Fine-Tuning Open-Source Models as a Viable Alternative to Proprietary LLMs for Explaining Compiler Messages

Lorenzo Lee Solano
University of New South Wales
Sydney, Australia
l.leesolano@unsw.edu.au

Charles Koutcheme
Aalto University
Espoo, Finland
charles.koutcheme@aalto.fi

Juho Leinonen
Aalto University
Espoo, Finland
juho.2.leinonen@aalto.fi

Alexandra Vassar
University of New South Wales
Sydney, Australia
a.vassar@unsw.edu.au

Jake Renzella
University of New South Wales
Sydney, Australia
jake.renzella@unsw.edu.au

Abstract

Cryptic compiler error messages continue to present a significant barrier for novice programmers, especially in foundational languages like C. Although large language models (LLMs) can generate accurate and comprehensible error explanations, their computational requirements, propensity for over-assistance, and privacy concerns constrain their suitability for widespread adoption in educational tools. This work investigates how Supervised Fine-Tuning (SFT) can enhance the performance of smaller, open-source models when explaining C compiler errors to students in introductory programming courses (CS1/2). We derive a training dataset of 40,000 input-output pairs from CS1/2 student C compiler errors to fine-tune three open-source models: Qwen3-4B, Llama-3.1-8B, and Qwen3-32B. Model performance was assessed through a dual evaluation framework involving expert human reviewers and a large-scale automated analysis of 8,000 responses using an ensemble of models as judges. Our results indicate that SFT significantly improves both expert and LLM-as-judge ratings in smaller open-source models, with reduced gains in the larger model. We analyse the trade-offs between model size and quality, and validate LLM-as-judge by demonstrating inter-rater agreement with experts. Our findings demonstrate that fine-tuning smaller models on high-quality data is a viable strategy for creating specialised pedagogical tools. We provide a replicable methodology for enabling broader access to advanced AI capabilities within educational contexts, especially with smaller, economical models.

CCS Concepts

• **Social and professional topics** → *Computing education.*

Keywords

LLMs, programming error messages, fine-tuning, CS1/2

ACM Reference Format:
Lorenzo Lee Solano, Charles Koutcheme, Juho Leinonen, Alexandra Vassar, and Jake Renzella. 2026. Fine-Tuning Open-Source Models as a Viable Alternative to Proprietary LLMs for Explaining Compiler Messages. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1 (SIGCSE TS 2026), February 18–21, 2026, St. Louis, MO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770762.3772576

1 Introduction

Large language models (LLMs) are increasingly being integrated into computing education [3, 22] where they are used for a variety of tasks such as generating instructional content [17, 28, 33], generating automated feedback on buggy code [1, 6, 9, 12, 20, 21], code explanations [13, 18, 28], and tutoring [15]. This growing interest is also visible in the number of studies published focusing on generative AI in computing education [22]. One of the earliest and most prominent use cases has been helping students to understand programming error messages [14, 26, 27, 30, 35].

Existing approaches have largely relied on proprietary LLMs accessed through third-party APIs. While these models are powerful and seem to be helpful, this approach brings a number of practical and ethical limitations. Student code and context must be transmitted to external servers, raising data privacy concerns, particularly in educational environments [8]. In addition, the use of commercial APIs for accessing LLMs often comes with significant financial costs and operational risks, including rate limits, service outages, and changes in model behaviour over time.

Addressing these challenges, we examine the effectiveness of smaller open-source language models, for generating error message explanations in introductory programming (CS1/2). While less capable out-of-the-box [11], these models can be fine-tuned to perform well on specific tasks [10]. Our research question in this work is as follows:

RQ How does Supervised Fine-Tuning on a dataset of CS1/2 error explanations affect the performance of smaller open-source models?

To investigate this, we fine-tuned a range of open-source language models using a large dataset of errors encountered by students and explanations generated by GPT-4.1, which was one of the top proprietary models available at the time of the study. We then compare the small fine-tuned models to proprietary LLMs to evaluate their explanations with regard to clarity and correctness, using a combination of expert and LLM-based annotation techniques for evaluation. Our contributions are threefold:

- We provide a comprehensive benchmark of fine-tuned open-source models compared to commercial LLMs for the task of explaining programming error messages.
- We introduce a rubric for assessing the quality of programming error message explanations.
- We employ both expert judges and LLM-as-judge, with inter-rater reliability between the two groups to scale evaluation to a large dataset.

2 Related Work

LLMs for Programming Error Messages. Research has begun to explore how LLMs can improve programming error messages (PEMs). For example, Leinonen et al. [14] found that Codex, a 2021 coding model, could enhance error messages about half of the time. Taylor et al. [30] found that GPT-3.5 could explain errors in 90% of the cases for compile-time and 75% of the cases for run-time errors. Santos et al. [27] found that providing code context to the LLM improved the generated error message explanations. Wang et al. [35] found that students using LLM-enhanced error messages repeated fewer errors, and resolved errors faster. In all cases, models display a propensity to overhelp, which contradicts pedagogical goals [6, 15, 23]. More recent work evaluating LLM-enhanced compiler error messages from the student perspective has found them to be ineffective in practice [26], largely due to the structure and detail included in the error message, since longer error explanations have previously been shown to be unhelpful [19].

Fine-tuning LLMs in CS Education. Few works in programming education have attempted to fine-tune open-source language models, despite privacy and reliability advantages. Hicke et al. [7] developed and evaluated an LLM-based solution for answering student questions on a programming forum. The authors trained medium size LLaMa-2 models and compared them against alternatives. However, the evaluation was constrained by a small sample size of 20 responses rated by a single annotator, highlighting a need for larger-scale human evaluations with multiple annotators to ensure robust inter-rater reliability. Kumar et al. [1] trained a Llama-3.1-8B model to provide guidance to students through asking Socratic questions. Their approach combined SFT and reinforcement learning, leveraging existing human annotations combined with LLM-generated alternatives. While this approach showcases the effectiveness of training open-source models, the method relies on costly human annotations, and does not support students with programming error messages. Kotalwar et al. [10] trained a Phi-mini model and Llama-3.1-8B model to produce in-browser feedback and Socratic hints for student code. Their results suggested that fine-tuned smaller models could attain performance parity with GPT-3.5 across three Python problem sets [10].

The work presented in this paper extends prior research and lessons learned, with the following core novelties: a) it benchmarks a variety of open-source models across parameter sizes, b) it is tightly integrated into a C compiler, which extends the data collection context to provide better explanations, and c) it provides a distinct evaluation of both compile- and run-time errors.

3 Methods

Our goal is to support novice C programmers by providing guidance for both compile- and run-time errors. While existing solutions rely on proprietary LLMs [15, 16, 30, 35], we focus on the development of fine-tuned open-source alternatives, specialised for delivering pedagogically sound error message explanations. More specifically, we rely on Supervised Fine-Tuning (SFT), which offers a cost-effective approach to refining a language model for specialised or domain-specific tasks.

3.1 Data

Training and evaluation data was sourced from student use of DCC Help, a compiler-integrated approach to generating error message explanations for novice C programming students [30]. For five teaching periods at a large Australian university, between September 2023 and February 2025, we collected approximately 180,000 compile-time and 50,000 run-time student invocations of DCC Help. Each logged example includes the source code, the context of the error, as well as the original DCC Help response generated by either GPT-3.5 Turbo or GPT-4o mini, which we use as a baseline of model performance.

All logged data is pre-processed and redacted in a best effort to remove identifying features, such as student IDs, names and emails, in line with the relevant ethics requirements. We also filtered out a few instances of unbounded recursion, which produced logs of excessive length, unsuitable for training or inference.

3.2 Creating a Training Dataset

Producing a representative labelled dataset required for SFT, consisting of example inputs and desired responses, can be challenging for the application of explaining error messages. As such, past work often utilised small curated or handcrafted datasets [16, 25]. To address this, we propose to distil the abilities of a large language model into smaller models [10]. We generate training data by using a proprietary LLM to provide responses for examples in the DCC Help dataset, resulting in a larger and more diverse training dataset.

Data Processing. We sample 40,000 examples from the first two teaching sessions in the DCC Help dataset, reserving the remaining sessions for evaluation. As invocations of DCC Help are not evenly distributed over time [30], a purely random sample risks disproportionately representing certain weeks and types of examples, such as those common near the deadlines of assignments. Instead, before taking a random sample, we first reduced the data pool by limiting the number of compile- and run-time examples from any given in-session week to a maximum (4,500 for compile-, 2,250 for run-time).

Prompting Strategy. We build on the approach of Taylor et al. [30] to construct prompts based on the available error context, such as compiler messages, source code, call-stack, and variable states. Our revised strategy, shown in Figure 1, additionally enforces a consistent three-part explanation structure that draws from two distinct strands of prior work [10, 21], synthesising research on error message interpretation with methods for structured feedback generation.

Feedback Prompt Template

System description: You are providing programming error messages in an intro programming course using C.
Your inputs are: C Program, Original Error, Variables Stack, Call Stack.

Output the following:

(1) **Error Message Clarification:** Write one short sentence, explaining the error message without programming jargon.

(2) **Potential Causes:** Follow with 1-2 short sentences, identifying and explaining potential issues in the code that may be causing this error.

(3) **Guidance (Hints Generation):** Follow with 1-2 short sentences, giving debugging hints and guidance, potentially including specific references to my code.

Details:

- Keep your response short, friendly, and without jargon.
- Do not give the solution in code.
- Address the student directly.

Figure 1: Three-Step Error Explanation Prompt Strategy.

The first step asks the model to clarify the error message in simple, accessible language. This brief, jargon-free, natural language *translation* of the original compiler message addresses common readability issues present in standard PEMs [2]. The next two steps are motivated by the literature on programming feedback generation [21]: we prompt the model to explain potential causes for the error, then provide actionable hints and guidance to help the student fix their code. This mirrors the approach of Phung et al. [10, 21], using explanations of student errors as a chain-of-thought scaffold for generating higher-quality hints.

Dataset Generation. Using this prompt strategy, we generate responses for the 40,000 examples using *gpt-4.1-2025-04-14*, via OpenAI's batch completions API, with a temperature of 0.

The resulting SFT training dataset consists of a ratio of approximately 3:1 compile- vs run-time examples, with total lengths ranging from 300 to 4,000 tokens (mean length of 849 tokens).

3.3 Fine-tuning Open-Source Models

Using the resulting training dataset, we train three open-source language models of various sizes: Qwen3-32B [32], Llama-3.1-8B [31], and Qwen3-4B [32]. We chose these models to have a representative sample of different sizes. While Qwen3-32B is not a smaller language model, it allows us to contextualize the efficacy of our training approach.

Fine-tuning Details. All models were trained for 1 epoch, with a learning rate of $2e-5$. Training was performed through the Unsloth fine-tuning API, on a single Nvidia A100 GPU (80GB vram).

We train all our models using QLoRa [4], a Parameter-Efficient Fine-Tuning technique (PEFT) that quantises a language model to 4-bit and adds on top a set of trainable parameters called adapters, while the base model remains frozen. Using PEFT techniques has two benefits: quantization reduces memory and computational requirements, facilitating deployment on resource-constrained devices, while adapters enable the recovery of original model functionalities.

3.4 Evaluation Dataset

To evaluate the performance of the fine-tuned open-source models, we construct an evaluation dataset of 8,000 examples, sourced from the remaining three teaching periods in the DCC Help dataset (from February 2024 to February 2025). Similar to our training dataset construction, we first limited the number of examples per week to a maximum (3,000 for compile-, 1,500 for run-time), before randomly sampling.

For this subset, we generate responses using the previously listed base and fine-tuned open-source models, using the Unsloth inference API, on a single A100 GPU. For comparison, we include the original DCC Help response and a proprietary LLM response generated by *gpt-4.1-2025-04-14.* All responses were generated using the same prompt strategy, described in Section 3.2. This produced an evaluation set of approximately 5,600 compile- and 2,400 run-time examples, each of which contains the student code and error context, and eight model responses (total evaluation responses: 64,000). From this, we randomly sampled a smaller dataset of 50 run-time and 50 compile-time examples for the expert (human) evaluation.

3.5 Evaluation Rubric

We define an annotation rubric of eight binary criteria, grounded in prior work on evaluating programming feedback [6, 11, 30]. Each criterion is scored as either 0 or 1, with 1 indicating that the explanation exhibits the desired property.

- Correctness: The explanation is technically correct.
- Selectivity: Contains no incorrect/irrelevant information.
- Completeness: Contains all information critical to understand the error.
- Clarity: Clear, easy to understand, presented in a readable format, using an economy of words.
- Novice Appropriate: Accessible for novices, avoiding technical jargon and advanced knowledge assumptions.
- No Solution: Does not directly provide the full solution, either in code or prose.
- No Overhelp: Avoids over-direction, leaving room for problem solving and critical thinking.
- Socratic: Provides guidance to solve the error, and includes at least one relevant guiding question or statement.

Criteria such as Correctness and Selectivity are common quality criteria for evaluating programming feedback [10, 11, 30], and address typical failure modes of LLM-generated explanations. Clarity and Novice Appropriate reflect established recommendations for effective feedback for novice programmers [2]. No Overhelp and Socratic are drawn from prior work by Ross et al. [24], and focus on the pedagogical function of these explanations in the context of introductory programming. Finally, we take a pedagogical interpretation of Completeness, focusing on whether all information needed to understand the error message is present.

These criteria align with our explanation prompting strategy (see Figure 1), where *Error Message Clarification* helps to address Clarity, Novice Appropriate and Completeness. *Potential Causes*, while not explicitly mapping to any criteria, acts in part as a reasoning step to help address Correctness and Selectivity; and *Guidance* aligns with pedagogically oriented criteria such as Socratic and No Overhelp.

Overall, the structural constraints within the feedback prompt are designed to promote novice-appropriate guidance and prevent the inclusion of explicit solutions.

3.6 Expert Evaluation

Expert evaluation was performed by four experienced CS1 teaching staff members, who each annotated 20 unique examples. An additional subset of 20 examples was annotated by all four experts, which was used to calculate inter-rater reliability (IRR) using Gwet's AC1 [5]. This brings the number of annotations per expert to 40 examples or 320 responses. Gwet's AC1 was chosen as some metrics had very high degrees of agreement between raters, which introduced errors with more typical IRR metrics.

Annotators were presented with the source code and error context for each example and were tasked with first annotating each model response against the evaluation rubric before ranking the same model responses from best to worst (with ranking 1 being best, and ranking 8 being worst). The model associated with each response was not revealed to annotators, and the order of model responses was randomised.

3.7 LLM Evaluation

We adopt the LLM-as-judge paradigm to label the expanded evaluation dataset of 8,000 examples, comprising 32,000 model-generated responses [36]. Prior work has demonstrated the viability of using automatic LLM-generated annotations to scale evaluations in the domain of pedagogical explanations and feedback [7, 11, 12, 29].

Ensemble of Judges. Rather than relying on a single judge, we use a panel of three strong LLMs: GPT-4.1, Gemini-2.5-Flash, and Qwen3-32B. Verga et al. [34] demonstrate that ensembles composed of diverse model families can outperform individual large models, particularly by mitigating model-specific biases.

To annotate the expanded evaluation dataset, we prompt each of the three judges individually to provide binary decisions across all evaluation criteria. We adopt the Single Answer Grading approach proposed by Koutcheme et al. [11], which utilises an additional reasoning step to generate annotations. The judge first generates its own error explanation in the first conversation turn, then compares and evaluates the candidate model's explanation in the second turn. As a key modification, we remove the explanation structure constraints (described in Section 3.2) from the conversation history prior to the second turn, as the judges would inadvertently apply it to their reasoning and annotations.

From the ensemble, we obtain the final verdict using a strict unanimity policy: a criterion is marked correct only if all judges agree. While this method does not provide absolute performance guarantees, as discussed in Section 5.1, it offers a consistent, scalable, and reliable strategy for relative comparisons.

Annotations for Qwen3-32B were generated via the vLLM interface on a single A100 GPU, with the other two judges accessed through model specific proprietary APIs. Outputs were produced with a temperature of 0, reasoning disabled, and all other inference parameters set to default.

We validate IRR between LLM-judged and expert annotations across each metric using Gwet's AC1.

4 Results

4.1 Expert Response Rankings

Table 1 lists the proportion of responses where each SFT open-source model was ranked above the base and proprietary models.

Table 1: Win-Rate (0-1) of SFT Models Ranked Against Other Models (n = 100) on Compile-/Run-Time Errors. Each Cell: Compile-/Run-Time. Bold = Win-Rate > 0.5

	Loser				
Winner	DCC Help	GPT-4.1	Llama-8B	Qwen-32B	Qwen-4B
SFT-Qwen-4B	**0.68** / **0.78**	0.34 / 0.36	**0.68** / **0.78**	0.48 / 0.40	**0.70** / **0.58**
SFT-Llama-8B	**0.62** / **0.74**	0.36 / 0.44	**0.74** / **0.72**	0.36 / 0.44	**0.68** / **0.74**
SFT-Qwen-32B	**0.70** / **0.82**	0.36 / 0.46	**0.78** / **0.84**	0.46 / 0.46	**0.72** / **0.80**

Both SFT-Llama and SFT-Qwen-4B are preferred over their untrained counterparts, in both compile- and run-time cases. For Qwen-32B, the untrained variant is ranked higher slightly more often. SFT-Llama performs equally well on both compile-time (0.74) and run-time errors (0.72) compared to its base counterpart. In contrast, while the SFT-Qwen-4B is significantly preferred over its base counterpart for compile-time errors, it is only slightly favoured for run-time errors (about 58% of cases). All SFT models are also preferred to original DCC Help responses, especially for generating run-time error explanations.

Table 2: Expert Ranking Scores by Model (n = 100)

Model	Mean Rank	95% CI	First Place %	Last Place %
GPT-4.1	3.09	2.68-3.50	0.29	0.05
SFT-Qwen-32B	3.60	3.20-4.00	0.16	0.02
Qwen-32B	3.62	3.23-4.01	0.16	0.05
SFT-Qwen-4B	4.12	3.69-4.55	0.15	0.06
SFT-Llama-8B	4.27	3.83-4.71	0.15	0.07
Qwen-4B	5.50	5.12-5.88	0.01	0.18
DCC Help	5.67	5.24-6.10	0.06	0.30
Llama-8B	6.13	5.77-6.49	0.02	0.27

GPT-4.1 achieved the best mean rank score, followed by both variants of Qwen-32B. The SFT variants of both Llama and Qwen-4B demonstrated improved rank scores, first place and last place rates over their base counterparts. The base Qwen-4B, Llama and the DCC Help responses consistently ranked low, with first place rates of 0.01, 0.02 and 0.06 respectively.

4.2 Expert and LLM-Judged Metrics

Understanding Results. The results of both the expert annotations (800 model responses) and LLM-judged annotations (64,000 model responses) are presented side-by-side in Table 3. For the subset with annotations from all four experts, only one random annotation per example was included in the aggregate calculations, to ensure equal weighting.

Performance Comparisons. GPT-4.1 showed the strongest performance across the majority of metrics, as evidenced by both expert and LLM-judged evaluations. Specifically, GPT-4.1 achieved

Table 3: Criterion True-Rate (0–1) by Model. Each Cell: Expert/LLM-as-Judge. Gwet's AC1 is: Expert/Expert–LLM. "all" Columns Report the Rate of Model Responses Satisfying All Criteria. Bold = Highest Rate in Each Column.

Model	Correctness	Selectivity	Completeness	Clarity	Novice Ap.	No Solution	No Overhelp	Socratic	all (compile)	all (run-time)
					Base models					
DCC Help	0.86 / 0.82	0.71 / 0.64	0.68 / 0.42	0.35 / 0.77	0.69 / 0.80	0.74 / 0.64	0.70 / 0.56	0.50 / 0.09	0.04 / 0.03	0.02 / 0.05
GPT-4.1	**0.92** / **0.97**	**0.90** / **0.95**	**0.87** / 0.45	0.75 / **0.98**	**0.89** / **0.99**	0.82 / **0.97**	**0.95** / **0.97**	**0.65** / 0.38	**0.34** / 0.23	0.30 / 0.19
Llama-8B	0.67 / 0.48	0.45 / 0.31	0.58 / 0.21	0.58 / 0.65	0.61 / 0.68	0.65 / 0.68	0.73 / 0.52	0.45 / 0.07	0.06 / 0.03	0.10 / 0.01
Qwen-32B	**0.92** / 0.94	0.82 / 0.90	0.78 / **0.63**	0.69 / 0.97	0.88 / 0.98	0.81 / 0.94	0.90 / 0.92	0.62 / **0.44**	0.22 / **0.34**	0.28 / **0.26**
Qwen-4B	0.74 / 0.70	0.61 / 0.65	0.59 / 0.23	0.59 / 0.81	0.65 / 0.81	0.86 / 0.89	0.86 / 0.83	0.50 / 0.15	0.12 / 0.07	0.10 / 0.03
					SFT models					
Llama-8B	0.88 / 0.84	0.78 / 0.81	0.71 / 0.26	0.72 / 0.93	0.79 / 0.95	0.85 / 0.96	0.87 / 0.94	0.56 / 0.28	0.26 / 0.13	0.28 / 0.08
Qwen-32B	**0.92** / 0.94	0.85 / 0.93	0.79 / 0.38	0.75 / 0.97	0.85 / 0.98	**0.91** / **0.97**	0.90 / 0.96	0.60 / 0.36	0.26 / 0.19	**0.34** / 0.14
Qwen-4B	0.89 / 0.86	0.84 / 0.83	0.77 / 0.28	**0.76** / 0.93	0.82 / 0.95	0.84 / 0.96	0.92 / 0.94	**0.65** / 0.28	0.24 / 0.13	0.30 / 0.09
Gwet's AC1	0.84 / 0.75	0.56 / 0.57	0.56 / 0.05	0.20 / 0.46	0.62 / 0.71	0.50 / 0.74	0.72 / 0.71	-0.12 / 0.08		

the highest rates for Correctness, scoring 0.92 in expert and 0.97 in LLM-judged evaluation; Selectivity, scoring 0.90 in expert and 0.95 in LLM-judged evaluation; Novice Appropriate, scoring 0.89 in expert and 0.99 in LLM-judged evaluation; and No Overhelp, scoring 0.95 in expert and 0.97 in LLM-judged evaluation.

Base and SFT variants of Qwen-32B exhibited performance competitive with GPT-4.1 across many metrics. They matched GPT-4.1's performance in Correctness and No Solution, with the fine-tuned variant actually outperforming when judged by experts. For the remaining criteria, both Qwen-32B variants performed within a 0.10 or better margin of GPT-4.1.

While both base and SFT variants of Qwen-32B perform strongly overall, their relative performance differed between annotation strategies. Base Qwen-32B generally exhibited higher performance across LLM-judged results, achieving higher rates in metrics such as Completeness (0.63 for base versus 0.38 for SFT), and Socratic (0.44 for base versus 0.36 for SFT). Additionally, the base Qwen-32B variant showed the highest performance of any model in LLM-judged all at both compile- (0.34) and run-time (0.26). In contrast, SFT-Qwen-32B outperformed its base counterpart across several expert-judged metrics, including No Solution (0.91), and Clarity (0.75). It also achieved the highest performance in expert-judged all (run-time) with a rate of 0.34, and the second highest performance in expert-judged all (compile) with a rate of 0.26.

The SFT Llama variant demonstrated improvements over its base model, with increases of 0.11 in expert-judged Socratic, and 0.33 in expert-judged Selectivity. A more substantial gain of 0.50 was observed in LLM-judged Selectivity for the SFT variant over its base counterpart. Likewise, SFT-Qwen-4B showed improvements across all metrics compared to its base counterpart, with the exception of expert-judged No Solution. The most significant gains for SFT-Qwen-4B were observed in expert-judged Socratic, with a rate of 0.65, and Clarity, with a rate of 0.76.

When compared to the baseline DCC Help responses, all SFT models showed improvement in every metric except for LLM-judged Completeness. The most substantial gains were specifically noted in expert-judged Clarity and in LLM-judged No Overhelp.

Expert Inter-Rater Reliability. IRR among the four expert annotators, as measured by Gwet AC1 scores, indicates varying levels of agreement across metrics, with almost perfect agreement for Correctness ($AC1$ = 0.84); substantial agreement for No Overhelp ($AC1$ = 0.72) and Novice Appropriate ($AC1$ = 0.62); moderate agreement for Selectivity ($AC1$ = 0.56) and Completeness ($AC1$ = 0.56); slight agreement for Clarity ($AC1$ = 0.29); and systematic disagreement for Socratic ($AC1$ = −0.12).

LLM-judge vs Expert IRR. IRR across the expert annotations and the LLM-judged annotations, using Gwet AC1 scores, indicate substantial agreement for Correctness ($AC1$ = 0.75), No solution ($AC1$ = 0.74), No Overhelp ($AC1$ = 0.71) and Novice Appropriate ($AC1$ = 0.71); moderate agreement for Selectivity ($AC1$ = 0.57) and Clarity ($AC1$ = 0.46); and slight agreement for Socratic ($AC1$ = 0.08) and Completeness ($AC1$ = 0.05).

5 Discussion

Our primary contribution is that SFT improves the capability of smaller, open-source language models to generate C programming error explanations for CS1/2 students. The smaller SFT models showed significant improvements over their base counterparts in both expert and LLM-judged metrics, including Correctness (Qwen-4B ∼20%; Llama ∼31%), Selectivity (Qwen-4B ∼28%; Llama ∼73%), Completeness (Qwen-4B ∼20%; Llama ∼22%), Novice Appropriate (Qwen-4B ∼17%; Llama ∼29%), and Socratic (Qwen-4B ∼30%; Llama ∼24%). This is corroborated by the expert rankings, where the small SFT models were preferred over their base counterparts in 58%–74% of cases, across both compile- and run-time errors, and achieved ≥ 20% improved mean rank scores. Overall, these findings validate the effectiveness of SFT as a strategy for enhancing the pedagogical value of small, open-source models.

Additionally, each of the fine-tuned models exceed the performance of DCC Help on almost all metrics, demonstrating gains of ≥20% in Clarity and No Overhelp, and with responses ranked higher in 68%–82% of cases. The sole exception was the LLM-judged Completeness metric, where DCC Help outperformed each of the SFT models. However, the comparatively low LLM-judged rates, along with very weak expert–LLM agreement ($AC1$ = 0.05), suggest a methodological problem with the LLM annotation process for Completeness, rather than an actual performance shortfall. Ultimately, all three of the fine-tuned open-source models surpass the baseline standard set by the original DCC Help responses, and so could already be adopted as viable replacements.

When compared to GPT-4.1, SFT-Qwen-4B performs within a margin of 0.10 in all expert-judged metrics, and achieves a mean ranks score worse by only 1.03. This result is competitive and valuable considering the small parameter size of Qwen-4B. The LLM-judged annotations show a larger gap in performance from GPT-4.1 to SFT-Qwen-4B in metrics such as Selectivity and Completeness, though this may be affected by the bias of GPT-4.1 as a judge. Regardless, both LLM and expert annotations demonstrate the competitive performance of SFT-Qwen-4B in Clarity, Novice Appropriate, No Overhelp and No Solution, when compared to the much larger models.

While Qwen-32B displays no definitive improvement from the fine-tuning process, both the base and SFT variants come close in performance to GPT-4.1 across expert/LLM-judged metrics. This is a valuable finding as it suggests that Qwen-32B can serve as a suitable replacement for GPT-4.1 when generating the training dataset. This would allow institutions to ensure data privacy during the development and training of specialised open-source models, including models which rely on sensitive data.

5.1 Limitations

Our SFT training dataset was generated using GPT-4.1, therefore our results are inherently bounded by the quality, style and biases of GPT-4.1. This also impacts the interpretability of LLM-judged results which used GPT-4.1 as part of the judge ensemble. The SFT process optimises smaller models to mimic the style and structure of GPT-4.1, consequently using GPT-4.1 as a judge may have favoured responses similar to its own output rather than rewarding pedagogical quality. We sought to mitigate this self-bias by employing an ensemble of diverse models for judgement and validating against a human expert evaluation.

Similarly, as Qwen-32B and GPT-4.1 were represented in the judging ensemble and as candidate models, the LLM-judged evaluation is vulnerable to self-bias. While we sought to mitigate this by adding a third impartial model (Gemini-2.5-Flash) to the ensemble, and enforcing a unanimous-verdict policy, the markedly higher LLM-judged performance for GPT-4.1 and base Qwen-32B suggests that self-bias persisted. As a result, their relative performance in LLM-judged metrics may be overstated.

The low AC1 scores for Socratic and Clarity, in both expert and expert–LLM agreement, reduce the reliability of conclusions drawn from these metrics, and they may require refinement. We also observed notably low expert–LLM agreement for Completeness, despite moderate expert–expert agreement. This indicates potential discrepancies between the expert and LLM-judged interpretations of the Completeness criterion. The expert judges were briefed on the intended definitions of each criterion, which may have provided a more nuanced understanding than that afforded by the textual descriptions alone.

The training and evaluation dataset included sources from real student errors in CS1/2 courses at a single institution. It may not capture the full spectrum of errors encountered in different contexts, and the risk of over-fitting is present. Further validation before large scale application is required, for example, in other C courses.

Finally, the four experts had no knowledge of the set of models being evaluated, and the order of responses was randomised for each example. The original DCC Help responses likely stood out as the only ones not conforming to the common response structure, potentially affecting their evaluation.

5.2 Future Work

Our findings confirm the viability of SFT for model specialisation, and present several promising avenues for future research:

- **Human-preference fine-tuning**: Limited gains in larger models such as Qwen3-32B suggest a potential ceiling for SFT techniques. Exploration into human alignment techniques such as Direct Preference Optimisation (DFO) or Reinforcement Learning from Human Feedback (RLHF) could further refine model behaviour.
- **In-situ evaluation**: In-classroom studies to assess the real-world impact of these fine-tuned models are crucial to produce empirical evidence of learning outcome achievement.
- **On-device model deployment**: Significant gains in the performance of smaller models found in this study highlight the opportunity to explore on-device model deployment, such as in-browser or integrated into a local development environment. Such a deployment would provide a clear solution to ongoing privacy, cost and scalability concerns.

6 Conclusion

We investigated the efficacy of SFT with open-source language models on a generated dataset to explain C compiler errors for novice programmers across a range of real student compile- and run-time errors from a large CS1/2 programming cohort. Our findings demonstrate that SFT is a viable and effective strategy for specialising smaller models for pedagogical applications. We observed substantial improvements across a range of quality criteria, most notably clarity, selectivity, and pedagogical appropriateness of generated explanations in smaller models.

We found that SFT is most effective at narrowing the gap between smaller, economical models like Llama-3.1-8B and large, frontier models like GPT-4.1. In all cases, the fine-tuned models outperformed existing tools deployed to the CS1/2 cohort.

This research provides a replicable methodology for creating specialised AI-driven educational tools that are not only cost-effective, but also mitigate the privacy and data security concerns associated with commercial frontier models. By demonstrating a practical pathway to developing high-quality, on-premises or even on-device models, this work empowers educators to leverage advanced AI capability to support student learning.

Acknowledgments

This work was supported by the Research Council of Finland grant #356114. We would also like to thank Google.org who supported this work in the form of a 2023 Google Award for Inclusion Research, and the NVIDIA Corporation for providing access to the GPU hardware used in this work.

References

[1] Nischal Ashok K. and Andrew Lan. 2024. Improving Socratic Question Generation using Data Augmentation and Preference Optimization. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*. Association for Computational Linguistics, Mexico City, Mexico, 108–118.

[2] Paul Denny, James Prather, and Brett A. Becker. 2021. On designing programming error messages for novices: Readability and its constituent factors. In *Conference on Human Factors in Computing Systems - Proceedings*. Association for Computing Machinery, 1–15. doi:10.1145/3411764.3445696

[3] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (1 2024), 56–67. doi:10.1145/3624720

[4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLORA: Efficient Finetuning of Quantized LLMs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, 441.

[5] Kilem Li Gwet. 2008. Computing inter-rater reliability and its variance in the presence of high agreement. *Brit. J. Math. Statist. Psych.* 61, 1 (5 2008), 29–48. doi:10.1348/000711006X126600

[6] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. In *ICER 2023 - Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*. Association for Computing Machinery, Inc, 93–105. doi:10.1145/3568813.3600139

[7] Yann Hicke, Anmol Agarwal, Qianou Ma, and Paul Denny. 2023. AI-TA: Towards an Intelligent Question-Answer Teaching Assistant using Open-Source LLMs. arXiv:2311.02775 [cs.LG]

[8] Lan Huang. 2023. Ethics of Artificial Intelligence in Education: Student Privacy and Data Protection. *Science Insights Education Frontiers* 16, 2 (6 2023), 2577–2587. doi:10.15354/sief.23.re202

[9] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. 2023. Exploring the Potential of Large Language Models to Generate Formative Programming Feedback. In *2023 IEEE Frontiers in Education Conference (FIE)*. 1–5. doi:10.1109/FIE58773.2023.10343457

[10] Nachiket Kotalwar, Alkis Gotovos, and Adish Singla. 2024. Hints-In-Browser: Benchmarking Language Models for Programming Feedback Generation. *38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks*. (2024).

[11] Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, Syed Ashraf, and Paul Denny. 2025. Evaluating Language Models for Generating and Judging Programming Feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 624–630. doi:10.1145/3641554.3701791

[12] Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, and Paul Denny. 2024. Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-As-A-Judge. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education, Volume 1* (Milan, Italy) *(ITICSE '24)*. doi:10.1145/3649217.3653612

[13] Juho Leinonen, Paul Denny, Stephen Macneil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Vol. 1. Association for Computing Machinery, 124–130. doi:10.1145/3587102.3588785

[14] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 563–569. doi:10.1145/3545945.3569770

[15] Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. In *23rd Koli Calling International Conference on Computing Education Research*. Association for Computing Machinery, 1–11. doi:10.1145/3631802.3631830

[16] Rongxin Liu, Julianna Zhao, Benjamin Xu, Christopher Perez, Yuliia Zhukovets, and David J. Malan. 2025. Improving AI in CS50 Leveraging Human Feedback for Better Learning. In *SIGCSE TS 2025 - Proceedings of the 56th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 715–721. doi:10.1145/3641554.3701945

[17] Evanfiya Logacheva, Arto Hellas, James Prather, Sami Sarsa, and Juho Leinonen. 2024. Evaluating Contextually Personalized Programming Exercises Created with Generative AI (pp. 95-113).. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1 (ICER, 2024)*, Vol. 1. ACM, 95–113. doi:10.1145/3632620.3671103

[18] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2022. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *54th ACM Technical Symposium on Computer Science Education*. ACM, 931–937. http://arxiv.org/abs/2211.02265

[19] Marie Hélène Nienaltowski, Michela Pedroni, and Bertrand Meyer. 2008. Compiler error messages: What can help novices? *SIGCSE'08 - Proceedings of the 39th ACM Technical Symposium on Computer Science Education* (2008), 168–172. doi:10.1145/1352135.1352192

[20] Maciej Pankiewicz and Ryan S Baker. 2023. Large Language Models (GPT) for automating feedback on programming assignments. *arXiv:2307.00150* (2023).

[21] Tung Phung, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models. (4 2023). http://arxiv.org/abs/2302.04662

[22] James Prather, Juho Leinonen, Natalie Kiesler, Jamie Gorson Benario, Sam Lau, Stephen MacNeil, Narges Norouzi, et al. 2025. Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools. In *2024 Working Group Reports on Innovation and Technology in Computer Science Education* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 300–338. doi:10.1145/3689187.3709614

[23] Jake Renzella, Alexandra Vassar, Lorenzo Lee Solano, and Andrew Taylor. 2025. Compiler-Integrated, Conversational AI for Debugging CS1 Programs. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V.1 (SIGCSETS 2025)*, Vol. 2022-March. Association for Computing Machinery, 994–1000. doi:10.1145/3641554.3701827

[24] Emily Ross, Yuval Kansal, Jake Renzella, Alexandra Vassar, and Andrew Taylor. 2025. Supervised Fine-Tuning LLMs to Behave as Pedagogical Agents in Programming Education. (2 2025). https://arxiv.org/abs/2502.20527v1

[25] Audrey Salmon, Katie Hammer, Eddie Antonio Santos, and Brett A. Becker. 2025. Debugging Without Error Messages: How LLM Prompting Strategy Affects Programming Error Explanation Effectiveness. doi:10.1145/arXiv.2501.05706

[26] Eddie Antonio Santos, Brett A Becker, and Brett A 2024 Becker. 2024. Not the Silver Bullet: LLM-enhanced Programming Error Messages are Ineffective in Practice. *Proceedings of the 2024 Conference on United Kingdom & Ireland Computing Education Research* (11 2024). doi:10.1145/3689535

[27] Eddie Antonio Santos, Prajish Prasad, and Brett A. Becker. 2023. Always Provide Context: The Effects of Code Context on Programming Error Message Enhancement. *CompEd 2023 - Proceedings of the ACM Conference on Global Computing Education* 1 (12 2023), 147–153. doi:10.1145/3576882.3617909

[28] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research*, Vol. 1. Association for Computing Machinery, Inc, 27–43. doi:10.1145/3501385.3543957

[29] Alexander Scarlatos, Digory Smith, Simon Woodhead, and Andrew Lan. 2024. Improving the Validity of Automatically Generated Feedback via Reinforcement Learning. *Artificial Intelligence in Education: 25th International Conference (AIED 2024)* (2024), 280–294. doi:10.1007/978-3-031-64302-6{_}20

[30] Andrew Taylor, Alexandra Vassar, Jake Renzella, and Hammond Pearce. 2024. dcc - Help: Transforming the Role of the Compiler by Generating Context-Aware Error Explanations with Large Language Models. In *SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 1314–1320. doi:10.1145/3626252.3630822

[31] Llama team. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[32] Qwen Team. 2025. Qwen3 Technical Report. arXiv:2505.09388 [cs.CL] https://arxiv.org/abs/2505.09388

[33] Andrew Tran, Kenneth Angelikas, Egi Rama, Chiku Okechukwu, David H. Smith, and Stephen MacNeil. 2023. Generating Multiple Choice Questions for Computing Courses Using Large Language Models. In *2023 IEEE Frontiers in Education Conference (FIE)*. 1–8. doi:10.1109/FIE58773.2023.10342898

[34] Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. 2024. Replacing Judges with Juries: Evaluating LLM Generations with a Panel of Diverse Models. arXiv:2404.18796 [cs.CL] https://arxiv.org/abs/2404.18796

[35] Sierra Wang, John Mitchell, and Chris Piech. 2024. A Large Scale RCT on Effective Error Messages in CS1. In *SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 1395–1401. doi:10.1145/3626252.3630764

[36] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and Chatbot Arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2020, 29 pages.