student annotators from a subsequent offering of the course participate in the evaluation. Annotators are shown pairwise comparisons between a response selected by QUEST and a response from a baseline system. The compared baselines include GPT-4o, the fine-tuned Qwen2.5 tutoring backbone, and the strongest classical reranker, MLP. Annotators were shown two anonymized tutoring responses for the same student query and asked: "Which response would be more helpful for a student learning Python?" They could choose Response A, Response B, or Tie if the two responses were comparable. Annotators were instructed to consider correctness, helpfulness, conceptual clarity, completeness, and whether the response addressed the student's likely confusion. Annotators were recruited from students in a subsequent offering of the same course. Participation was voluntary and uncompensated, and non-participation had no effect on grades or course standing. We report the proportions of QUEST wins, ties, and QUEST losses. This evaluation complements automatic G-Eval by measuring whether the selected responses are perceived as more useful by the target learner population.

A.6 Controlled Boundary Diagnostics

The controlled boundary diagnostic is designed to analyze whether each compact module can model non-linear query–answer interaction patterns. This diagnostic is not intended to replace the main tutoring response reranking experiment. Instead, it provides a controlled visualization of how different compact rerankers behave when the decision boundary over tutoring representations requires non-linear separation.

We construct low-dimensional tutoring query–answer representations and apply synthetic boundary rules such as radial-ring, checkerboard, two-moons, and island-like distributions. These settings allow us to visualize whether each method learns a flexible decision boundary or collapses to simpler linear or weakly non-linear patterns. The main paper reports the radial-ring setting because it produced the clearest separation between methods; additional visualizations are provided in Appendix C.

Table 10: Dataset statistics and split information for the tutoring response selection experiments. We retain the full set of 2,608 deployment queries to preserve the distribution encountered in practice. Splits are performed at the query level, and all reranking methods use the same splits, candidate pools, and frozen query–candidate representations.

Split / Item	Queries	Candidates	Query–Candidate Pairs
Train	1,826	4	7,304
Validation	391	4	1,564
Test	391	4	1,564
Total	2,608	4	10,432

All methods are trained and evaluated under the same seeds and data splits. We report validation accuracy, test accuracy, F1, and AUC. We also visualize the learned decision boundaries for the top-performing diagnostic setting.

A.7 Dataset Statistics and Splits

Table 10 reports the dataset statistics used in the main reranking experiments. All methods use the same query-level train/validation/test split, the same four-candidate response pools, and the same frozen query–candidate representations. Each student query is paired with four candidate responses, resulting in four query–candidate pairs per query.

B Implementation of QUEST

Listing 1 shows a compact implementation of the QUEST reranking module. The frozen query–candidate bottleneck vector is amplitude-encoded into a quantum state, transformed by trainable rotations and ring entanglement, and measured with Pauli-Z readouts. The measured interaction features are then used by a lightweight scoring head to produce a candidate score.

```python
import pennylane as qml
import torch
import torch.nn as nn

def quest_layer(theta_layer, wires):
    """One QUEST interaction layer with RY/RZ rotations and ring entanglement."""

    # Local trainable rotations
    for q, w in enumerate(wires):
        qml.RY(theta_layer[q, 0], wires=w)
        qml.RZ(theta_layer[q, 1], wires=w)
    # Ring entanglement
    for q in range(len(wires) - 1):
        qml.CNOT(wires=[wires[q], wires[q + 1]])
    qml.CNOT(wires=[wires[-1], wires[0]])
    # Final local rotation
    for q, w in enumerate(wires):
        qml.RY(theta_layer[q, 2], wires=w)

def make_quest_qnode(n_qubits, depth):
    """Construct a QUEST circuit with amplitude encoding and Pauli-Z readout."""

    wires = list(range(n_qubits))
    dev = qml.device("default.qubit", wires=n_qubits)

    @qml.qnode(dev, interface="torch")
    def circuit(z, theta):
        # z is a normalized bottleneck vector.
        # AmplitudeEmbedding maps z into a 2^n-dimensional Hilbert space.
        qml.AmplitudeEmbedding(
            features=z,
            wires=wires,
            normalize=True,
            pad_with=0.0,)
        # Trainable interaction module
        for layer in range(depth):
            quest_layer(theta[layer], wires)

        # Measurement readout
        z_readout = [qml.expval(qml.PauliZ(w)) for w in wires]
        return z_readout

    return circuit


class QUESTReranker(nn.Module):
    """Amplitude-encoded quantum-enhanced reranker."""

    def __init__(self, bottleneck_dim, n_qubits=4, depth=1):
        super().__init__()
        self.bottleneck_dim = bottleneck_dim
        self.n_qubits = n_qubits
        self.depth = depth

        self.qnode = make_quest_qnode(n_qubits=n_qubits, depth=depth)

        # Trainable circuit parameters:
        # depth x n_qubits x 3 rotations
        self.theta = nn.Parameter(
            0.01 *torch.randn(depth,n_qubits,3))

        # Lightweight scoring head.
        self.head = nn.Sequential(

    nn.Linear(bottleneck_dim+n_qubits,16),
            nn.Tanh(),
            nn.Linear(16, 1),
        )

    def forward(self, z):
        scores = []
        for zi in z:
            # Quantum measured interaction features
            q_feat = self.qnode(zi, self.theta)
            q_feat = torch.stack(q_feat).to(zi.device)
            # Combine frozen representation with measured features
            feat = torch.cat([zi, q_feat], dim=-1)
            # Candidate score
            score = self.head(feat).squeeze(-1)
            scores.append(score)
        return torch.stack(scores)
```

Listing 1: Python-based implementation of QUEST.

C Additional Qualitative Examples

To complement the main qualitative example in Table 7, we provide additional inference examples comparing the response generated by the full fine-tuned Qwen2.5 backbone with the response selected by QUEST. These examples illustrate cases where both responses may be broadly relevant, but the QUEST-selected response better explains the underlying programming concept, removes ambi-

guity, or gives more actionable guidance for novice learners.

For double-blind review, we show English translations and temporarily omit the original A-language queries and responses when they may reveal institutional context. Across these cases, QUEST tends to select responses that are not merely correct, but more precise, explanatory, and pedagogically useful.

Table 11: Hyperparameters and module settings used for the reranking baselines and QUEST. Curly brackets indicate values considered during hyperparameter selection. The fine-tuned Qwen2.5 backbone, query–candidate encoder, candidate pools, and bottleneck projection are fixed for all methods; only the reranking module is trained.

Method	**Hyperparameter**	**Values / Setting**
MLP	Batch size	$\{16, 32\}$
	Optimizer	AdamW
	Scheduler	Linear scheduler
	Learning rate	$\{1e{-}4, 3e{-}4, 1e{-}3\}$
	Hidden dimension	$\{8, 16\}$
	Output	Scalar candidate score
Adapter	Batch size	$\{16, 32\}$
	Optimizer	AdamW
	Scheduler	Linear scheduler
	Learning rate	$\{1e{-}4, 3e{-}4, 1e{-}3\}$
	Adapter bottleneck	$\{4, 8, 16\}$
	Structure	Down-projection $\rightarrow$ nonlinearity $\rightarrow$ up-projection $\rightarrow$ score
LoRA	Batch size	$\{16, 32\}$
	Optimizer	AdamW
	Scheduler	Linear scheduler
	Learning rate	$\{1e{-}4, 3e{-}4, 1e{-}3\}$
	Rank r	$\{4, 8, 16\}$
	Scaling α	$\{8, 16\}$
	Structure	Low-rank update over scoring transformation
DoRA	Batch size	$\{16, 32\}$
	Optimizer	AdamW
	Scheduler	Linear scheduler
	Learning rate	$\{1e{-}4, 3e{-}4, 1e{-}3\}$
	Rank r	$\{4, 8, 16\}$
	Scaling α	$\{8, 16\}$
	Structure	Direction–magnitude decomposed low-rank scoring head
AdaLoRA	Batch size	$\{16, 32\}$
	Optimizer	AdamW
	Scheduler	Linear scheduler
	Learning rate	$\{1e{-}4, 3e{-}4, 1e{-}3\}$
	Initial rank	$\{8, 16\}$
	Target rank	$\{4, 8\}$
	Structure	Adaptive low-rank scoring transformation
SoRA	Batch size	$\{16, 32\}$
	Optimizer	AdamW
	Scheduler	Linear scheduler
	Learning rate	$\{1e{-}4, 3e{-}4, 1e{-}3\}$
	Rank r	$\{4, 8, 16\}$
	Sparsity / selection	Sparse low-rank component selection
	Structure	Sparse low-rank scoring transformation
QAA	Batch size	$\{16, 32\}$
	Optimizer	AdamW
	Scheduler	Linear scheduler
	Learning rate	$\{1e{-}4, 3e{-}4, 1e{-}3\}$
	Encoding	Amplitude encoding
	Circuit depth	$\{1, 2\}$
	Circuit	R_y/R_z rotations + entangling gates
	Readout	Pauli-Z measurement features $\rightarrow$ score
QPA	Batch size	$\{16, 32\}$
	Optimizer	AdamW
	Scheduler	Linear scheduler
	Learning rate	$\{1e{-}4, 3e{-}4, 1e{-}3\}$
	Encoding	X
	Circuit depth	$\{1, 2\}$
	Circuit	Trainable rotations + entangling gates
	Readout	Measured quantum features $\rightarrow$ score
QUEST	Batch size	$\{16, 32\}$
	Optimizer	AdamW
	Scheduler	Linear scheduler
	Learning rate	$\{1e{-}4, 3e{-}4, 1e{-}3\}$
	Encoding	Amplitude encoding of bottleneck representation
	Circuit depth	$\{1, 2\}$
	Interaction module	R_y/R_z rotations + entangling gates
	Readout	Measured interaction features
	Scoring head	Bottleneck / measured features $\rightarrow$ scalar candidate score

Table 12: Additional inference examples comparing the full fine-tuned Qwen2.5 response with the response selected by QUEST. English translations are shown, while original A-language queries and responses are temporarily omitted for double-blind review.

Student Query (Original):
[Original A-language text is temporarily omitted to avoid potential deanonymization during double-blind review.]

Student Query (English):
Does the `list + list` operation modify the original list, or create a new list?

Fine-Tuned Qwen (English):
"The `list + list` operation does not create a new list, but returns one concatenated list. Therefore, the original lists are not changed."
→ **Gives the right conclusion but contains a confusing contradiction about whether a new list is created.**

QUEST (Ours) (English):
"The + operator combines two lists and returns a new list. The original lists are not modified. For example, in `merged_list = list1 + list2`, `list1` and `list2` remain unchanged."
→ **Clearly distinguishes value creation from in-place modification and gives a concrete example.**

Student Query (Original):
[Original A-language text is temporarily omitted to avoid potential deanonymization during double-blind review.]

Student Query (English):
If `result = 2 + 3 * 4` gives an unexpected result, how should I modify it considering operator precedence?

Fine-Tuned Qwen (English):
"You can use parentheses, as in `result = (2 + 3) * 4`, to explicitly set the operator precedence."
→ **Correctly fixes the expression, but gives limited explanation of why the fix works.**

QUEST (Ours) (English):
"Use parentheses to control precedence: `result = (2 + 3) * 4`. Python evaluates `(2 + 3)` first, so the result becomes 20."
→ **Connects the code edit to the underlying execution rule, making the response more pedagogically useful.**

Student Query (Original):
[Original A-language text is temporarily omitted to avoid potential deanonymization during double-blind review.]

Student Query (English):
Find the error in the following code and fix it so that it prints the state for the user-input temperature: `if temperature < 0: state = 'ice'; else temperature > 100: state = 'steam'; else: state = 'water'.`

Fine-Tuned Qwen (English):
"The error is in the `temperature > 100` condition. The corrected code should use `elif temperature > 100:` and keep an `else` branch."
→ **Identifies the broad location of the error but does not clearly state the invalid syntax pattern.**

QUEST (Ours) (English):
"The error comes from using `else temperature > 100`; this should be `elif temperature > 100`. The corrected structure is `if temperature < 0`, `elif temperature > 100`, and then `else`."
→ **Pinpoints the syntactic error and explains the correct conditional-chain structure.**

Note. Original A-language queries and responses are temporarily omitted to avoid potential deanonymization during double-blind review. Upon acceptance, we will disclose the original text subject to institutional and privacy constraints.