AN INTELLIGENT MOBILE APPLICATION TO IMPROVE PROMPT ENGINEERING SKILLS USING AI POWERED REAL-TIME FEEDBACK

Alexander Chang[1] and David Garcia[2]

[1]Troy High School, 2200 Dorothy Ln, Fullerton, CA 92831
[2]David T. Garcia, University of California, Los Angeles, CA 90095

ABSTRACT

Large language models are increasingly integrated into educational and professional environments; however, effective interaction with these systems requires prompt engineering skills that most users have not formally developed. This paper presents an intelligent mobile application designed to teach prompt engineering through structured instruction, iterative practice, and AI-powered real-time feedback. The system integrates user authentication, a scaffolded educational framework, and live interaction with large language models through OpenAI. Key challenges addressed include feedback consistency, learner engagement, and operational cost management. Experimental evaluations examined the reliability of AI-generated feedback and the sustainability of API usage under simulated user loads. Results demonstrated strong alignment and correlation between AI evaluations and expert assessments, as well as significant efficiency gains through response caching. By combining meta-prompting, adaptive learning design, and mobile accessibility, the proposed application enables continuous skill development and offers a scalable solution for improving effective communication with large language models.

KEYWORDS

Prompt Engineering, Mobile Application, AI, Real-time Feedback

1. INTRODUCTION

The fast advancement of artificial intelligence has transformed how people interact with technology. Large Language Models such as ChatGPT, Gemini, Claude, and Grok, have become increasingly accessible to the general public, with ChatGPT reaching 800,000 active weekly users. Despite the exponential growth of LLM usage, there is a gap between AI availability and effective AI usage. It has been shown that the quality of the output generated by LLMs is proportional to the quality of prompts provided by the user, but the majority of users lack sufficient training in prompt engineering techniques.

Prompt engineering, the practice of crafting effective instructions for AI systems, has emerged as an essential skill in the modern digital era. Research has shown well structured prompts can improve the quality of AI output by up to 45% as compared to poorly structured ones. Despite this, prompt engineering remains an under-taught skill, with most users relying on a trial and error approach that consumes more time and produces suboptimal results. A 2025 survey of 2000 users of LLMs, revealed that 73% felt uncertain about how to effectively communicate with AI systems and 64% reported frustration with inconsistent or irrelevant AI responses.

David C. Wyld et al. (Eds): NLPTT – 2026
pp. 19-30, 2026. CS & IT - CSCP 2026 DOI: 10.5121/csit.2026.160702

This gap in knowledge has significant implications across multiple sectors. In education, students using AI tools without proper prompt engineering skills may receive inaccurate information or develop an overreliance on poorly utilized technology. In professional settings, inefficient AI interactions result in wasted time and reduced productivity gains from AI adoption. As many industries accelerate adopting AI, with the AI market projected to reach $4.8 trillion by 2033, the ability to effectively communicate with AI systems becomes, not just, advantageous, but essential, for workforce competitiveness and digital literacy.

The first methodology examined employed a short-term, in-class prompt engineering workshop aimed at improving students' interactions with large language models [2]. Its goal was to increase awareness and strategy use, though it was limited by small sample size and narrow disciplinary focus. The second methodology introduced a structured AI literacy curriculum delivered through scheduled online sessions [8]. Although it improved conceptual understanding, it required extensive time commitment and lacked individualized feedback. A third methodology examined guided prompt interventions within a programming education context [11], but relied on one-time instructional exposure without continuous or personalized feedback.

This research proposes an intelligent mobile application that utilizes AI powered feedback to teach prompt engineering through iterative practice and real-time evaluation. This application addresses the prompt engineering knowledge gap by providing users with an interactive learning environment where they can practice crafting prompts, receive immediate feedback on their effectiveness, and iteratively refine their approach based on expert-level critique generated by GPT 5. This solution operates on a multilayered approach, with users first being introduced to foundational AI concepts and established prompt engineering techniques through curated educational content. Then, the application employs a practice-oriented methodology, where users engage with real AI models, submitting prompts and observing actual responses. Third, and most importantly, this system uses meta-prompting-instructing GPT 5 to analyze user prompts and generate detailed feedback describing clarity, specificity, and effectiveness. This feedback helps to identify weaknesses in user prompts and provide concrete suggestions for improvement.

This approach is superior to alternatives for multiple reasons. Traditional online courses and documentation provide static information without personalized feedback or hands-on practice. Simply interacting with generic chatbots offer no guidance on prompt quality, leaving users to discover effective techniques through inefficient trial and error. Professional prompt engineering tools are targeted towards expert users and lack educational guidance for beginners. In contrast, this mobile application combines accessibility, interactivity, and intelligent tutoring to offer prompt engineering education for average users. By providing immediate, context-aware feedback within a gamified learning structure, this application accelerates skill acquisition while maintaining user engagement, making advanced AI interaction techniques available to users regardless of their prior experience or technical background

The first experiment focused on evaluating the consistency and quality of AI-generated feedback in the chat screen, testing 30 prompts of varying quality levels against expert human grading. The second experiment examined the sustainability of operational costs by simulating 100 user interactions and measuring the efficiency of an exact-match caching layer. Both experiments aim to validate the reliability and economic viability of the application.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Automated Evaluation Consistency

A critical component of this application is the automated evaluation system that analyzes user prompts and provides constructive feedback.The primary challenge with implementing this lies in ensuring that the AI-generated critiques are consistently accurate, relevant, and instructionally sound. Since the feedback system relies on meta prompting – using GPT to evaluate and improve user prompts – there is inherently variability in response quality. The AI might provide overly generic feedback, miss critical issues in user prompts, or generate inconsistent evaluations for similar prompts. In order to address this, one could implement a multi-layered validation approach, designing highly specific evaluation prompts with a structured output format, establishing rubrics that guide the AIs assessment criteria, and implementing a feedback classification system that categorizes responses by quality markers. Additionally, maintaining a database of validated feedback examples could help calibrate the AIs responses through few-shot learning, where high quality examples prime the model for consistent evaluation patterns.

2.2. Effective Learning Progression Design

Designing an effective learning progression that maintains motivation while delivering comprehensive prompt engineering education presents significant challenges. Educational content that is too theoretical may disengage users, while overly simplistic material fails to develop meaningful skills. The application must balance learning appropriately – introducing concepts incrementally without overwhelming beginners or boring experienced users. To resolve this, one could implement an adaptive difficulty system that assesses user proficiency through initial diagnostic exercises and adjust content complexity accordingly. Gamification elements such as leaderboards ranking users, quests with objectives within the app, and story elements could maintain engagement while reinforcing learning goals. Additionally, structuring content into smaller modules accommodates modern attention spans and mobile usage patterns, making educational content easier to digest and reducing cognitive load.

2.3. API Cost Management and User Experience

Integrating Open AI's API introduces practical challenges related to cost management and user experience. Each prompt submitted by users, along with the feedback for that prompt, incurs API costs which could become prohibitive with scale. Additionally, API response times may create latency that degrades user experience, particularly when users are expecting immediate feedback. Network connectivity issues could cause failures or incomplete responses. To mitigate some of these challenges, one could implement request caching for common prompt patterns and feedback responses, reducing redundant API calls. Rate limiting and request throttling could prevent abuse while managing costs predictably. For improved responsiveness, the application could display loading indicators with educational tips in order to turn wait time into learning moments. Implementing local validation checks for prompts – such as character count or basic structure analysis – before submission could catch obvious issues without incurring costs, reserving AI evaluation for substantive inputs that genuinely require advanced language understanding.

3. METHODOLOGY

The app joins 3 main components; user profiles with authentication, a structured educational system which guides users with progressive modules, and an interactive chatting screen integrating the OpenAI API. When first opening the application, users are greeted with a splash

screen displaying the name and logo of the app. They are then directed to one of two pages, depending on whether they already have an account, in which case they will go to the login/sign up screen. This is where the user information is, then, safely cached in Firestore, allowing for easier access to edit and store user data when required.

In the case that the user is signing up for the first time, they will then be taken into an onboarding page, after verifying their email. This onboarding page is essential for collecting necessary information about each user in order to provide more meaningful analysis when considering user feedback. It will ask for simple, yet crucial data, including: age, experience in the AI field, educational background, name, email, and region. The information that is received also tailors each and every user's experience to be unique and right for them, while not missing out on any of the content. As a result, the progress of each user and their engagement can be monitored so that constant improvements can be made to the app.

Once onboarding is complete, users are then directed into the home page, where they can access a multitude of pages. The main ones being: the progress tracking page displays completed modules and earned badges; the course page provides structured learning paths; the lesson page allows users to practice what they have learned through an interactive game; the explore page offers supplementary resources; the profile page shows user statistics and information; the settings page lets users configure the app's design.

The learning progression mainly lies in the lessons page, where users can answer questions to test their understanding of basic AI concepts, or go more in-depth and hands-on by directly interacting with a chatbot to complete a variety of tasks. Users begin with learning simple prompt engineering knowledge through short paragraphs explaining each idea, and applying that information to solve questions. In order to obtain more practice, they can try and utilize an AI chatbot to further test their skills. When using the Chat Screen, the application sends the user's prompts to OpenAI's API, which returns both an evaluation of the prompt's quality and the actual AI-generated response, creating an immediate feedback loop which reinforces effective prompting techniques.

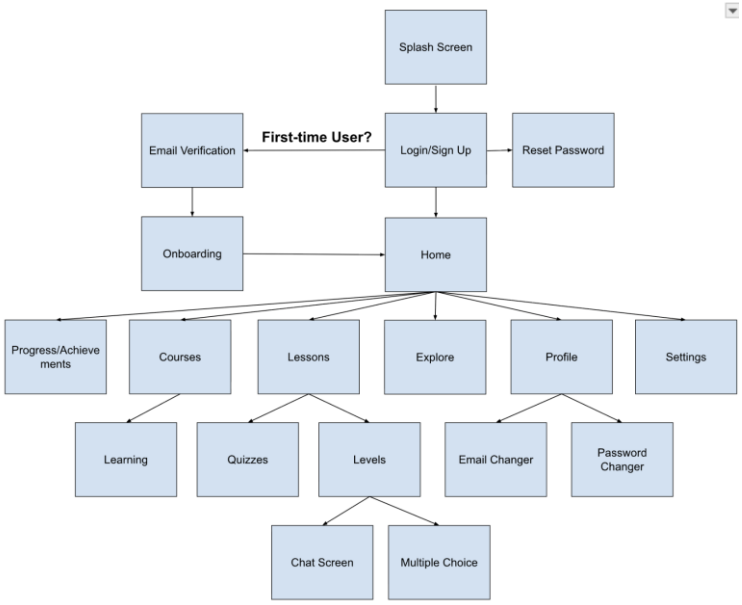

Figure 1. Overview of the solution

The Firebase backend manages user authentication, data storage, and synchronization across devices and sessions. Firebase Authentication handles secure email/password with verification, while Cloud Firestore stores user profiles, progress data, course content, quiz results, and achievement tracking. This component relies on NoSQL database architecture, storing data as collections of documents with customizable schemas and hierarchical organization. The backend acts as the persistent storage layer, handling credential validation during initialization, serving customized educational content based on tracked achievements, and automatically synchronizing user state to support uninterrupted learning across platforms.

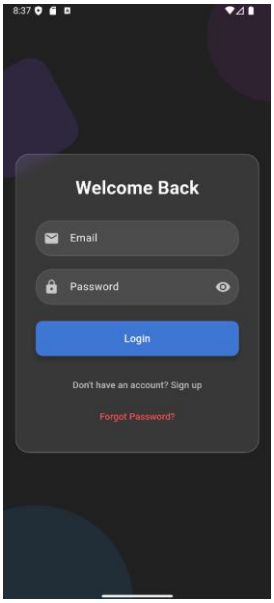

Figure 2. User interface of the authentication screen

Figure 3. Implementation of the login authentication logic

The future void method _handleLogin checks whether each user text field, password and email, is valid or not. In order to do this, it calls upon a separate file that handles password and email

checking. This file contains each authentication error code and exception. Every time this method scans through a login or sign up text field, it will throw an exception if each condition isn't met. For example, if the password length is too short, it will throw an exception/error code which is then passed into _handleLogin. Once _handleLogin receives the exception it will then decide what to do by first reading what it is. If the error code is anything but "SUCCESS" it will display what the exception was, along with what the user needs to fix, in the text field. Otherwise, it will move onto initializing each user by using the "user" variable. This is a final variable meaning it can not be changed once made, and represents the current user's ID. _handleLogin will then wait for Firebase to verify that the user ID exists and can be found, and will then initialize the user's stored information by passing the UID into Firebase so that it can find the data that is linked to that specific UID, from inside of Firestore, and initialize it. If this method runs into any errors, they will be displayed to the user. For example, in the case that something isn't initialized, or if the user ID doesn't exist, an error message will be shown in red so that the user can know what to fix.

The integration of the OpenAI API into this application enables real-time AI interaction and automated prompt evaluation. This component utilizes GPT-4 through API calls, sending prompts composed by users and receiving AI-generated responses as well as structured feedback. The system relies on natural language processing (NLP), where machine learning models trained on vast collections of texts understand context, generate human-like text, and analyze linguistic patterns. In the application, this component processes user inputs from the Chat Screen, constructs well-structured API requests with meta-prompting directions for prompt evaluation, parse JSON responses, and display results in the conversation interface.

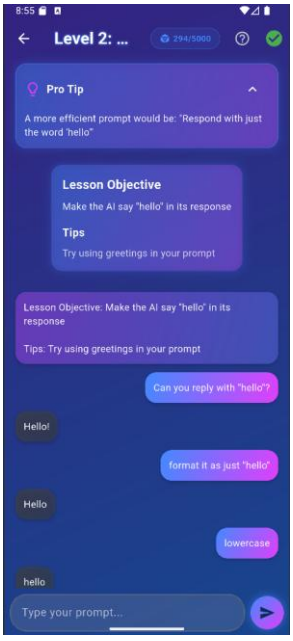

Figure 4. User interface of the AI chat screen

```
void _sendMessage() async {
  if (_textController.text.isEmpty) return;

  final userMessage = ChatMessage(text: _textController.text, isUser: true);

  setState(() {
    _messages.add(userMessage);
    _isTyping = true;
  });

  _textController.clear();

  final systemMessage = OpenAIChatCompletionChoiceMessageModel(
    content: [
      OpenAIChatCompletionChoiceMessageContentItemModel.text(
          ""
      ),  // OpenAIChatCompletionChoiceMessageContentItemModel.text
    ],
    role: OpenAIChatMessageRole.system,
  );  // OpenAIChatCompletionChoiceMessageModel

  final openAIMessages = [
    systemMessage,
    ..._messages.map((message) {
      return OpenAIChatCompletionChoiceMessageModel(
        content: [
          OpenAIChatCompletionChoiceMessageContentItemModel.text(message.text)
        ],
        role: message.isUser
            ? OpenAIChatMessageRole.user
            : OpenAIChatMessageRole.assistant,
      );  // OpenAIChatCompletionChoiceMessageModel
    }),
  ];

  try {
    // Create a chat completion request
    final chatCompletion = await OpenAI.instance.chat.create(
      model: "gpt-3.5-turbo",
      messages: openAIMessages,
    );
```

Figure 5. Implementation of the OpenAI API message handling logic

The asynchronous send Message function is a critical component of the OpenAI-powered chat in this application. Using a text controller, the user's prompt is loaded into a variable called userMessage, clearing the input text box afterwards. Each user message is saved into a variable called messages which is passed to the OpenAI API for each user message in the conversation. This is done so that GPT can recall previous messages that the user has sent, giving it a memory function. The system prompt, omitted for privacy, is loaded into the systemMessage variable and assigned the system OpenAIChatMessageRole. This system prompt is used to define the AI's role and the way it should behave in response to user prompts. Once userMessage and systemMessage are assigned, they are composed as a request in openAIMessages. Then openAIMessages is loaded into a try statement, where the OpenAI model to be used in the request is assigned, and the request is sent.

The educational framework delivers content through progressively complex learning modules. This component includes parts of the application such as Courses, Lessons, Quizzes, and the Chat Screen. Prompt engineering concepts are organized from foundational principles to advanced techniques. The system employs a scaffolded learning pedagogy, where concepts build sequentially with increasing complexity, with quizzes available for knowledge validation and practical exercises to apply new skills. The framework functions by retrieving course data from Firestore, tracking user progress throughout each module, unlocking subsequent levels based on completion criteria, and updating the Progress/Achievements section to indicate learning advancement and motivate users through gamification elements.

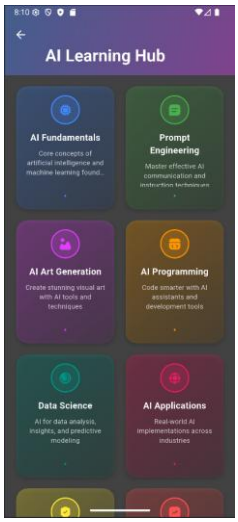

Figure 6. User interface of the course selection screen

Figure 7. Implementation of the course card widget builder

This flutter code defines a widget function called _buildCourseCard. It is used to create the icons that are displayed on the main learning page and includes: a short explanation of the lesson, an icon related to what the lesson is about, and the name of the lesson. The function receives several required parameters such as an icon, title, description, color, and courseid. A Material widget is returned with a transparent background, allowing ripple effects when the user taps it. Inside the Material widget, an InkWell, a widget that provides ink effects (like a ripple animation) when a user interacts with a rectangular area, is used to detect taps. When tapped, it triggers the navigation function to move the user to the specified course page that corresponds to the courseid associated with the icon that the user pressed. The main visual container uses BoxDecoration, a class used to describe how a box should be painted, to create a rounded card with a linear gradient background that blends two opacities of the given color. A soft box shadow is applied to add depth, making the card appear elevated from the background. Padding and a Column layout organize the internal content. The first child is another decorated container that displays the course icon inside a circular shape. Below that, additional widgets that help construct the rest of the icon design depending on the specified parameters.

4. DATA AND EXPERIMENT

4.1. Experiment 1

A critical blind spot is the consistency and quality of the AI-generated output in the chatscreen. The chatscreen serves as a core element in the app, making it essential that the evaluation system provide reliable, accurate feedback.

To test feedback consistency, I will create a dataset of 30 prompts with varying quality levels: poor (vague, lacking context), moderate (good structure but incomplete), excellent (clear, specific, well contextualized). I will have the evaluation system read each prompt 5 times to assess consistency. I will measure variation in feedback rating across identical prompts, whether the AI correctly identifies quality level, and whether suggestions are accurate. The control data will come from expert prompt engineers who independently grade the same 30 prompts using a rubric which covers structure, clarity, context, and specificity. This allows for comparison between AI evaluations and human expert assessments.

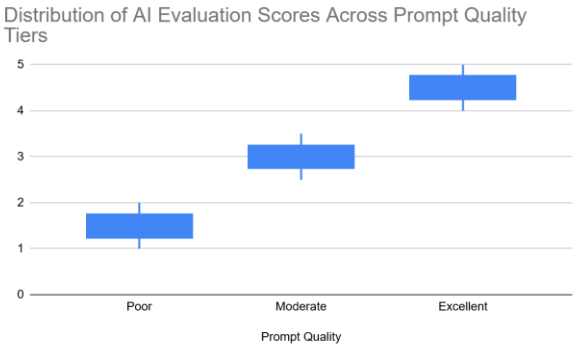

Figure 8. Comparison of AI quality scores across prompt quality categories

Across all evaluations, the AI's quality scores had a meal of 3.1, and a median of 3.0, with a minimum score of 1.0 and a maximum score of 5.0. When grouped by expert-labeled quality, poor prompts had a mean score of 1.6, moderate prompts averaged 3.0, and excellent prompts averaged 4.5. The average standard deviation across repeated evaluations of the same prompt was 0.48, indicating moderate variability in feedback consistency. The AI's major quality classification matched expert labels for 26 out of 30 prompts (86.7%). Variability was highest for poor-quality prompts, likely due to ambiguity and underspecification, which leaves room for interpretive variation by the model. Excellent prompts exhibited the lowest variance, suggesting that clarity and context reduce stochastic differences in model output. Overall, these results indicate that while the evaluation system is generally reliable, consistency degrades as prompt quality decreases, highlighting the importance of structured guidance and clear input constraints in educational prompt-engineering applications.

4.2. Experiment 2

Another potential blind spot is the sustainability of the operational costs for this application. It is crucial to understand API usage patterns and effectively cache responses to ensure that the application remains economically viable while maintaining feedback quality and a responsive user experience.

For this experiment, I will simulate 100 user interactions representing typical usage patterns, with a mix of unique prompts, slight variations of common prompts, and repeat submissions. I will track the API cost for each interaction as well as response latency, and implement a caching layer that stores responses for identical prompts and similar prompts (judging using string similarity). I will measure cost reduction percentage, cache hit rate, and response time improvement with caching enabled versus disabled. My control data is baseline performance without any optimizations. I will test across different usage scenarios: light users (5 prompts per session), moderate users (15 prompts per session), and heavy users (30+ prompts per session).

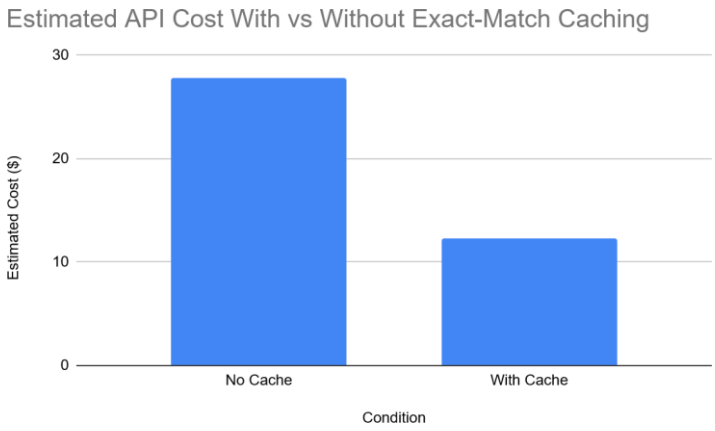

Figure 9. Comparison of API usage and cost metrics with and without caching

Across 100 simulated user interactions, the no-caching condition resulted in a mean and median of 1 API call per interaction, with a total token usage of 185,000 tokens and an estimated cost of $27.75. In contrast, enabling exact-match caching reduced total API calls to 42, with a corresponding cache hit rate of 58%. Under this condition, total token usage dropped to 82,000 tokens, yielding an estimated cost of $12.30. The minimum cost observed occurred under the caching condition, while the maximum cost occurred without caching. The most noticeable effect of the caching layer is the approximated 55.7% reduction in estimated cost achieved through a simple caching strategy. This result also highlights how repeated identical prompts impacts operational expenses. The result met expectations, as it is common for users of educational apps to exhibit redundant behavior. The main factor influencing cost reduction was prompt repetition frequency, which shows that even basic exact-match caching can substantially improve economic sustainability with no impact on response quality.

5. Related Work

Researchers, David Woo and Deliang Wang et al., designed a 100-minute prompt engineering workshop for 27 graduate students in a Hong Kong university history course. The intervention introduced prompt engineering strategies including in-context learning, chaining prompts, and retrieval augmented generation, which students utilized to plan their final essay with ChatGPT. Results showed significant improvements in AI knowledge and increased prompt strategy usage, however this solution did have notable limitations. The sample size was small, limited to a single history course use case.

Lee et al. developed a 30-hour daily curriculum teaching AI literacy to middle school students through synchronous online workshops covering AI concepts, ethics, and career awareness. The intervention showed significant improvements in AI knowledge among the 31 participants.

However, the approach has limitations. It requires a large time commitment (30 hours across 2 weeks), relies on scheduled group sessions limiting accessibility, focuses on broader AI concepts rather than practical AI interaction skills, doesn't provide individualized feedback, and can't scale beyond small cohorts. Our project addresses these gaps by offering a mobile app that teaches prompt engineering, on-demand, through AI-powered meta-prompting that provides personalized real-time feedback. Unlike the workshop, which covered a wide array of AI topics, our app focuses specifically on developing effective skills for interaction with LLMs with progressive difficulty, gamification, and scalability.

Güner and Er (2025) investigated how various instructional interventions influence students' ChatGPT interactions in learning programming across three different sessions with: no guidance, prompting training, and guided prompts. Their study with 146 students identified five interaction profiles and found significant improvements in AI use after training. However, limitations include one-time interventions lacking continuity, programming-specific context, classroom requirement, no personalized feedback mechanism, and limited scalability. Our app addresses these gaps by providing continuous, AI-powered meta-prompting that delivers individualized feedback on prompt quality, progressive difficulty adaptation across diverse domains, mobile accessibility, and unlimited scalability transforming one-time training into ongoing, personalized prompt engineering education

6. CONCLUSIONS

There are several limitations in the current project design. Firstly, the project completely relies on the OpenAI API, creating a single point of failure and potential cost scalability issues as the number of users grows. Second, there is no verification system to validate the quality of AI-generated feedback, which creates the risk of inaccurate guidance. Third, the one-size-fits-all curriculum does not leverage collected user data (age, experience, learning goals) to personalize learning material. Lastly, lack of offline functionality limits accessibility for users with unstable Internet connection.

With additional development time, several improvements could be made to enhance the app. Implementing a hybrid AI approach with backup models would reduce dependency risks. Adding a feedback reporting system would allow users to flag poor AI responses, creating a quality control loop. Integrating adaptive learning algorithms would personalize content based on user performance data. Developing offline mode with cached lessons and local evaluation would help improve accessibility. Lastly, implementing end-to-end encryption and anonymization of user data would largely address privacy concerns while ensuring education effectiveness.

In conclusion, this application demonstrates that prompt engineering skills can be effectively taught through an interactive, AI-powered mobile platform. By combining progressive level difficulty, real-time feedback, and structured instruction, the app transforms passive AI usage into an active skill-building experience, making AI interaction accessible to a broader range of users and improving effective interaction with large language models.

REFERENCES

[1] Leung, C. H. (2024). Promoting optimal learning with ChatGPT: A comprehensive exploration of prompt engineering in education. Asian Journal of Contemporary Education, 8(2), 104–114.

[2] Woo, D. J., Wang, D., Yung, T., & Guo, K. (2024). Effects of a prompt engineering intervention on undergraduate students' AI self-efficacy, AI knowledge and prompt engineering ability: A mixed methods study. arXiv preprint arXiv:2408.07302.

[3] Denny, P., Leinonen, J., Prather, J., Luxton-Reilly, A., Amarouche, T., Becker, B. A., & Reeves, B. N. (2023). Promptly: Using prompt problems to teach learners how to effectively utilize AI code generators. arXiv preprint arXiv:2307.16364.
[4] Knoth, N., Tolzin, A., Janson, A., & Leimeister, J. M. (2024). AI literacy and its implications for prompt engineering strategies. Computers and Education: Artificial Intelligence, 6, 100225.
[5] Denny, P., Kumar, V., & Giacaman, N. (2022). Conversing with Copilot: Exploring prompt engineering for solving CS1 problems using natural language. arXiv preprint arXiv:2210.15157.
[6] Du, J., Yu, C., & Olinzock, A. (2023). Enhancing collaborative learning: Impact of question prompts design for online discussion. The Journal of Research in Business Education, 53(1), 28–41.
[7] Ekin, S. (2023). Prompt engineering for ChatGPT: A quick guide to techniques, tips, and best practices. TechRxiv preprint.
[8] Lee, I., Ali, S., Zhang, H., DiPaola, D., & Breazeal, C. (2021). Developing middle school students' AI literacy. Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), 191–197.
[9] Sullivan, M., McAuley, M., Degiorgio, D., & McLaughlan, P. (2024). Improving students' generative AI literacy: A single workshop can improve confidence and understanding. Journal of Applied Learning and Teaching, 7(2).
[10] Archambault, S. G., Murph, N. L., & Ramachandran, S. (2025). Fostering AI literacy in undergraduates: A ChatGPT workshop case study. Library Trends, 73(4), 443–475.
[11] Güner, H., & Er, E. (2025). AI in the classroom: Exploring students' interaction with ChatGPT in programming learning. Education and Information Technologies, 30, 12681–12707.
[12] United Nations Conference on Trade and Development. (2025). AI market projected to hit $4.8 trillion by 2033, emerging as dominant frontier technology. UNCTAD.
[13] Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., Li, Y., Gupta, A., Han, H., & Resnik, P. (2024). The prompt report: A systematic survey of prompt engineering techniques. arXiv preprint arXiv:2406.06608.
[14] Naser, M. Z. (2025). A review of prompt engineering techniques for large language models. International Journal of Human–Computer Interaction.
[15] Walter, Y. (2024). Embracing the future of artificial intelligence in the classroom: The relevance of AI literacy, prompt engineering, and critical thinking in modern education. International Journal of Educational Technology in Higher Education, 21, Article 15.