Low-code LLM: Graphical User Interface over Large Language Models

Yuzhe Cai*, **Shaoguang Mao***, **Wenshan Wu, Zehua Wang, Yaobo Liang, Tao Ge**
Chenfei Wu, Wang You, Ting Song, Yan Xia, Nan Duan and **Furu Wei**
Microsoft Research Asia
v-yuzhecai, shaoguang.mao, wenswu, zehwang, yalia, tage,
chewu, v-wangyou, tsong, yanxia, nanduan, fuwei@microsoft.com

Abstract

Utilizing Large Language Models (LLMs) for complex tasks is challenging, often involving a time-consuming and uncontrollable prompt engineering process. This paper introduces a novel human-LLM interaction framework, **Low-code LLM**. It incorporates six types of simple low-code visual programming interactions to achieve more controllable and stable responses. Through visual interaction with a graphical user interface, users can incorporate their ideas into the process without writing trivial prompts. The proposed Low-code LLM framework consists of a Planning LLM that designs a structured planning workflow for complex tasks, which can be correspondingly edited and confirmed by users through low-code visual programming operations, and an Executing LLM that generates responses following the user-confirmed workflow. We highlight three advantages of the low-code LLM: user-friendly interaction, controllable generation, and wide applicability. We demonstrate its benefits using four typical applications. By introducing this framework, we aim to bridge the gap between humans and LLMs, enabling more effective and efficient utilization of LLMs for complex tasks. The code, prompts, and experimental details are available at LowcodeLLM. A system demonstration video can be found at LowcodeLLM.

1 Introduction

Large language models (LLMs), such as ChatGPT(OpenAI, 2022) and GPT-4(OpenAI, 2023), have garnered significant interest from both academia and industry, as they demonstrate impressive capability across a range of tasks(Bubeck et al., 2023), and are increasingly utilized in a variety of other fields as well(Nori et al., 2023; Choi et al., 2023; Baidoo-Anu and Owusu Ansah, 2023). However, it is not yet perfect in handling complex tasks. For example, when generating a long paper, the presented arguments, supporting evidence, and overall structure may not always meet expectations in diverse user scenarios. Or, when serving as a task completion virtual assistant, ChatGPT may not always interact with users in the intended manner and may even display inappropriate behavior in various business environments.

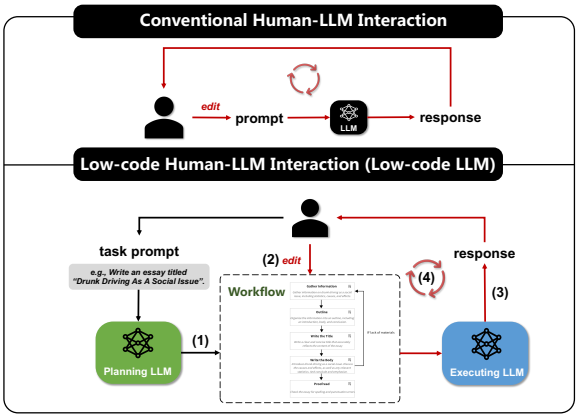


Figure 1: Overview of the Low-code human-LLM interaction (Low-code LLM) and its comparison with the conventional interaction. The red arrow indicates the main human-model interaction loop.

Effective utilization of LLMs like ChatGPT requires careful prompt engineering(Zhou et al., 2022; Wang et al., 2023b). However, prompt engineering can be particularly challenging when instructing LLMs to perform complex tasks, as reflected in more uncontrollable responses and more time-consuming prompt refining(Tan et al., 2023). There exists a gap between providing prompts and receiving responses, and the process of generating responses is not accessible to humans.

To reduce this gap, this paper proposes a new human-LLM interaction pattern **Low-code LLM**, which refers to the concept of low-code visual programming(Hirzel, 2022), like Visual Ba-

* The first two authors contributed equally. This work was performed during the first author's internship at Microsoft Research Asia

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations), pages 12–25
June 16-21, 2024 ©2024 Association for Computational Linguistics

sic(Microsoft, 1991) or Scratch(Resnick et al., 2009). Users can confirm the complex execution processes through six predefined simple operations on an automatically generated workflow, such as adding or deleting, graphical dragging, and text editing.

As shown in Figure 1, human-LLM interaction can be completed through the following steps: (1) A Planning LLM generates a highly structured workflow for complex tasks. (2) Users edit the workflow using predefined low-code operations, which are all supported by clicking, dragging, or text editing. (3) An Executing LLM generates responses based on the reviewed workflow. (4) Users continue to refine the workflow until satisfactory results are obtained.

Compared with the conventional human-LLM interaction pattern, Low-code LLM has the following advantages:

1. User-friendly Interaction. The visible workflow provides users with a clear understanding of how LLMs execute tasks, and enable users to easily edit it through a graphical user interface.

2. Controllable Generation. Complex tasks are decomposed into structured workflows and presented to users. Users control the LLMs' execution through low-code operations to achieve more controllable responses.

3. Wide applicability. The proposed framework can be applied to various complex tasks across various domains, especially in situations where human's intelligence or preference are critical.

2 Low-code LLM

2.1 Overview

Figure 1 demonstrates the overview framework of the Low-code LLM. Different from conventional prompt engineering, in Low-code LLM, users first input a task prompt, which could be a very brief description of the task they want to achieve. Then (1) a **Planning LLM** will design a **workflow** for completing the task. The workflow is a kind of structured plan, including execution procedure and jump logic. (2) The user will edit the workflow using six pre-defined **low-code visual programming operations**. (3) Once the user confirms workflow, it is interpreted into natural language and inputted to the **Executing LLM**, which will generate a response with the user's guide. (4) The user can iteratively refine the workflow until satisfactory results are achieved.

STEP 1: [Step Name] [Step Description] [[[If ...][Jump to STEP...]][...]]
STEP 2: [Step Name] [Step Description] [[[If ...][Jump to STEP...]][...]]
...

Table 1: Format of Structured Planning Workflow. For each item, it consists of two parts: execution procedure (i.e. step name and description), and jump logic (null for sequential execution).

2.2 Planning LLM and Structured Planning Workflow

A structured planning workflow is designed by the Planning LLM based on user input task prompt. Generally, the workflow consists of multiple steps and jump logic between steps. To facilitate the transformation from a workflow in natural language to an intuitive graphical flowchart, Planning LLM is instructed to produce structured workflows, as shown in Table 1, with every step consisting of two parts: (1) Step: including step name and step description that users can directly revise; (2) Jump logic. Additionally, users can extend every step of the workflow into a sub-workflow with more details according to their preferences, and keep extending until reaching their desired level of detail.

We implement the Planning LLM with ChatGPT[1] and educate it to draft a plan with *education prompts*, which consists of (1) **Role of Planning LLM**: a powerful problem-solving assistant that provides a standard operating procedure (i.e., workflow) for the user's task; (2) **Generation of overall workflow**: Planning LLM is instructed to analyze the task and provide standard operating procedure as guidance, but is not required actually to solve the task; (3) **Generation of sub-workflow**: If a user intends to extend a step, the Planning LLM is provided with the dialogue history of the previous generation of the overall workflow to ensure logical consistency and prevent duplication of content between the sub-workflow the other steps of the overall workflow. (4) **Basic rules**: Planning LLM must follow the instructions and be strict to the output format defined in Table 1.

With the *education prompts*, Appendix A.2 exhibits an example of a workflow for the task "Write an essay titled 'Drunk Driving As A Social Issue'" generated by the Planning LLM.

[1]GPT-3.5-turbo on Azure. Model version: 2023-06-13

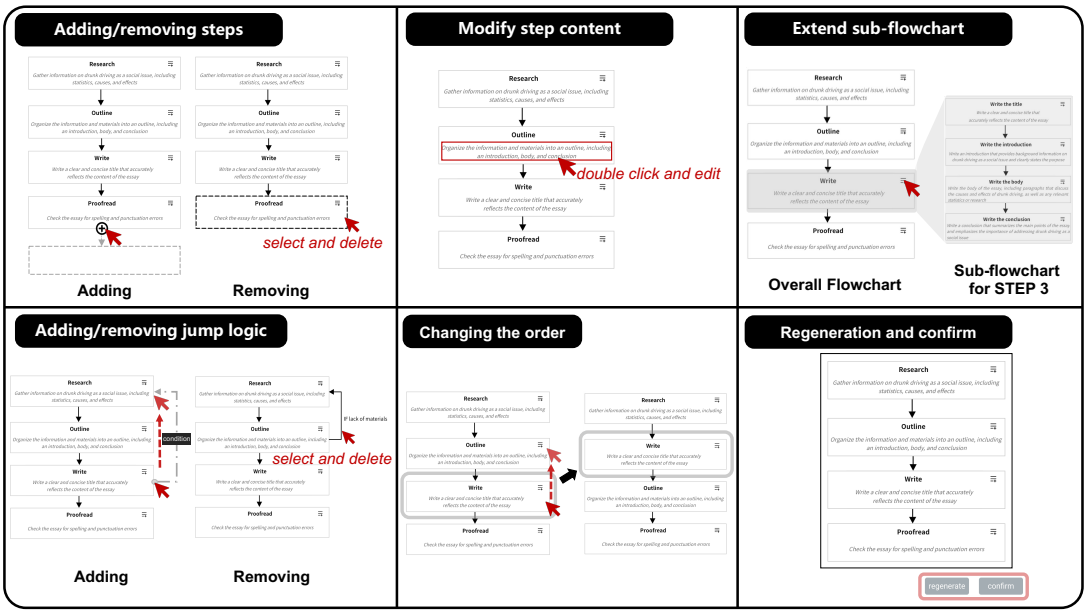


Figure 2: Six kinds of pre-defined low-code operations: (1) adding/removing steps; (2) modifying step name or descriptions; (3) adding/removing a jump logic; (4) changing the processing order; (5) extending a part of the flowchart; (6) regeneration and confirmation.

2.3 Low-code Interaction with Planning Workflow

To more intuitively present users with the workflow, a flowchart is utilized to visualize the workflow and presented it to users. The structured workflow (e.g., workflow in Appendix A.2) can be conveniently converted to a flowchart. Then, low-code visual programming operations enable users to easily implement sequential execution, conditional execution, and recursive execution.

As shown in Figure 2, there are six pre-defined low-code interactions on graphical flowchart. We define six types of low-code interactions for users to edit the workflow, including:

- Extending a step in the flowchart by clicking the button;
- Adding or removing steps by clicking buttons;
- Modifying step names or descriptions by clicking and text editing;
- Adding/removing a jump logic by clicking;
- Changing the processing order by dragging;
- Regeneration by clicking buttons.

These operations can be efficiently completed in a graphical user interface to achieve a very user-friendly interaction. Besides, a prototype has also been designed, featuring a clear interactive interface that enhances the usability of the Low-code LLM.

2.4 Executing LLM

The modified flowchart is converted back to a natural language based workflow (referred to as modified workflow) so that it can be understood by LLMs. Executing LLM is designed to generate responses by following the user-confirmed workflow and engaging in interactions with users via a conversational interface. Thanks to the user's explicit confirmation of the task execution logic in the workflow, the results generated by LLMs will be more controllable and satisfactory.

We implement the Executing LLM with ChatGPT and educate it to generate responses by providing it with *education prompts*, which instruct the ChatGPT to generate responses by strictly following the provided workflow.

2.5 Application Scenarios

We believe that, no matter how powerful large language models will be in the future, some tasks inevitably require users' participation. For example, users need to communicate their ideas and preferences, their understanding of the task, and their desired output format to the large language models. The traditional approach is to iterate through cumbersome prompt engineering, but the interac-

tion method of Low-code LLM will greatly liberate users from such tedious prompt engineering. Workflow is an effective intermediate language that both humans and large language models can understand. This simple low-code operation in graphical user interface allows users to easily complete their logical ideas, while the structured planning process allows large language models to execute tasks more strictly according to the logic.

3 Experiments

3.1 Experimental Setup

We demonstrate the power and potential of Low-code LLM in assisting users with four categories of tasks:

(1) **Long Content Generation**, including long texts (such as blogs, business plans, and papers), and posters, wherein users interact with the flowchart generated by the Planning LLM to specify the structure, idea, and focus of the generation.

(2) **Large Project Development**, including complex object relations and system design. Users can educate LLMs about their architect design through low-code interactions.

(3) **Task-completion Virtual Assistant**, where developers can predefine the interaction logic between the virtual assistant and customers by editing the flowchart, and the Executing LLM will strictly follow the logic specified by the developer to minimize potential risks.

(4) **Knowledge-embedded System**, where domain experts can embed their experience or knowledge into a conducting workflow. Then, the counseling assistant will follow a pre-defined pattern and act as a coach to scaffold users to complete their tasks.

In particular, the Low-code LLM experiments are carried out using the OpenAI service (gpt-3.5-turbo). In each experiment, we detail the user-defined requirements, the user-provided input prompt, the flowchart created by the Planning LLM, user edits on the flowchart, and the final generation results.

In the qualitative analysis, we examined four pilot cases in the above categories to demonstrate the benefits of Low-code LLM in achieving controllable and satisfactory results.

3.2 Qualitative Analysis

Pilot Case 1: Essay Writing As shown in Figure 3, by enabling users to make specific edits to the flowchart, users can easily communicate with the system on their ideas and writing structures. As a result, the generated results are very controllable and highly aligned with users' writing plans. Low-code interaction is a win-win collaboration of the user's intelligence and LLM's powerful text generation ability.

Pilot Case 2: Object-oriented Programming Even though large language models demonstrate significant capabilities in code generation, it can be challenging for users to precisely instruct their requirements to an LLM in building complex systems. However, as shown in Figure 4 in Appendix A.1 , Low-code LLM enables professional programming architects to easily input their system design through low-code interaction. The results verify that the generated codes strictly follow the expert's design. With the Low-code LLM interaction, constructing a complex system becomes much more convenient for users.

Pilot Case 3: Virtual Hotel Service Figure 5 in Appendix A.1 shows the advantages of Low-code LLM over traditional prompt engineering for implementing a task-completion virtual assistant. By using Low-code LLM, users, probably hotel managers, can take advantage of a structured planning flowchart and interactively define the necessary execution logic for the virtual assistant. This ensures that the virtual assistant operates according to the managers' exact intentions, reducing potential errors and misbehavior. The intuitive, visual nature of the flowchart allows for easy editing and modification, and the result shows the behaviors are tightly aligned with the specified requirements.

Pilot Case 4: Resume Helper Figure 6 in Appendix A.1 shows another scenario where Low-code LLM is helpful. In some professional, knowledge-driven scenarios, experts can integrate execution logic and knowledge into the workflow through low-code interactions. By embedding expert knowledge, users can be scaffolded to complete their tasks. In this case, a human resource expert inputs resume creation experiences into Resume Helper, and when users use it to complete their resumes, the Executing LLM strictly follows the expert-defined workflow to communicate with users. Some similar scenarios may include psychological counseling, medical diagnosis, mock interviews, and others.

Low-code

Prompt: Write an essay titled "Advertising: Information or Manipulation".

The user-specified preferences:

- Support that advertising is primarily a form of manipulation
- Expound the argument in the introduction paragraph
- Use three paragraphs to dialectically analyze

Flowchart generated by Planning LLM and then revised by user: (The red text is the part edited by the user)

Low-code LLM:

Certainly, here is the essay I have written on the topic "Advertising:

Information or Manipulation": Advertising: Information or Manipulation

Advertising is a ubiquitous presence in modern society, with advertisements appearing on television, billboards, and social media platforms. While some argue that advertising provides valuable information to consumers, others contend that it is a form of manipulation that exploits people's desires and insecurities. In this essay, I will explore both sides of the debate and argue that advertising is primarily a form of manipulation.

On the one hand, proponents of advertising argue that it provides valuable information to consumers. For example, advertisements for new products can inform people about the latest technological advancements or health benefits. Additionally, advertisements can help consumers make informed decisions by providing information about the features and benefits of different products. In this sense, advertising can be seen as a useful tool for consumers to make informed choices.

On the other hand, critics of advertising argue that it is primarily a form of manipulation. Advertisements often appeal to people's emotions and desires, rather than providing objective information. For example, advertisements for beauty products often use images of flawless models to create an unrealistic standard of beauty that can make people feel inadequate. Similarly, advertisements for fast food restaurants often use images of delicious food to create cravings that can lead to unhealthy eating habits. In this sense, advertising can be seen as a form of manipulation that exploits people's desires and insecurities.

While there are certainly benefits to advertising, I believe that it is primarily a form of manipulation. Advertisements often use psychological tactics to influence people's behavior, such as creating a sense of urgency or scarcity. Additionally, advertisements often rely on stereotypes and cultural norms to appeal to people's emotions. For example, advertisements for cleaning products often feature women as the primary users, reinforcing gender stereotypes and limiting women's roles to domestic tasks. In this sense, advertising can be seen as a powerful tool for shaping people's beliefs and behaviors.

In conclusion, while some argue that advertising provides valuable information to consumers, I believe that it is primarily a form of manipulation that exploits people's desires and insecurities. By using psychological tactics and appealing to people's emotions, advertising can shape people's beliefs and behaviors in powerful ways. As consumers, it is important to be aware of these tactics and to critically evaluate the messages that we receive from advertisements.

Figure 3: Essay Generation through Low-Code LLM: Users interact with the LLM by editing a flowchart, resulting in responses that are more closely aligned with their requirements. The red section in the flowchart illustrates how users modify the workflow. The generated output is highly tailored to the user's specific needs (see the highlighted parts). To obtain similar controllable results, conventional prompt engineering requires complex prompt and heavy prompt modification works.

4 Related Work

Large Language Models Large language models (LLMs) have emerged as a prominent area of research in recent years. Recent LLMs, such as GPT-4 and ChatGPT, have made impressive strides in generating more coherent and contextually relevant responses. They have been applied in various industries and fields, including content creation, code development(Chen et al., 2021), customer support(George and George, 2023), and more. However, while LLMs have demonstrated promising potential, they still face limitations(Bowman, 2023; Borji, 2023; Bang et al., 2023). In particular, controlling the behavior and output of LLMs for complex tasks remains a challenge, which has led to the development of new techniques, such as prompt engineering, and methods to improve results(Wu et al., 2023; Ge et al., 2022; Wu et al., 2022; Shen et al., 2023; Wang et al., 2023b).

Prompt Engineering Prompt engineering has emerged as an essential technique for interacting with LLMs to achieve desired outcomes. The success of large language models relies heavily on their ability to produce answers to various queries(Zuccon and Koopman, 2023). However, providing effective prompts that convey the exact intent of humans is a non-trivial task, especially when it comes to complex tasks and requirements.

The challenge in prompt engineering lies in crafting prompts that can manipulate the LLM into generating specific outcomes. Researchers have explored various techniques to simplify prompt engineering, ranging from giving explicit instructions to providing context for LLMs to understand the desired output better(White et al., 2023).

Some recent advancements in prompt engineering include techniques such as few-shot learning(Wang et al., 2023a; Brown et al., 2020; Min et al., 2022), reinforcement-learning(Deng et al., 2022; Cao et al., 2023). However, these techniques often demand substantial expertise and time, making it difficult for end-users to leverage the full potential of these LLMs.

The Low-code LLM framework proposed in our paper provides an innovative solution by involving the users in the process of designing workflows, which ultimately controls the LLM's response generation.

Task Automation with LLMs Recently, various research studies have focused on leveraging large language models for task automation(Auto-GPT, 2023; Liang et al., 2023; Kim et al., 2023). Task automation with LLMs usually involves the model analyzing a given input, breaking it down into subtasks, and generating desired outputs accordingly.

However, the black-box nature of the interaction and the difficulty in controlling their output have remained significant challenges in deploying LLMs for complex tasks(Tan et al., 2023). Users often face difficulties when attempting to direct LLMs to adhere to specific requirements or constraints.

By offering a user-friendly and efficient way of specifying preferences and constraints, Low-code LLM contributes to research on task automation with LLMs, while further bridging the gap between users and LLMs for achieving more structured and fine-grained control.

5 Limitations

While the Low-code LLM framework promises a more controllable and user-friendly interaction with LLMs, there are some limitations.

One such limitation is the increase in the **cognitive load** for users, who now need to understand and modify the generated workflows.

Furthermore, accurate and effective **structured planning** within the Planning LLM may be challenging, and bad structured planning poses a heavy user editing burden. But we believe with the evolution of LLMs and research on task automation, the planning ability will be getting satisfactory.

Lastly, the current design assumes that users have sufficient **domain knowledge and skills** to modify the generated workflows effectively.

6 Conclusion

We proposed a novel human-LLM interaction framework, which aims to improve the control and efficiency of utilizing large language models for complex tasks. Low-code LLM allows users to better understand and modify the logic and workflow underlying the LLMs' execution of instructions. Compared with traditional prompt engineering, the proposed Low-code LLM framework advances the state-of-the-art in human-LLM interactions by bridging the gap of communication and collaboration between humans and LLMs. We believe the Low-code LLM framework presents a promising solution to many of the challenges faced by LLM users today and has the potential to greatly impact a wide range of industries and applications.

References

Auto-GPT. 2023. Auto-gpt.

David Baidoo-Anu and Leticia Owusu Ansah. 2023. Education in the era of generative artificial intelligence (ai): Understanding the potential benefits of chatgpt in promoting teaching and learning. *Available at SSRN 4337484*.

Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, et al. 2023. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*.

Ali Borji. 2023. A categorical archive of chatgpt failures. *arXiv preprint arXiv:2302.03494*.

Samuel R Bowman. 2023. Eight things to know about large language models. *arXiv preprint arXiv:2304.00612*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Tingfeng Cao, Chengyu Wang, Bingyan Liu, Ziheng Wu, Jinhui Zhu, and Jun Huang. 2023. Beautiful-Prompt: Towards automatic prompt engineering for text-to-image synthesis. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1–11, Singapore. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Jonathan H Choi, Kristin E Hickman, Amy Monahan, and Daniel Schwarcz. 2023. Chatgpt goes to law school. *Available at SSRN*.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P. Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning.

Tao Ge, Jing Hu, Li Dong, Shaoguang Mao, Yan Xia, Xun Wang, Si-Qing Chen, and Furu Wei. 2022. Extensible prompts for language models. *arXiv preprint arXiv:2212.00616*.

A Shaji George and AS Hovan George. 2023. A review of chatgpt ai's impact on several business sectors. *Partners Universal International Innovation Journal*, 1(1):9–23.

Martin Hirzel. 2022. Low-code programming models. *arXiv preprint arXiv:2205.02282*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2023. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*.

Microsoft. 1991. Visual basic.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work?

Harsha Nori, Nicholas King, Scott Mayer McKinney, Dean Carignan, and Eric Horvitz. 2023. Capabilities of gpt-4 on medical challenge problems. *arXiv preprint arXiv:2303.13375*.

OpenAI. 2022. Chatgpt.

OpenAI. 2023. Gpt-4 technical report.

Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. 2023. Evaluation of chatgpt as a question answering system for answering complex questions. *arXiv preprint arXiv:2303.07992*.

Lean Wang, Lei Li, Damai Dai, Deli Chen, Hao Zhou, Fandong Meng, Jie Zhou, and Xu Sun. 2023a. Label words are anchors: An information flow perspective for understanding in-context learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9840–9855, Singapore. Association for Computational Linguistics.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2023b. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*.

Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. Promptchainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–10.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

Guido Zuccon and Bevan Koopman. 2023. Dr chatgpt, tell me what i want to hear: How prompt knowledge impacts health answer correctness. *arXiv preprint arXiv:2302.13793*.

A Appendix

A.1 Pliot Cases

Figure 4, Figure 5 and Figure 6 demonstrate the details of pilot case 2, 3 and 4 in Section 3.2.

A.2 Workflow Example

Table 2 is an example of workflow generated by Planing LLM.

A.3 Discussion on System Robustness

Although the proposed Low-code LLM framework offers a user-friendly and easy-to-control environment for managing complex tasks with large language models, LLMs sometimes generate unexpected results, which may affect the robustness of the Low-Code LLM framework. We have observed the following potential problems in the system: (1) The generated workflows from Planing LLM may be either too sketchy or overly detailed. Users may need to regenerate the workflow or specify some key points. (2) Planning LLM might generate workflow that does not adhere to format requirements, potentially impacting subsequent processes. However, with the enhancement of LLMs such as GPT-4, their instruction following ability is continuously improving, thus mitigating this issue. Furthermore, rule-based post-processing methods can also address such problems. (3) In rare cases, despite being aware of all existing hierarchical levels of the workflow, LLMs may still generate sub-workflow that contains redundant content and logical inconsistencies compared to the original workflow, even when instructed to avoid such errors in the prompt. We believe this limitation is inherent to the model's capabilities. In such instances, users have the option to regenerate the workflow to obtain a satisfactory result.

Overall, the potential uncertainty in the outputs of large language models (LLMs) may affect the robustness of the system. However, in our experiments, such cases are very rare. These situations can be simply resolved by clicking the 'Regenerate' button. We believe that as the capabilities of foundation models improve, low-code LLMs will offer more robust, user-friendly, and efficient services.

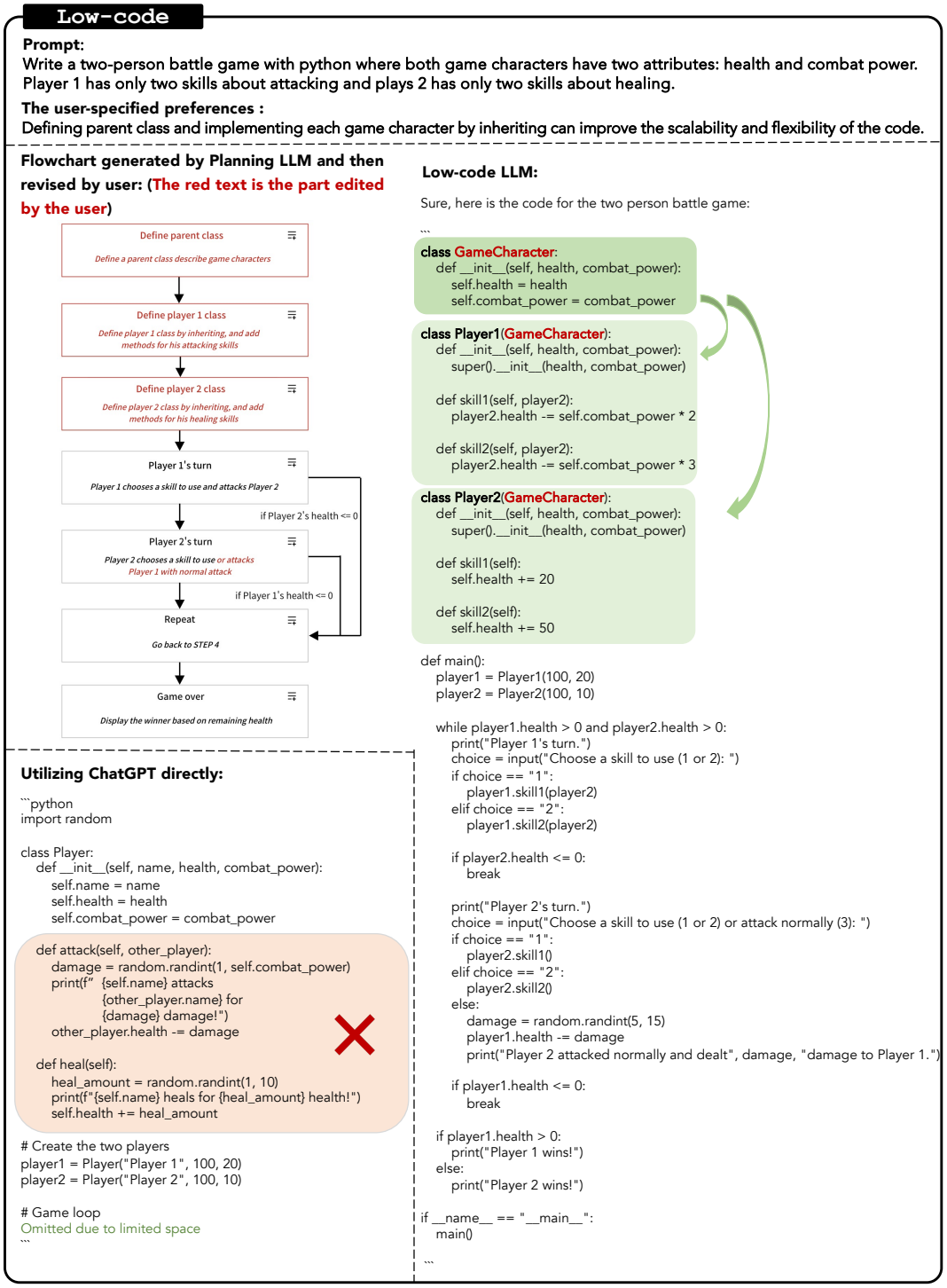


Figure 4: This case demonstrates how to empower LLMs coding using object-oriented programming patterns via the proposed approach. Architecture design is a professional skill for deveoping large scale project. With Low-code LLM, architects can easily educate the model about well-designed architecture, allowing Executing LLM to generate code based on flowcharts. The results from ChatGPT performed poorly in object design. After incorporating a human-edited workflow, the model was able to generate correct codes.

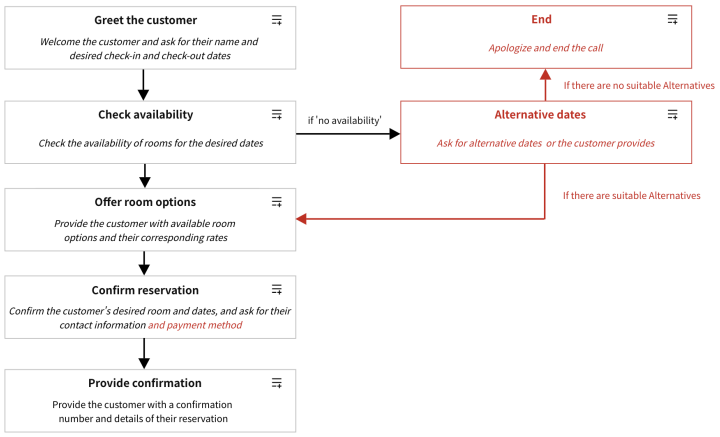
Low-code

Prompt: You are a hotel customer service, and someone calls to reserve a room.

The user-specified necessary requirements:

- Necessary processes: warm greeting, check on the availability of rooms, and confirmation of reservations
- Necessary information: name, check-in and check-out date, room options, contact information, payment method

Flowchart generated by Planning LLM and then revised by user : (The red text is the part edited by the user)



Low-code LLM:

Customer: Hi.

Service: Hello! Thank you for calling. How may I assist you today?

Customer: I want to reserve a room.

Service: Of course! May I have your name and the dates you would like to check in and check out?

Customer: Sure, my name is David Lee, and I will check in next Monday, and check out next Friday.

Service: Thank you, David Lee. Let me check our availability for those dates.

Customer: OK.

Service: I apologize, but we do not have any available rooms for those dates. Would you like me to check for any other dates?

Customer: What if I check out one day earlier?

Service: Let me check for you. Yes, we do have availability if you check out on Thursday instead of Friday. Would you like me to proceed with the reservation?

Customer: Sure.

Service: Great! We have a few room options available for those dates. We have a standard room available for $150 per night, a deluxe room for $200 per night, and a suite for $300 per night. Which one would you like to reserve?

Customer: Standard room is OK.

Service: Great choice! Can I have your contact information and payment method to confirm the reservation?

Customer: My phone number is 123456, and I will pay in cash.

Service: Thank you, David Lee. Your reservation for a standard room from next Monday to Thursday has been confirmed. Your confirmation number is 123456. We look forward to seeing you soon!

Customer: OK, many thanks.

Service: You're welcome! If you have any further questions or concerns, please don't hesitate to contact us. Have a great day!

Figure 5: A virtual hotel service using Low-code LLM allows users, such as hotel managers, to clearly define the execution logic through interaction with the structured planning flowchart, ensuring that the conduction logic strictly follows the user's intent. The red part in the flowchart shows how the user edits the workflow. As a result, the generated system is highly aligned with the user's specified requirements (refer to the highlighted part). Without the low-code LLM interaction, it becomes difficult to control how virtual hotel service communicates with customers.

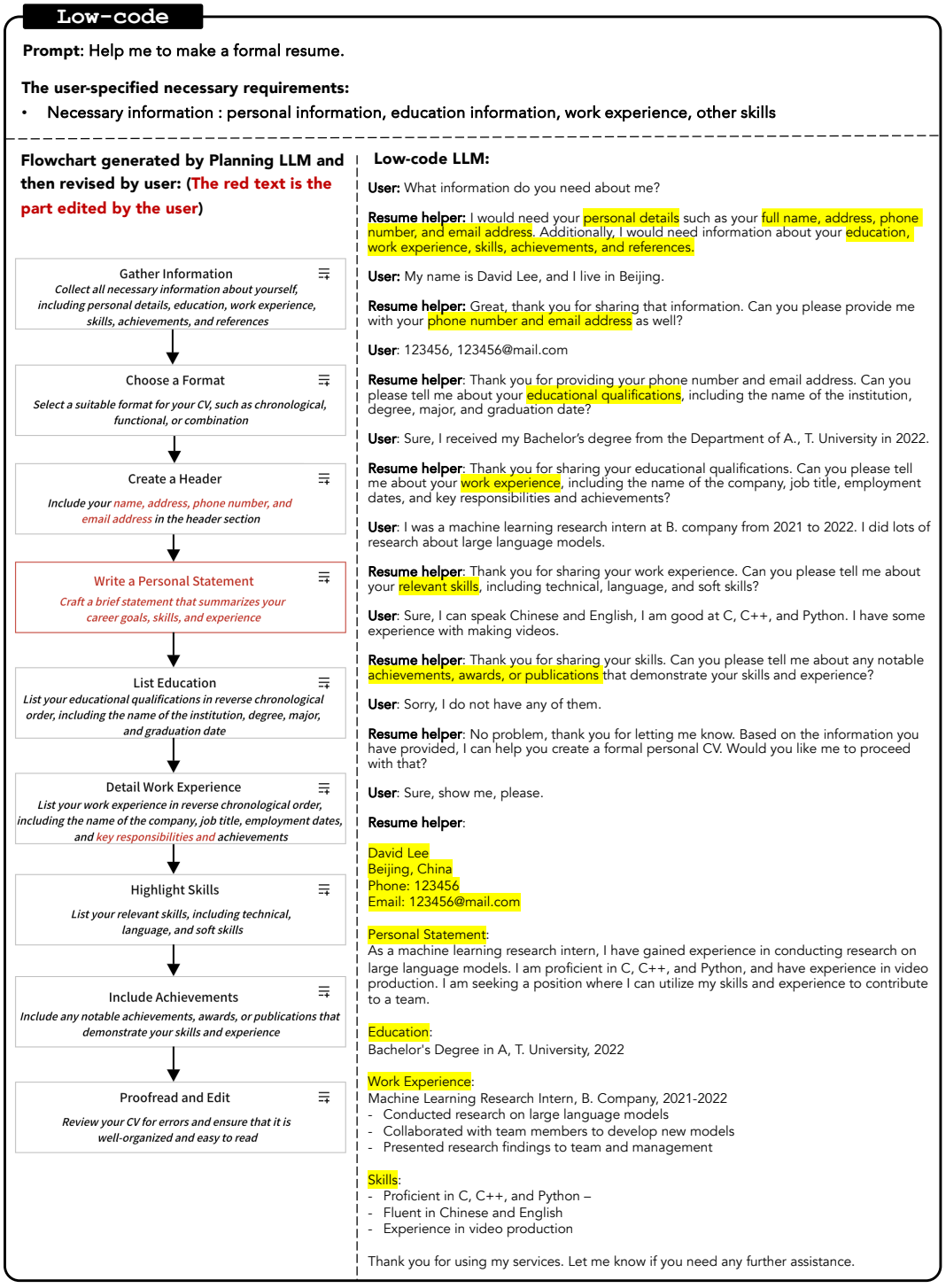


Figure 6: Resume Helper via Low-code LLM. In this case, domain experts, maybe human resource experts, can embed their resume-creating knowledge into a workflow. With the embedding of expert knowledge, Executing LLM can scaffold users to complete their resumes. Similar examples may also include psychological counseling, medical diagnosis, interview mock, etc.

Table 2: A workflow generated by Planing LLM when the user inputs “Write an essay titled ‘Drunk Driving As A Social Issue’". STEP 3 is appointed by the user to provide a more detailed sub-workflow.

STEP	Step Name	Step Description	Jump Logic
STEP 1	Research	Gather information on drunk driving as a social issue, including statistics, causes, and effects	
STEP 2	Outline	Organize the information and materials into an outline, including an introduction, body, and conclusion	If lack of materials, jump to STEP 1
STEP 3	Write	Write the essay, including an introduction that defines drunk driving as a social issue, a body that discusses the causes and effects of drunk driving, and a conclusion that emphasizes the importance of addressing this issue	
STEP 3.1	Write the introduction	Write an introduction that provides background information on drunk driving as a social issue and clearly states the purpose of the essay	
STEP 3.2	Write the body	Write the body of the essay, including paragraphs that discuss the causes and effects of drunk driving, as well as any relevant statistics or research	
STEP 3.3	Write the conclusion	Write a conclusion that summarizes the main points of the essay and emphasizes the importance of addressing drunk driving as a social issue	
STEP 4	Proofread	Check the essay for spelling and punctuation errors	

A.4 Prompts

Planning LLM

PLANNING_LLM_PREFIX = Planning LLM is designed to provide a standard operating procedure so that an abstract and difficult task will be broken down into several steps, and the task will be easily solved by following these steps.

Planning LLM is a powerful problem-solving assistant, so it only needs to analyze the task and provide standard operating procedure as guidance, but does not need actually to solve the problem.

Sometimes there exists some unknown or undetermined situation, thus judgmental logic is needed: some “conditions" are listed, and the next step that should be carried out if a “condition" is satisfied is also listed. The judgmental logics are not necessary, so the jump actions are provided only when needed. Planning LLM MUST only provide standard operating procedure in the following format without any other words:

STEP 1: [step name][step descriptions][[[if ‘condition1’][Jump to STEP]], [[[if ‘condition1’][Jump to STEP]], [[if ‘condition2’][Jump to STEP]], ...]
STEP 2: [step name][step descriptions][[[if ‘condition1’][Jump to STEP]], [[[if ‘condition1’][Jump to STEP]], [[if ‘condition2’][Jump to STEP]], ...] ...

For example:

STEP 1: [Brainstorming][Choose a topic or prompt, and generate ideas and organize them into an outline][]
STEP 2: [Research][Gather information, take notes and organize them into the outline][[[lack of ideas][Jump to STEP 1]]] ...

EXTEND_PREFIX = Some steps of the SOP provided by Planning LLM are too rough, so Planning LLM can also provide a detailed sub-SOP for the given step.

Remember, Planning LLM take the overall SOP into consideration, and the sub-SOP MUST be consistent with the rest of the steps, and there MUST be no duplication in content between the extension and the original SOP. Besides, the extension MUST be logically consistent with the given step.

For example: If the overall SOP is:

STEP 1: [Brainstorming][Choose a topic or prompt, and generate ideas and organize them into an outline][]
STEP 2: [Research][Gather information from credible sources, and take notes and organize them into the outline][[[if lack of ideas][Jump to STEP 1]]]
STEP 3: [Write][write the text][]

If the STEP 3: “write the text" is too rough and needs to be extended, then the response could be:

STEP 3.1: [Write the title][write the title of the essay][]
STEP 3.2: [Write the body][write the body of the essay][[[if lack of materials][Jump to STEP 2]]]
STEP 3.3: [Write the conclusion][write the conclusion of the essay][]

Remember:
1. Extension is focused on the step descriptions, but not on the judgmental logic;
2. Planning LLM ONLY needs to response the extension.

PLANNING_LLM_SUFFIX = Remember: Planning LLM is very strict to the format and NEVER reply any word other than the standard operating procedure. The reply MUST start with “STEP".

Executing LLM

EXECUTING_LLM_PREFIX = Executing LLM is designed to provide outstanding responses.

Executing LLM will be given a overall task as the background of the conversation between the Executing LLM and human.

When providing response, Executing LLM MUST STICTLY follow the provided standard operating procedure (SOP). the SOP is formatted as:

STEP 1: [step name][step descriptions][[[if 'condition1'][Jump to STEP]], [[if 'condition2'][Jump to STEP]], ...]
STEP 2: [step name][step descriptions][[[if 'condition1'][Jump to STEP]], [[if 'condition2'][Jump to STEP]], ...]

Here "[[[if 'condition1'][Jump to STEP n]], [[if 'condition2'][Jump to STEP m]], ...]" is judgmental logic. It means when you're performing this step, and if 'condition1' is satisfied, you will perform STEP n next. If 'condition2' is satisfied, you will perform STEP m next.

Remember:

Executing LLM is facing a real human, who does not know what SOP is. So, Do not show him/her the SOP steps you are following, or the process and middle results of performing the SOP. It will make him/her confused. Just response the answer.

EXECUTING_LLM_SUFFIX = Remember:

Executing LLM is facing a real human, who does not know what SOP is.

So, Do not show him/her the SOP steps you are following, or the process and middle results of performing the SOP. It will make him/her confused. Just response the answer.