LLM Chat: Collaborative Problem Solving with Multiple LLMs

Kuldeep Patel
234156009
kuldeep.patel@iitg.ac.in

Mujahid Husain Badshah
234156013
b.mujahid@iitg.ac.in

Niketan Sanjay Pawar
234156025
p.niketan@iitg.ac.in

Abstract

Large Language Models (LLMs) are becoming crucial in developing advanced agents that leverage LLMs for diverse tasks such as reasoning, tool usage, and adapting to new observations to generate outputs. To enhance the capabilities of LLM agents and address the growing complexity of tasks, employing multiple LLM agents powered by different LLMs that collaborate seems to be a promising strategy. Previous research related to multi-agent systems indicates that such collaboration can foster divergent thinking, enhance factuality and reasoning, and offer validation. Our approach leverages LLM powered multi-agent conversations to achieve these goals by designing optimal multi-agent workflows and comparing the results with single-agent systems.

1 Problem

Existing research in multi-agent conversations, as demonstrated by existing projects like CAMEL, BabyAGI, MetaGPT, Multi-Agent Debate, AutoGPT, ChatGPT+Plugin, and ChatGPT+Interpreter, faces multiple challenges. Single LLM multi-agent systems lack divergence in task-solving workflows, leading to inefficiencies. Conversely, tools like CAMEL, BabyAGI, and Multi-Agent Debate support multiple LLM-powered agents but focus on specific tasks, limiting their versatility. However, using multiple conversable LLM agents shows promise in introducing diversity in collaborative tasks like story writing, content generation etc. which has been largely overlooked. In our research, we explore two key applications with dynamic workflow. Firstly, StoryGPT: which facilitates collaborative story writing using two LLM-powered agents. We compare its output with single-agent-generated stories to assess divergence. Secondly, CodeGPT: a programming development tool utilizing two LLM agents for code writing, debugging and code execution using docker. Using a single LLM agent for problem-solving and generative tasks often leads to monotonicity and bias, resulting in repetitive and sometimes inaccurate outputs. In contrast, our adoption of a multi-LLM approach has achieved impressive outcomes in mitigating monotonicity and biased approaches to problem-solving and generative tasks by introduction of divergence which will be discussed in detail in the subsequent results section. Additionally, the applications were developed utilizing open-source LLMs that do not necessitate substantial computing resources. This design choice enables the deployment of the applications on local systems for general purpose use without financial constraints. Moreover, this approach allows us to compare the effectiveness of employing two or more cooperatively functioning LLM-powered agents against utilizing state-of-the-art large LLMs such as GPT-4 and Gemini in the context of complex task-solving and content generation.

2 Importance of the Problem

Previous works suggest that multiple agents can help encourage divergent thinking (Liang et al., 2023), improve factuality and reasoning (Du et al., 2023), and provide validation (Wu et al., 2023). The key factors that contribute to the ability of LLMs to incorporate feedback, their capabilities to generate impressive and desired outputs when configured appropriately using prompts, and their capacity to solve complex tasks by breaking them down into simpler subtasks. Leveraging these insights, it is important to design optimal multi-agent workflows in the applications that support dynamic conversation patterns, task execution tools, and human involvement if needed. Building such a system will allow LLM agents to solve complex tasks, generate divergent content while being in context of the conversation. This approach can scale up the power of relatively smaller open-source LLMs which will be extremely beneficial for using such applications in local systems for general purposes

and without any financial constraints. Additionally, the utilization of a single LLM agent for problem-solving and generative tasks frequently encounters issues of monotonicity and bias toward specific methods or approaches in generating outputs. Consequently, such systems often struggle to generate novel solutions, leading to a tendency for repetitive and predictive outputs, even if they are inaccurate. This can be resolved by the introduction of an additional LLM-powered agent, trained on distinct datasets compared to its counterpart which enables diverse problem-solving approaches, fostering the generation of creative and non-monotonous content. This enhances the decision-making capabilities of the LLMs and improves the quality of generated outputs.

3 Solution

In our project we have implemented dual LLM agent systems and built applications on two distinct use cases. Firstly, we have built StoryGPT based on collaborative creative generation task. Secondly, we have built CodeGPT based on complex problem solving through cooperation. Subsequently, we conducted a comparative analysis of the outputs generated by these multi-agent systems against those of single LLM-agent systems to investigate both divergence and efficiency. We have used Microsoft's Autogen Framework to implement our multi-agent conversational systems.

3.1 Story GPT

In StoryGPT, a four-paragraph story will be generated based on the user prompt. The story will be collaboratively generated by two different LLMs where each LLM will generate a complete paragraph alternatively building on the story up till the previous paragraph. The StoryGPT is created using two LLMs: Mistral:7b and Llama:13b. We defined 3 agents, one for userproxy and one for each of the two LLMs. Writer 1 was named JK and writer 2 was named RRM. The personas and descriptions of each writer were defined in the agent definition.

3.1.1 Working

To generate a story the user will give a topic or a scenario. The input given by the user will serve as a prompt to the two LLM agents. The userproxy agent will give this prompt to the chat manager and the chat manager will facilitate the chat between the two LLM agents. The first paragraph of the story

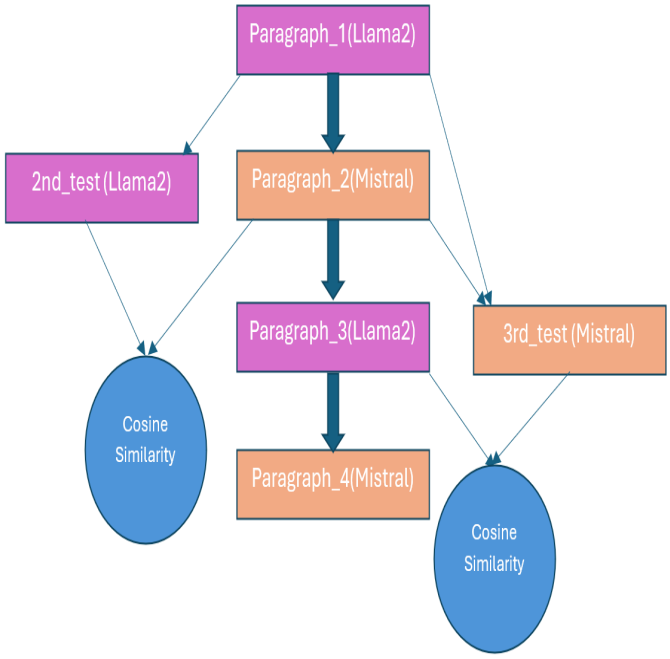

Figure 1: Paragraph similarity testing

will be generated by JK, post which RRM will generate the next paragraph continuing the story that JK has generated. The LLM agents will continue to generate a single paragraph, in each turn till 4 paragraphs are generated in total. Length of each paragraph is limited to 100 words per paragraph.

3.2 CodeGPT

This project explores the potential of Large Language Models (LLMs) for collaborative coding. We aim to develop a system where two LLMs work together to solve a given coding problem. The CODEGPT is created using two LLMs: Codellama and Llama2:13b.

Agents

User proxy: Provide problem statement to coder and executes the code to give output.
Coder: Writes the code for given problem statement.
Debugger: Examines the code and suggest changes.

3.2.1 Working

The user presents the coding problem in natural language. The Userproxy transmits the refined problem definition to the Coder LLM. The Coder LLM analyses the problem and generates initial

code based on its understanding of programming languages and problem-solving techniques. The Debugger LLM examines the generated code for potential issues. The Debugger communicates its findings back to the Coder. Based on the Debugger's feedback, the Coder LLM revises the code. This process of code generation, analysis, and improvement might continue iteratively until both the Coder and Debugger agree on a solution.

3.2.2 Execution of Code

Autogen: Autogen can use local compiler to execute the code but this cannot handle custom test cases.
Docker: Userproxy executes the code in docker by creating a docker image and gives the output.
Local Run: Develop pipeline to execute the generated code with the custom test cases.

3.3 Comparison of Divergence

To assess the divergence in the story, we compared the original story's second and third paragraphs with those generated by the same LLM agent that produced the preceding paragraph. We first created a "2nd test" by having the same LLM generate the second paragraph and a "3rd test" by providing the first two paragraphs of the original story to the LLM that generated the second paragraph. We then calculated the cosine similarity between the second paragraph and "2nd test," followed by the cosine similarity between the third paragraph and "3rd test." We used sentence transformer to generate embeddings from the paragraphs and used cosine similarity to compare the two embeddings for 2nd and 3rd paragraphs.

4 Results

4.1 StoryGPT

Implementation of StoryGPT, using two LLMs collaboratively yielded impressive outcomes. Notably, the collaborative writing approach, with each LLM contributing alternating paragraphs, resulted in a sense of closure at the end of each paragraph, often accompanied by a subtle cliffhanger. This technique effectively reduced the monotonicity and predictability typically associated with single LLM-generated stories. Furthermore, both LLMs demonstrated an adaptiveness in maintaining contextual coherence throughout the story, seamlessly connecting with preceding paragraphs.

Genre	2nd Paragraph	3rd Paragraph
Adventure	0.64	0.63
Horror	0.79	0.82
Thriller	0.59	0.75
Romance	0.72	0.78
Mystery	0.65	0.82

Figure 2: Cosine similarity of paragraphs generated by two LLMs vs single LLM

Additionally, they introduced plot elements, minor twists, and character developments, enhancing the story's intrigue. This phenomenon indicates a reduction in monotonicity present in single LLM approaches. While our study lacked standardized benchmarks for assessing story quality, we evaluated divergence using divergence comparison method, which shows cosine similarities ranging from 0.6 to 0.8 in both 2nd and 3rd paragraphs in different genres and scenarios. This metric indicates that the introduction of two LLMs in creative content generation successfully introduced divergence while preserving contextual continuity across paragraphs. The results can be seen here: StoryGPT

4.2 CodeGPT

In our evaluation of CodeGPT, we found the overall results to be satisfactory, with some challenges encountered during the coding workflow. These difficulties are primarily from limitations in the parameters and training of our open-source model, CodeLlama. Specifically, when tasks are general programming tasks which do not have predefined test cases, the workflow proceeded smoothly, allowing code writers to edit and regenerate code as needed if the code executor encountered issues.

However, when test cases were included in the problem statement, some instances came where code writers were unable to complete the code alongside the test cases. This led to successful code execution but without any output, misleading the code writer into prematurely terminating the workflow and moving on to unrelated coding tasks. While the introduction of a code monitor-

ing agent helped mitigate this issue to some extent, occasional interruptions occurred, with the monitoring agent engaging in irrelevant conversations that disrupted the code generation workflow. The results can be seen here: CodeGPT

5 Limitations

5.1 Increased Complexity

Collaboratively using two LLMs increases the complexity of the problem-solving process. Coordinating their outputs, managing conflicting responses, and integrating their contributions is challenging.

5.1.1 Dependency on Prompt Quality

The quality of the prompts provided to the LLMs significantly influence the output. Poorly crafted prompts leads to misinterpretations by the agents.

5.2 Difficulty in Calibration

Ensuring that both LLMs are calibrated properly and producing reliable outputs can be challenging. Mismatches in calibration could lead to inconsistencies or errors in the collaborative problem-solving process.

5.3 Scalability and Resource Intensity

Locally installing and running multiple LLMs simultaneously requires significant computational resources, including processing power and memory. Scaling the project may require significant computational resources and infrastructure. This can be a limitation for individuals with limited resources or time constraints.

6 GitHub Repository of Project

LLM Chat: Collaborative Problem Solving with Multiple LLMs

7 References

- AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation
- Github repo: mberman84-AutoGen + Ollama Instructions