Utilization of LLMs for Software Design

The **goal** of this survey is to explore how developers utilize Large Language Models (LLMs) in the context of software design, the benefits LLMs bring, and limitations faced by developers. This questionnaire aims to gather insights into the experiences of industry practitioners when utilizing LLMs for software design. It contains 7 closed-ended questions and 4 open-ended questions, which may take about **7～10 minutes**.

Note that no personal information will be involved in this questionnaire and nor will your response be disclosed to the third parties. Please feel free to contact us if you have any questions or concerns.

Thank you very much for your participation!
Yifei Wang (on behalf of the team)
Wuhan University, China
whiten@whu.edu.cn

* Indicates required question

As shown in the example below, the developer chatted with an LLM (specifically, ChatGPT here) for addressing a software design issue, and they engaged in multiple rounds of questions and answers (not all round are fully presented).

> In a Python data pipeline where some stages need to be GPU-accelerated while others run on CPU, how would you structure modules to handle heterogeneous hardware dependencies? What architectural patterns (like dependency inversion or facade interfaces) would help keep the core logic decoupled from acceleration-specific implementations?

To handle heterogeneous hardware dependencies (like GPU and CPU) in a Python data pipeline, where certain stages are accelerated using GPU while others run on the CPU, you should focus on an architectural design that achieves clear separation of concerns and keeps the core logic decoupled from acceleration-specific implementations.

Here are some steps and architectural patterns that can help:

1. Abstraction Layer (Facade or Strategy Pattern)

You can introduce an **abstraction layer** that defines common interfaces for both CPU and GPU operations. This pattern allows the core logic to interact with a unified API without needing to know the underlying hardware implementation.

- **Facade Pattern**: This pattern provides a simplified interface to a set of interfaces in a subsystem. It hides the complex interactions with hardware acceleration libraries (like CUDA or OpenCL) behind a clean, easy-to-use API.

Q1. How many years have you been involved in software development? *

- ◯ < 1 year
- ◯ 1 ~ 3 years
- ◯ 3 ~ 5 years
- ◯ > 5 years

Q2. What role do you play in the software development of your organization? *

- ○ Project manager
- ○ Team Leader
- ○ Requirements Engineer
- ○ Consultant
- ○ Architect
- ○ Developer
- ○ Tester
- ○ Other:

Q3. How often do you chat with LLMs for software design issues? *

- ○ Never
- ○ Very few
- ○ Sometimes
- ○ Whenever I encounter such issues

Q4. Which design tasks have been supported by LLMs according to your experience in software development? *

- [ ] Interface and Protocol Design
- [ ] Architecture Design
- [ ] Data Model Design
- [ ] Code Refactoring
- [ ] Component Dependency Optimization
- [ ] Performance Optimization
- [ ] Security Design
- [ ] Use of Design Patterns
- [ ] User Interface Design
- [ ] Other:

Q5. How many rounds of dialogue with an LLM are ***typically*** required to resolve a design issue or to achieve a satisfactory outcome? *

- ○ Single round
- ○ 2~3 rounds
- ○ 4~10 rounds
- ○ More than 10 rounds

Q6. What are the **typical** purposes of your prompts for addressing software design issues using LLMs? *

- [ ] Recommendation of Design Solution. From the perspective of requirements, describe the problem and request the LLM to generate a design solution. For example, "How to implement a platform like IMDb? Show the architecture."
- [ ] Knowledge Query about Design. Inquire relevant concepts about software design, such as "What is the singleton pattern? Please provide an example".
- [ ] Verification of Design Solution. Present the design solution, code structure, and other relevant details, with the expectation that the LLM will identify any flaws and inconsistencies at the design level. For example, "I designed a system to staging files, check it and show me the unreasonable aspects."
- [ ] Code Generation for Design. Provide design requirements (such as "based on the SOLID principles" and "ensure high coupling and low cohesion") and let the LLM generate code snippets that meet these design specifications.
- [ ] Other:

Q7. What levels of software design issues can be addressed utilizing LLMs? *

- [ ] Architectural level. Primarily involves high-level, overarching architecture and module information. For example, "Can I employ blackboard pattern to coordinate all the data? Alternatively, do you have any suggestions for coordinating them?"
- [ ] Detail design level. This mainly refers to the design at the class level, and is manifested in a number of code files involved, with an emphasis on resolving issues in software design. The design patterns proposed by the Gang of Four are at this level. For example, "... In this context, which is more advantageous: the command pattern or the chain of responsibility pattern?"
- [ ] Code idiom level. Code idioms are code fragments for meeting design requirements when implementing a simple task, algorithm, or data structure that is not a built-in feature in the programming language being used. For example, "Show me examples to achieve lazy initialization using double-checked locking in Java, and explain the details."

Q8. What are the benefits of using LLMs in ***software design*** according to your experience? *

Your answer

Q9. What are the limitations and challenges of using LLMs in ***software design*** according to your experience? *

Your answer

Q10. Do you have any further comments about utilization of LLMs for addressing software design issues?

Your answer

Q11. Do you want to get the results of this survey? If so, please provide your email address. (Email will be kept confidential)

Your answer

Submit Clear form