Beyond Unified Access: How Documentation and Instructional Context Shape Student LLM Use in Multi-Model APIs

Hiba Eltigani
Tufts University

Rukhshan Haroon
Tufts University

Abdullah Bin Faisal
Tufts University

Fahad R. Dogar
Tufts University

Abstract

Advances in generative AI tools have motivated a transformation of the software development industry toward an AI-centered paradigm. Computer science educators bear the responsibility for preparing students with the competencies required to meet industry needs. Yet little is known about how students navigate choices among competing Large Language Models (LLMs) when given access to multiple models. Our analysis of log data collected across three CS courses highlights students' reliance on example values from documentation and uncovers various patterns of model exploration behaviors. It shows that instructional design shapes the extent of that exploration. We argue for contextual, academic-focused abstractions that incorporate instructional design and support scaffolding.

CCS Concepts

• **Social and professional topics → Computing education**; • **Human-centered computing → Human computer interaction (HCI)**.

Keywords

LLMs, Generative AI, CS Education

1 Introduction

Large Language Models (LLMs) are reshaping software development through code generation [6], debugging support [7], and natural language capabilities [2, 10, 12]. As a result, experience with LLM-based tools and their integration into software applications is increasingly important for graduates preparing for industry and an AI-centered future [2, 14, 15, 17, 26].

Consequently, practitioners in Computer Science Education (CSE) are exploring approaches to incorporate the teaching of these skills into software engineering curricula [6, 17], as well as into more general courses. However, instructors are often faced with various challenges [28], which are further heightened by the rapid proliferation of LLMs. At the same time, instructors must remain mindful of the impact that integrating these models may have on their courses' core learning goals.

Existing solutions either rely on commercial LLM platforms that are often unsuitable for educational settings [9]; or use highly customized systems [2, 29] that lacks adaptability across diverse contexts and use cases. This raises the question of how to design systems that provide both flexibility and pedagogical support.

In this experience report, we examine the classroom use of a middleware system that supports access to multiple LLMs across three courses at a medium-sized university in the United States. Using log data, we analyze how middleware systems can support model exploration, and discuss implications for students' conceptual understanding.

Our findings underscore the impact of instructional context on student interactions with LLMs, while also showing limits in students' parameter experimentation. Students often relied on example values from documentation; however, assignments that mandated exploration or involved open-ended problems were associated with more diverse model use and switching behavior. Within open-ended tasks, model switching served distinct functions, including systematic benchmarking, application-driven model specialization, and late-stage comparison.

2 Related Work

Large Language Models (LLMs) can support a range of applications that rely on natural language processing [2]. They can provide advanced features such as converting textual descriptions into visual interface components [10] and interpreting the emotional tone of text messages to improve communication accessibility [12].

In parallel, researchers and educators have increasingly explored the use of LLMs in computer science education. Prior work has shown that LLMs can effectively support learning activities such as grading [18], explaining complex material [25], and generating customized educational resources [11, 13, 16, 27], thereby reducing instructor workload and time commitment while enhancing student support [4, 24].

As LLMs become integral to modern software development, students are expected to develop familiarity with these tools and learn how to use and integrate them into software applications, a skill highlighted as essential for meeting industry expectations and preparing for an AI-driven future [2, 14, 15, 17]. In response, some educators have begun redesigning software engineering courses to explicitly teach the integration of LLMs [6, 17].

However, adopting LLM-based activities beyond specialized courses remains challenging [28]. Educators often encounter difficulties stemming from the complexity of LLM APIs, which is exacerbated by the variety of available open-source and commercial systems. Additional challenges include the effort needed to evaluate model suitability for a specific use case, manage pricing and budgeting, and determine the time investment needed to implement support for techniques like retrieval-augmented generation to enhance model output and combat hallucination and accuracy issues [2, 26, 28].

Prior work has largely examined specialized LLM applications, such as API-code generation for interface design [2] or multi-agent systems for software-development learning [29]. Middleware systems offer a broader abstraction layer for simplifying LLM API complexity and unifying access to multiple models [5, 22, 23], but their role in supporting student learning across diverse educational contexts remains underexplored.

In this work, we examine the classroom use of an LLM middleware system to understand how abstraction layers can support model integration across diverse educational use cases.

3 Methods

```
call(model='4o-mini',
  system="Explain like I'm five",
  query="What is computing?",
  temperature=0.0, lastk=0,
  session_id='GenericSession',
  rag_usage=True,
  rag_threshold=0.5, rag_k=1)

add(path="path/to/document.pdf",
  strategy='smart',
  description='details about ABC',
  session_id='GenericSession')

retrieve(query="Tell me about ACM?",
  session_id='GenericSession',
  rag_threshold=0.5, rag_k=1)
```

Figure 1: Example calls for Main uniLLM's methods.

3.1 Study Tools

uniLLM is an open-source, free-to-use, research-based proxy system that provides an abstraction layer for unified access to multiple LLMs, similar to other systems like LangChain [8] and LiteLLM [5]. The system is accessible via a RESTful API and exposes common core parameters typically used to control LLM outputs (e.g. temperature). By abstracting model-specific details, uniLLM allows new models to be added and older ones to be deprecated without changing the system interface. It also supports cost management through API-key-based usage plans and quotas, enabling request limits and model access to be managed at the user or group level. In addition to text-based interactions, uniLLM implements retrieval-augmented generation (RAG) to enhance model outputs and support context management [9, 28]. In this study, uniLLM was deployed on Amazon Web Services (AWS) [3] using a serverless architecture based on Lambda functions. uniLLM exposes three primary methods (Figure 1). Requests with missing or malformed parameters automatically default to pre-configured values.

- **`call.`** Submits a `query` and a `system` prompt to a selected LLM using the `model` parameter and returns the model response. The `session_id` parameter maintains conversational state across requests, while addional parameters such as `temperature`, `lastk`, `rag_threshold`, and `rag_k` support customized interactions.
- **`add.`** Stores text or PDF resources for reuse across sessions, with configurable chunking through the `strategy` parameter and optional metadata for retrieval.
- **`retrieve.`** Retrieves relevant stored-resource chunks for a `query`, enabling users to inspect context selection and reuse chunks across sessions.

Key Management and Usage Quotas. All uniLLM activity is managed through individual API keys that track usage, enforce quotas, and control access. Limits include maximum request rate, daily request counts, concurrent requests, file size, and execution time.

3.2 Study Context

Approximately 130 students at a medium-sized private university in the United States used uniLLM across Spring and Fall semesters over two years. Students were provided with documentation and example-based guides. Across courses, students had access to multiple large language models, including OpenAI's "GPT-4o mini" [1], Anthropic's "Claude 3 Haiku" [20], Microsoft's "Phi-3" [21], and Meta's "Llama 3" models [19]. uniLLM was used in three advanced-level computer science courses available to both graduate and undergraduate students:

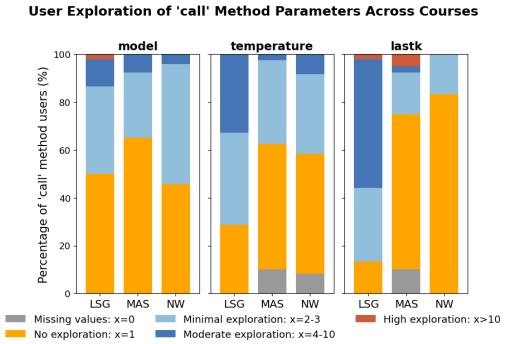
- **LLMs for Social Good (LSG).** Focuses on LLM-based chatbot applications addressing social challenges faced by marginalized communities. After individual assignments on prompt engineering and RAG, students developed group projects using techniques such as agentic workflows and human-in-the-loop design. Applications spanned academic support, civic participation, and health and well-being.
- **Multi-Agent Systems (MAS).** Covers multi-agent system theory and implementation for students with prior AI coursework. In the final project, students developed hybrid systems combining traditional and LLM-based agents across domains such as path-finding, voting, auctions, and game theory.
- **Networks (NW).** Focuses on computer communication networks and the Internet Protocol Suite for students with advanced programming experience. In the final group project, students built an LLM-powered HTTPS proxy with enhancements such as accessibility support, content filtering, text summarization, and reduced page load times.

3.3 Data Collection and Analysis

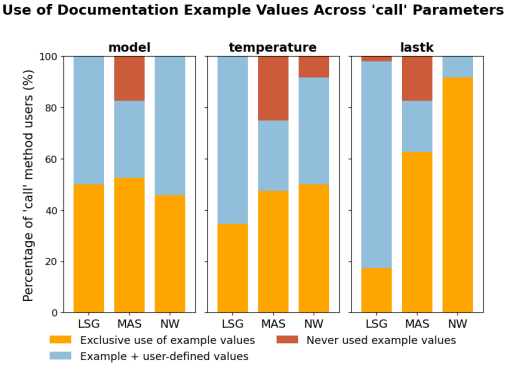
This study was approved by our Institutional Review Board (IRB). We collected API request log data and quantitatively analyzed REST request details as identified by the source API key. We treated requests as well-formed regardless of whether they were executed successfully. Throughout this paper, we use "user" to refer to a distinct API key. Depending on the course configuration, an API key may correspond to either an individual student or a shared student group.

Table 1: Usage by Course

	LSG (2 cohorts)	MAS (2 cohorts)	NW (1 cohort)
Avg. unique models/user	2.8	2.2	2.1
Avg. requests/user	1,789	1,345	306
Top 10% users' request share	61%	39%	31%
P50 requests/user	466	684	238



(a) Exploration intensity across `call` method parameters.



(b) Documentation-values usage across `call` method parameters.

Figure 2: Distribution of user behavior across `call` method parameters.

4 Results

4.1 Usage Across Courses

On average, users submitted approximately 759 requests per day. At the request level, the `call` method was the most frequently used, accounting for 87% of all requests, followed by `add` and `retrieve`, which accounted for 8% and 4%, respectively. A similar pattern emerged at the user level. Nearly all users (99%) used the `call` method, and approximately 46% used it exclusively. In contrast, no users relied exclusively on the `add` or `retrieve` methods. Instead, 50% and 29% of users used `add` and `retrieve`, respectively, in conjunction with `call`. Overall, 26% of users used all three methods.

Approximately 61% of users enabled RAG in at least one `call` request, corresponding to 12% of all `call` requests. However, not all RAG-related activity was preceded by document storage. Around 21.9% of users who enabled RAG had never stored documents using `add`. Similarly, 8.8% of users who issued `retrieve` requests had not stored documents themselves. These cases accounted for a small share of activity: 6.7% of RAG-enabled `call` requests and 0.1% of all `retrieve` requests, respectively.

Among document-storing requests using the `add` method, PDFs were the dominant format, accounting for 75.6% of such requests. Document formats, however, varied across courses. Students in LSG relied predominantly on PDFs, which represented 82.1% of their document-storing requests. In contrast, students in NW and MAS relied almost entirely on text files, which accounted for 99.1% and 96.0% of their document-storing requests, respectively. These differences were consistent with the distinct application contexts of the courses: NW and MAS projects frequently operated on text-based application content, whereas LSG projects commonly integrated predefined documents into conversational applications.

Courses also differed in the extent to which activity was concentrated among heavy users (Table 1). However, differences in the duration of uniLLM use, assignment structure, and API-key granularity make direct comparisons of activity levels difficult.

4.2 Documentation-Driven Model and Parameter Use

To understand users' exploration of uniLLM parameters, we analyzed parameter usage within the `call` method, which accounted for the majority of API requests. Because identifying appropriate parameter values is a common challenge when incorporating LLMs into software applications [9], we treat variation in parameter values as a proxy for experimentation.

Figure 2 shows how users engaged with the three primary call parameters (`model`, `temperature`, `lastk`) across courses. Overall, a substantial share of users relied on a single value per parameter and used only example values from the uniLLM documentation, indicating limited experimentation. However, the degree of exploration varied markedly by course. Students in LSG explored `temperature` and `lastk` far more than those in MAS or NW; for example, over half of LSG users varied `lastk` across four or more values, whereas roughly 83% of NW users never varied it. We separately analyzed `session_id`, which controls conversation history and therefore naturally exhibited greater variation (median 10.5 unique values, maximum 3,244). Even so, 16.9% of users used exclusively documentation example values for `session_id` and 22.0% relied on a single value. Considering all parameters together, including `session_id`, 8% of users retained a single fixed configuration throughout their usage, and 5% used only values from the documentation examples.

Model use showed a similar concentration. "GPT-4o mini," the model used in the documentation examples, dominated usage, accounting for 56.4% of all `call` requests, followed by "Claude 3 Haiku" at 19.0%. Requests with empty or malformed model values were rare, representing only 0.3% of `call` requests. Together, these results show that the model and parameter values emphasized in

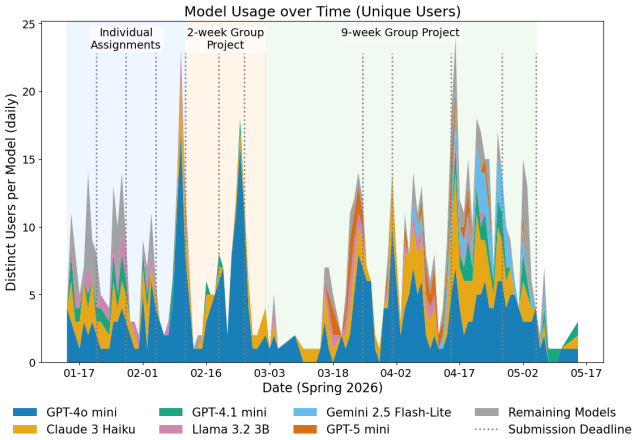


Figure 3: Unique users per model over time across the LSG course term.

the documentation were also among those most frequently used by students.

4.3 Model Exploration Across Instructional Activities

To examine how model use varied across instructional activities, we focus on the Spring 2026 cohort of the LSG course, which included 25 students with access to 17 models. We first examine model use across course activities and then analyze model-switching behavior during the nine-week group project.

Across the course, all users selected "GPT-4o mini," and six users used it exclusively. Requests to "GPT-4o mini" accounted for 45% of total `call` requests. However, its share varied substantially across course activities (Figure 3). Its use was lowest during the first assignment, which explicitly required students to experiment with multiple models. During this assignment, "GPT-4o mini" accounted for 29.8% of requests, while three other models each accounted for more than 10%.

"GPT-4o mini" use was also relatively low during the nine-week group project, accounting for 31.6% of requests. During this period, two additional models each accounted for more than 10% of requests. Students developed applications for loosely defined problems, such as democratizing access to civic knowledge and providing reading support for people with ADHD. In contrast, "GPT-4o mini" accounted for 50–90% of requests in more structured assignments. Thus, broader model use coincided with activities that either explicitly required comparison or involved open-ended application development.

During the group project period, 14 users issued `call` requests, including one user who used only "GPT-4o mini." Because the projects were conducted in groups of three to four students and involved activities beyond API use, such as literature reviews and user studies, not all students actively used uniLLM during this period. The remaining 13 users issued approximately 31,000 requests and used between 2 and 12 models.

We define a switching event as a `call` request in which a user selected a different model from the model selected in their immediately preceding `call` request. Using this definition, we identified approximately 5,800 switching events during the group project period. Most switching events (99.4%) involved returning to a model the user had previously selected, indicating that switching primarily involved repeated movement among familiar models rather than the continued introduction of new models.

We further define toggling as repeated alternation between the same two models. Toggling accounted for 16.0% of switching events. The longest consecutive toggle sequence contained 34 switches, and the most common toggle pair was "GPT-4o mini" and "Claude 3 Haiku," which accounted for 50.9% of toggle events. Aggregate switching counts, however, obscured substantial differences in how switching occurred. We therefore examined chronological request sequences to identify illustrative patterns.

4.4 Illustrative Patterns of Model Switching

The following cases illustrate three distinct patterns observed in the LSG course's nine-week project data: sustained systematic benchmarking, application-driven model specialization, and late-stage comparative evaluation. These cases were selected to illustrate differences in the timing, structure, and apparent function of switching rather than to estimate the prevalence of each pattern.

Across these cases, we also observed null values for parameters such as `lastk` and `temperature`. uniLLM silently replaces such values with preconfigured defaults; for example, a null `lastk` value is replaced with a value of 100. Null values appeared at different points in users' request sequences, but the logs do not indicate why users alternated between explicitly supplying a value and leaving it null.

4.4.1 Sustained Systematic Benchmarking. User-17 was already among the most active users before the final project, generating 1,492 `call` requests. However, their pre-project model use was narrow: they had used only "GPT-4.1 mini" and "GPT-4o mini" and had generated only one switching event. During the project, User-17 generated the highest number of `call` requests and switching events, accounting for 26.4% of all `call` requests and 76.0% of all switching events in that period. Their activity expanded to include six additional models.

Starting in the first week of the project, their switching followed a systematic benchmarking workflow involving repeated comparisons across model sets, datasets, and prompt formats. The user initially conducted a pairwise comparison between "GPT-4o mini" and "LLaMA 3 3B." They submitted the same `query` and `system` values to both models while creating a new `session_id` for each request, thereby isolating each request from prior conversation history. This initial comparison used a five-question dataset focused on nutrition and health advice, including questions such as *"I live in Sudan. What is a cheap and healthy meal I can make for dinner?..."* and *"I was going to feed my 10 month old some honey. That should be fine because it is a liquid, right?"* Across these questions, the user used a consistent system prompt positioning the model as an expert nutritionist specializing in developing regions.

The user subsequently expanded the benchmark from two to four models by adding "Claude 3 Haiku" and "Phi 3," and reran the

same five-question dataset across all four models using the same system prompt. On a later day, they benchmarked the same models using a new ten-question dataset. Unlike the earlier benchmark, this dataset varied the `system` value according to the question type. Multiple-choice questions used a system prompt instructing the model to choose an option: *"... These are multiple response questions. Only answer with a, b, c, d, e, f..."* Open-response questions instead used a prompt asking for a specific tone from the perspective of a nutrition expert: *"... This is an open response question. Write a friendly, informational, accurate answer..."*

Later, the user replaced two of the four models and benchmarked the updated model set on an expanded 136-question dataset. This dataset included questions from the earlier evaluations as well as new evaluation tasks. In these tasks, prior model responses were included as answer options, and the model was asked to judge which response was better: *"You are an expert nutritionist judging answers to helath[health] questions. Answer that you like 1 or 2 better where 1 is the first option and 2 is the second option and provide your reasoning..."* These judging tasks were consistent with the LLM-as-a-Judge framework [30] covered in the course.

Throughout these evaluations, the user continued to create a new `session_id` for each question and paired each question with the corresponding `system` value. In the initial runs, each question was submitted to all models before the user moved to the next question. In a later evaluation using 70 questions from the 136-question dataset, the user instead submitted batches of five questions to each model before switching. Thus, the benchmarking workflow evolved in both the scale of the datasets and the batching strategy while retaining controlled comparisons across models. After these benchmark runs, switching sequences were associated with unique queries and stable `session_id` values within application tasks, distinguishing them from the earlier controlled comparisons.

4.4.2 Application-Driven Model Specialization. User-13 generated 1,640 `call` requests before the final project and was the most active user in two earlier assignments. Before the project, they had generated five switching events and used four models. During the project, their number of switching events increased approximately 25-fold, although their model use was limited to "GPT-4o mini," "Claude 3 Haiku," and "Gemini 2.5 Flash-Lite." Their project was a civic helpdesk assistant designed to help users find options for wildfire-recovery assistance.

Switching began in week 6 and was characterized by a close association among `model`, `query`, and `system` values. The user generally retained the same `session_id` across sequences of requests, while particular system prompts were consistently paired with particular models. For example, a prompt about webpage-content extraction was always paired with "Gemini 2.5 Flash-Lite," a model with web access.

Another system prompt, used 197 times, supported an output-validation step. In this workflow, the prompt required strict rule-following: *"...You will be shown explicitly what the forward cell contains. Trust that field over your own understanding of the grid if there is any disagreement. Reply with exactly one digit (0 or 1)..."* This prompt was consistently paired with "Claude 3 Haiku." In contrast, prompts involving conversational tone and flexible judgment were paired with "GPT-4o mini." For example, one prompt instructed the model to *"...Sound approachable and helpful without being casual ... Encourage users not to share sensitive personal details unless absolutely necessary..."*

The query sequences also reflected repeated application testing. For example, *"HTML: UserInput: Summarize the content of this webpage in 4 sentences. Mention the main topic, key words..."* was issued 139 times. Queries also formed multi-step sequences that used shared conversation history. For example, a query containing raw HTML was followed by a query asking the model to *"...return all the URLs that start with http and relate to news articles..."* in a specified list format. User-13's model choices remained closely coupled to distinct functions within the application, reflecting application-driven specialization rather than controlled cross-model comparison.

4.4.3 Late-Stage Comparative Evaluation. User-22 explored the highest number of models during the project period, introducing 11 new models and generating 124 switching events. Before the project, they had used only "GPT-4o mini" and generated 630 `call` requests. Their first switching event occurred in week 3, when they selected "GPT-5 mini" before immediately returning to "GPT-4o mini." All remaining switching activity occurred during the final two days of the project, when the user introduced the remaining ten models.

During this period, the user evaluated models in the "Llama" family using a 140-question dataset and compared eight additional models using a 41-question dataset. User-22's evaluation was concentrated at the end of the project, whereas User-17's comparisons unfolded across multiple project weeks. User-22 also adopted a model-level batching strategy, submitting the complete set of questions to one model before moving to the next. The questions were situational and related to the project application, indicating that the comparison evaluated models on project-relevant scenarios.

5 Discussion

Our findings suggest that unified access alone is insufficient to support deliberate model selection. Students' model and configuration choices were associated with both documentation examples and assignment structure, while model switching reflected several distinct practices. These results motivate pedagogically aware abstraction layers that make such choices more visible and responsive to instructional context.

Documentation as an Implicit Default. Across courses, many users relied on the model and parameter values emphasized in the documentation. "GPT-4o mini," which appeared in the example calls, accounted for the majority of model requests overall, while a substantial share of users relied exclusively on example values for key parameters. These patterns suggest that documentation examples may function not only as explanations of syntax, but also as implicit recommendations about which models and configurations are appropriate. This makes documentation an important part of the instructional design of educational APIs. Instructors could therefore provide contrasting examples that explain when different models or settings are appropriate. Middleware designers should clearly distinguish example values from system defaults, vary the models used across examples, and expose the resolved configuration used for each request, particularly when missing or null values are replaced automatically.

Assignment Design and Model Exploration. Model use varied across instructional activities. In the LSG course, the documented model accounted for a smaller share of requests during the assignment that explicitly required students to try multiple models and during the open-ended project period. In contrast, its use was substantially higher in more structured assignments. Although these observations do not establish that assignment design caused the difference, they suggest that students are more likely to explore alternatives when assignments make model choice visible and consequential.

This finding points to the importance of treating model selection as part of assignment design rather than leaving it as an optional implementation detail. Instructors could require an early low-stakes comparison, ask students to submit an intermediate model-selection rationale, and require them to revisit that choice after testing their application on representative tasks. Middleware systems could support this process by allowing instructors to define assignment-specific model sets, parameters, and comparison expectations, and by showing whether exploration occurred throughout development or only near the final deadline.

Model Switching and Systematic Benchmarking. The project-level analysis shows that model switching can reflect distinct practices, including sustained benchmarking, task-specific specialization, and late-stage comparison. These patterns have different educational meanings; therefore, simple measures such as the number of models used or switching events should not be equated with meaningful exploration or learning.

The contrast between sustained and late-stage comparison also suggests that systematic benchmarking should be taught explicitly. Students should learn to define task-relevant criteria, construct representative test sets, hold prompts and configurations constant where appropriate, account for stochastic variation, and consider trade-offs among output quality, latency, cost, and reliability. Instructors can support this through structured evaluation templates and intermediate checkpoints. Middleware designers can support it through benchmarking workflows that run a common test set across multiple models while recording outputs, resolved parameters, latency, token use, and cost.

Context-Aware Support for Learning. Taken together, these findings motivate abstraction layers that are aware not only of models and parameters, but also of instructional context. The same usage pattern may warrant different interpretations across assignments. Repeated use of one model may be appropriate in a constrained implementation exercise, but may warrant reflection in a project where model comparison is an explicit learning objective. Similarly, frequent switching may indicate systematic benchmarking, task-specific specialization, debugging, or deadline-driven compliance.

A pedagogically aware middleware system could combine instructor-defined goals with observable usage patterns. Instructors could specify the models, parameters, and comparison practices relevant to a particular assignment. The system could then offer targeted prompts when observed behavior may warrant reflection in light of those goals. For example, it might ask students to justify continued use of the documented model in a comparison assignment, warn when null parameters are replaced by defaults, or suggest running a controlled comparison before a project milestone. The same infrastructure could support instructors by surfacing patterns that may warrant follow-up. Dashboards could distinguish between interactive exploration, batch benchmarking, and application traffic; show when new models were introduced; and indicate whether comparison occurred early or only near submission.

The goal of such support is not to encourage switching for its own sake, but to help students make deliberate, task-sensitive, and evidence-based model-selection decisions.

5.1 Limitations and Future Work.

This analysis is based on log data, which limits our ability to infer students' motivations or conceptual understanding. Differences in course usage periods, assignment structures, and API-key granularity also constrain comparisons across courses. Moreover, logs do not capture off-system activity, such as planning, interpreting responses, discussing decisions with teammates, or contextualizing model outputs.

Future work could combine log analysis with interviews, surveys, project artifacts, and measures of learning outcomes. It could also evaluate whether visible defaults, benchmarking workflows, assignment-specific nudges, and instructor-facing dashboards lead to more systematic evaluation practices and better-justified model-selection decisions.

6 Conclusion

We analyzed usage data from a middleware system that enables students to incorporate LLMs into software applications. Our findings highlight students' reliance on fixed, documentation-driven configurations, show that instructional design shapes the extent of their exploration, and reveal distinct model-switching patterns—including systematic benchmarking, application-driven model specialization, and late-stage comparison.

Acknowledgments

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 Technical Report. https://arxiv.org/abs/2303.08774. arXiv preprint, submitted 15 March 2023.

[2] Timothy J. Aveni, James Smith, Armando Fox, and Björn Hartmann. 2025. Supporting Students in Prototyping AI-backed Software with Hosted Prompt Template APIs. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Nijmegen, Netherlands) *(ITiCSE 2025)*. Association for Computing Machinery, New York, NY, USA, 65–71. doi:10.1145/3724363.3729109

[3] AWS. 2024. Cloud Computing Services - Amazon Web Services (AWS). https://aws.amazon.com.

[4] Drew Bent, Kunal Handa, Esin Durmus, Alex Tamkin, Miles McCain, Stuart Ritchie, Ryan Donegan, Jennifer Martinez, and Jason Jones. 2025. Anthropic Education Report: How Educators Use Claude. https://www.anthropic.com/news/anthropic-education-report-how-educators-use-claude

[5] BerriAI. 2023. LiteLLM. https://github.com/BerriAI/litellm. Accessed: 2026-07-03.

[6] Uwe M. Borghoff, Mark Minas, and Jannis Schopp. 2025. Generative AI in Student Software Development Projects: A User Study on Experiences and Self-Assessment. In *Proceedings of the 6th European Conference on Software Engineering Education (ECSEE '25)*. Association for Computing Machinery, New York, NY, USA, 161–170. doi:10.1145/3723010.3723012

[7] Mohamed Boukhlif, Nassim Kharmoum, and Mohamed Hanine. 2024. LLMs for Intelligent Software Testing: A Comparative Study. In *Proceedings of the 7th International Conference on Networking, Intelligent Systems and Security* (Meknes, AA, Morocco) *(NISS '24)*. Association for Computing Machinery, New York, NY, USA, Article 42, 8 pages. doi:10.1145/3659677.3659749

[8] Harrison Chase. 2022. LangChain. https://github.com/langchain-ai/langchain

[9] Xiang Chen, Chaoyang Gao, Chunyang Chen, Guangbei Zhang, and Yong Liu. 2025. An Empirical Study on Challenges for LLM Application Developers. *ACM Trans. Softw. Eng. Methodol.* 34, 7, Article 205 (Aug. 2025), 37 pages. doi:10.1145/3715007

[10] Yoonseo Choi, Eun Jeong Kang, Seulgi Choi, Min Kyung Lee, and Juho Kim. 2025. Proxona: Supporting Creators' Sensemaking and Ideation with LLM-Powered Audience Personas. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 149, 32 pages. doi:10.1145/3706598.3714034

[11] Avid Fayaz, Richard Glassey, and Alexander Baltatzis. 2025. Generating Personalized Assignments with Students in the Loop. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Nijmegen, Netherlands) *(ITiCSE 2025)*. Association for Computing Machinery, New York, NY, USA, 305–311. doi:10.1145/3724363.3729070

[12] Rukhshan Haroon and Fahad Dogar. 2024. TwIPS: A Large Language Model Powered Texting Application to Simplify Conversational Nuances for Autistic Users. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility* (St. John's, NL, Canada) *(ASSETS '24)*. Association for Computing Machinery, New York, NY, USA, Article 24, 18 pages. doi:10.1145/3663548.3675633

[13] Mollie Jordan, Kevin Ly, and Adalbert Gerald Soosai Raj. 2024. Need a Programming Exercise Generated in Your Native Language? ChatGPT's Got Your Back: Automatic Generation of Non-English Programming Exercises Using OpenAI GPT-3.5. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 618–624. doi:10.1145/3626252.3630897

[14] Matthew Kam, Cody Miller, Miaoxin Wang, Abey Tidwell, Irene A. Lee, Joyce Malyn-Smith, Beatriz Perret, Vikram Tiwari, Joshua Kenitzer, Andrew Macvean, and Erin Barrar. 2025. What do professional software developers need to know to succeed in an age of Artificial Intelligence?. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (Clarion Hotel Trondheim, Trondheim, Norway) *(FSE Companion '25)*. Association for Computing Machinery, New York, NY, USA, 947–958. doi:10.1145/3696630.3727251

[15] Vassilka D. Kirova, Cyril S. Ku, Joseph R. Laracy, and Thomas J. Marlowe. 2024. Software Engineering Education Must Adapt and Evolve for an LLM Environment. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 666–672. doi:10.1145/3626252.3630927

[16] Evanfiya Logacheva, Arto Hellas, James Prather, Sami Sarsa, and Juho Leinonen. 2024. Evaluating Contextually Personalized Programming Exercises Created with Generative AI. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1* (Melbourne, VIC, Australia) *(ICER '24)*. Association for Computing Machinery, New York, NY, USA, 95–113. doi:10.1145/3632620.3671103

[17] Daniel Mejia, Ernest D.V. Holmes, Jenn Marroquin, and Jamie Gorson Benario. 2025. Bridging Academia and Industry: Leveraging Generative AI in a Software Engineering Course for Practical Industry Experiences. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Nijmegen, Netherlands) *(ITiCSE 2025)*. Association for Computing Machinery, New York, NY, USA, 507–513. doi:10.1145/3724363.3729036

[18] Goda Nagakalyani, Saurav Chaudhary, Varsha Apte, Ganesh Ramakrishnan, and Srikanth Tamilselvam. 2025. Design and Evaluation of an AI-Assisted Grading Tool for Introductory Programming Assignments: An Experience Report. 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, 805–811. doi:10.1145/3641554.3701913

[19] n.d. 2025. Llama 3 models. https://www.llama.com/models/llama-3/.

[20] n.d. 2025. Meet Claude Anthropic. https://www.anthropic.com/claude.

[21] n.d. 2025. Phi open model family. https://azure.microsoft.com/en-us/products/phi/.

[22] n.d. 2026. The Unified Interface For LLMs. https://openrouter.ai.

[23] n.d. 2026. Unify AI Power. https://www.librechat.ai.

[24] Nishat Raihan, Mohammed Latif Siddiq, Joanna C.S. Santos, and Marcos Zampieri. 2025. Large Language Models in Computer Science Education: A Systematic Literature Review. 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, 938–944. doi:10.1145/3641554.3701863

[25] Liam Stienstra, Abdallah Mohamed, and Mostafa Mohamed. 2025. Exploring GenAI as a Tutoring Tool: A Case Study in First-Year Computer Programming. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Nijmegen, Netherlands) *(ITiCSE 2025)*. Association for Computing Machinery, New York, NY, USA, 452–457. doi:10.1145/3724363.3729060

[26] Benyamin Tabarsi, Heidi Reichert, Sam Gilson, Ally Limke, Sandeep Kuttal, and Tiffany Barnes. 2025. LLMs' Reshaping of People, Processes, Products, and Society in Software Development: A Comprehensive Exploration with Early Adopters. arXiv:2503.05012 [cs.SE] https://arxiv.org/abs/2503.05012

[27] Ismael Villegas Molina, Audria Montalvo, Shera Zhong, Mollie Jordan, and Adalbert Gerald Soosai Raj. 2024. Generation and Evaluation of a Culturally-Relevant CS1 Textbook for Latines using Large Language Models. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 325–331. doi:10.1145/3649217.3653600

[28] Kevin Shukang Wang and Ramon Lawrence. 2025. Quantitative Evaluation of Using Large Language Models and Retrieval-Augmented Generation in Computer Science Education. 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, 1183–1189. doi:10.1145/3641554.3701917

[29] Tianjia Wang, Matthew Trimble, and Chris Brown. 2025. DevCoach: Supporting Students Learning the Software Development Life Cycle with a Generative AI powered Multi-Agent System. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (Clarion Hotel Trondheim, Trondheim, Norway) *(FSE Companion '25)*. Association for Computing Machinery, New York, NY, USA, 987–998. doi:10.1145/3696630.3727255

[30] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.