Table of Contents

1. Additional Information

1.1. Prompt Engineering Strategy

- How did you design your prompts for the LLM?
  - The prompts were designed with two main goals in mind.
  - As I have grown to have a greater and deeper understanding of the LLMs, I have learned that too much in the prompt causes issues. When there is too much guidance it can start to hallucinate.
  - The best prompts were short, concise and well formatted.
- Did you iterate or change prompts over time?
  - It turns out that even the style of the prompt influences the style of the tutor.
  - Overtime the prompts grew to be formatted in markdown, which seemed to work the best, and were stripped down to the shortest possible prompt.
- What worked well? What failed?
  - What didn’t work well is giving the tutor a million rules or guidelines to follow.

        - In fact the more that I tried to enforce or “tell” it what to do, it would hallucinate more.
- How did you make prompts “educational” instead of generic?
    - Really the best way that the prompts were educational was telling the tutor in a straightforward and direct way:
        - You are a Tutor for Utah Valley University
    - More specifically I gave the particular students courses along with those instructions, like:
        - You are an expert in [Student’s Course] and that course is about [Course Catalog Description]

1.2. LLM Integration Decisions

- Which LLM or APIs did you use? Did you try different LLMs
    - OpenAI
        - We used OpenAI due to the fact that we had a grant for several thousand dollars of credits.
    - Anthropic
        - We switched to the Anthropic API, when one day we had issues accessing our OpenAI credits. We were able to quickly switch, due to the fact that our service was broken.
- Why did you choose it?
    - OpenAI: because we had the grant money.
    - Anthropic: As mentioned above, but I did like Anthropic because it seemed to be more emotionally intelligent, but more expensive.
- How did you manage API limits or costs?
    - We used use gpt-4o-mini when possible, given that it cost about an order of magnitude or two less than the gpt-4o API.
    - We also recorded the total tokens used on every message in our analytics service.
- How did you handle rate limits or token usage?
    - Automatically, the code would clear context (the chat conversation), not saving messages by default. This saved a lot of money in credits and tokens.

1.3. Handling Hallucinations and Errors

- Did you see the LLM give wrong or hallucinated answers?

    - Yes, usually after a conversation had gone on for some time, The AI would forget the initial prompt and start making up assignments or grades.
    - This was really bad, and we used tool calls to have more consistent and correct output.
- How did you test outputs for correctness?
    - Given that its nearly impossible to test every message the AI can send, I as the developer spent many hours testing the output of the LLM before making changes.
- Did you filter or post-process outputs?
    - Yes, we used Regular Expressions and some special tricks to format the output. This was specifically useful for math.
    - We also moderated every chat message to check for violence, sexual messages, or self harm. Anytime those messages were sent, they were logged and blocked.
    - Note in the case of self-harm, we sent the UVU student health services link.
- What would you improve here next time?
    - I would have more tool calls and more post processing, but that would also use more credits.

1.4. Adapting AI for Educational Use

- How did you try to make AI responses educationally useful?
    - We used the activity stream from the Canvas LMS so that the tutor was always up to date with the student's performance in courses.
    - We also used the 2024-2025 UVU course catalog to provide the AI with more context about each course as a whole.
- Did you add scaffolding, hints, or explanations? Any thing else?
    - The most similar to this would have been the Course catalog as mention above.
- How did you ensure clarity and appropriateness?
    - Along with moderation, we just add that the LLM should be "Excited and provide concise and useful responses".

1.5. AI Performance Metrics

- Typical latency / response time?
    - We never had any issues with response time.
    - The only issue was that occasionally if the app was not being used, it would be slow to start up at first. (This was to save hosting costs)

- Cost per API call or session?
  - Approximately one message and response was between 150 - 500 tokens plus the system message, which was also between 150-300 tokens.
  - Although, the cost per session is somewhat hard to calculate, This is just because you have to take the cost of each token $t \approx \frac{1{,}000{,}000}{\$2.00}$ and then multiply that by all the previous messages **including** the one you are sending now. So it is somewhat exponential in cost as the length of a chat increases.

$$S \cdot t + \sum_{i=1}^{n} (U + A) \cdot \left(n(i-1) - \frac{i(i+1)}{2} \right) \cdot t$$

- Average tokens used per prompt/response?

  See above.

- Any notable error rates or failures?

  Not necessarily

1.6. Lessons and Recommendations

- What would you recommend to another team trying to integrate LLMs for learning?
  - First of all good luck, its tough to get the AI to say what you want. It has taken me about 2 years.
  - Secondly, tool calls are the future of truly useful LLM applications.
- What trade-offs would you highlight?
  - The trade off between hallucinations vs novelty or fun conversations is truly tough.
- What would you do differently next time?
  - I would just have more tool calls and rely on them much much more.

2. Small-Scale AI Tutoring Systems

- Balancing Technical Constraints and Educational Goals

2.1. Prioritization and MVP Decisions

- How did you prioritize tasks and features given limited time?
  - This is a tough question. I think generally speaking, I evaluate the entire list of things that need to get done, and then I construct a directed acyclic graph of the tasks. A directed acyclic graph or DAG is a data structure where, in this example, tasks can have a topological ordering. Once I have a rough topological ordering of features and tasks, I just determine what is most important and then start working on the tasks that need to be completed first.

- What did you decide to “leave for later” and why?
  - This fits in with the answer above, mainly for the reason that when writing software, the question should always be what needs to be done, and is truly important.
  - Too many software projects just create too many features that aren’t really useful.
- How did you decide what was good enough for an MVP (Minimal Viable Product)?
  - Tying into the above questions and answers, the answer was what is genuinely useful to me, as a student and other students who were using the tutor.
- How did you manage trade-offs between code quality and delivery speed?
  - There is no good code. There is only code that is easy to read and understand, actually works, and is performant enough to meet the needs of the business.
  - There are places in the AI Tutor code that can easily be said are not “good code”, but that code works, it is easy to understand, and it is not slow. Sometimes that is just what software is.

2.2. Trade-Offs

For this section, I think what would be best is to create the table when we are writing the report.

- Cost vs. Cloud Provider
- Ease of Use vs. API Limitations
- Database Choice: MongoDB vs. SQL
- Prompt Engineering vs. Code Control
- Live Users vs. Maintainability
- MVP Speed vs. Technical Debt
- Security vs. User Convenience
- Hosting Costs vs. Scalability
- Speed of Development vs. Feature Completeness
- Time Constraints vs. Code Quality

2.3. Challenges with Live Use

- Your system was used by real teachers and students. What challenges did that create?
  - Users always find bugs that developers didn’t know existed. While this is a challenge, a more difficult challenge is getting those users to tell you the bugs, if they even **know** it was a bug.

    - Also, many headaches were had attempting to migrate or change database schema or code during live use. This is like the epitome of difficult in software development in my opinion.
    - For example, adding a setting to save chats. This caused the code to have to check, if a user has saved a chat If the version of the tutor that their data reflects knows it can save a chat, add a default `NULL` value, if it was their first time using the tutor with the ability to save chats, and then change all the code to be able to deal with that `NULL` value. This is just one example in the code, but this was a common pattern.
- How did you deal with issues like bugs, uptime, or scaling?
    - Uptime is incredibly important. Live services should always stay up.
    - So when changes needed to be made, they were usually pushed up to the server in the middle of the night, specifically when no one was using the Tutor.
    - Scaling was not an issue we had to deal with due to good upfront design.
- How did “real user” demands shape your design or priorities?
    - Users think they know what they want. Developers think they know what users want. Both are wrong. This is why collecting telemetry (user data) is so incredibly valuable. Telemetry gave us the ability to see what users were actually doing. Then as a data scientist, that data can properly inform developers about what features to add and what bugs are actually features.

2.4. User-Centered Design Choices

- Who did you see as your typical user?
    - The typical user was freshmen and seniors funny enough. Freshmen seemed to really like the features, specifically that the tutor had knowledge about their courses, grades, and assignments. While seniors liked that the tutor was fast and to the point.
- How did you learn about their needs?
    - The little sidebar provided by the extension was a favorite feature.
    - Students also seemed to like that the tutor would help them understand content of an assignment, instead of just giving them the answer.
- How did you test or validate whether your design met their needs?
    - Lots of questions to users that I knew.
- What did you do to make the system easier to use for them?
    - The biggest features I added to do this were: A message that would send when they opened the tutor, that just explained what the tutor was, what it could do, and where to learn more.

    - I also designed the tutor with big buttons to start a new chat, save a chat, etc.

2.5. Educational Value Considerations

- How did you try to make the AI Tutor helpful for learning?
- What kind of feedback or support did you want it to provide?
- How did you balance complexity vs. clarity in responses?

All of these questions I feel have been answered in the first section

- How would you improve its educational value next time?
    - I think giving the tutor the ability to investigate deeper into the Canvas LMS would help.

2.6. Usability and Interface Design

- How did you design the chat interface?
    - The chat interface itself was designed to be as simple and minimal possible.
    - I wanted it to be free from distractions, as well as feel friendly, which I believe I did well.
- What choices did you make to keep it simple or clear?
    - I made the sidebar automatically collapse, as well as trial and error to make it intuitive to use.
- Did you get any user feedback on the interface?
    - Mostly just communication with users that I knew. As well as the team.
- What would you change about the UI/UX based on your experience?
    - There are nice little fancy features that I would like to add, but all-in-all I think they would distract from the main purpose of the tutor.
    - I am not sure I would change much, the simplicity is excellent in my opinion.

2.7. System Goals and Scope Decisions

- What were your main goals for the AI Tutor system?
    - My main goals were to create a system that actually knows about students courses.
    - Anyone can go to ChatGPT, but ChatGPT is not connected to Canvas in the way our tutor is.
- How did you decide which features to include or leave out?
    - I already answered this question above.

- How did technical constraints (time, cost, resources) shape your scope?
  - This ties in with the idea of the DAG above, but time, cost, and resources are the shape of the scope.
- What educational goals were most important to you when designing?
  - Accuracy, friendliness, genuine utility.

2.8. Reflections on Working as a Team

- How did you divide roles and responsibilities?
  - Having a project manager was incredibly helpful. When I am doing technical work, it is a very rough context switch to think about meetings, emails and communication in general.
- How did you manage time and tasks?
  - Referencing the DAG again, determining what is most important, and what needs to be done first.
  - There are still features and code I wish I could write, but those things were not nearly as important.
- What worked well in your teamwork?
  - Github issues was incredibly helpful for me.
  - Having consistent team meetings, and just chatting with the team helps with moral and cohesion.
- What was challenging about working as a team on this project?
  - While communication is good, communication does not scale. What I mean by this is that, when new people are added to the team, it takes more time away from development to answer questions and educate team members.
  - I am a strong believer that small teams (no larger than 4 **maybe** 5) write truly innovative and useful software, in ways that big teams simply cannot.

2.9. Personal Learning Outcomes

- What technical skills did you develop (e.g., API integration, UI design, backend work)?
  - System Design: How the system works together as a whole.
  - Database Design: The shape and function of the data that powers and application.
  - Asynchronous Code: Using Asynchronous code allowed the tutor to feel more smooth and fast.
  - Network Communication: Every piece of the tutor had to communicate.
- What did you learn about balancing technical limitations with user needs?

    - Oftentimes spending time creating innovative and clever solutions to user needs and wants is better than trying to create exactly what the user thinks they want.
- How did you grow as a collaborator or project manager?
    - Patience, as an excited developer it is incredibly difficult to wait for things outside of my control.
- How might this experience affect your future studies or career plans?
    - This project has taught me more about real software development than any course ever could.
    - Many lessons learned from this experience have become foundational to my future career.

2.10. Recommendations for Future Teams

- What advice would you give to another student team trying to build a small-scale AI Tutor?
    - Break dumb rules in a way that won't get you in too much trouble.
- What would you do differently next time?
    - Spend more time designing the database schema, and not stray from that schema.
- What is one big lesson you think is most important to share?
    - Don't be afraid to rewrite most or all the code often.

2.11. Privacy : Explain privacy thinking:

- Did you store any user data? How did you protect it?
    - Yes, we stored an immense amount of user data.
    - We protected the data using several different methods at the same time.
    - In order to access data, the following would be needed:
        - The correct API Key, which is only on my computer.
        - A valid JWT which can only be created with the proper API Key and JWT secret.
    - Our database is also not even accessible from the web. This little trick gives me as a lead developer an incredible sense of security.
- How would you improve privacy/security next time?
    - I am not sure I would improve it. I would probably try to make it less strict.