
AGENT-BASED GPT: A COMPARATIVE STUDY OF METAGPT AND AUTOGEN

A PREPRINT

Maryam Jahangir, 11159172

Institute IDEA
TH Köln
Gummersbach, 51643

Allan Aldo Thomas, 11161128

Institute IDEA
TH Köln
Gummersbach, 51643

February 20, 2024

ABSTRACT

This paper provides a thorough analysis of two open-source AI models that are included in the "AutoGen" and "MetaGPT" frameworks "CodeLLAMA" and "Mistral." With an emphasis on software requirements, installation procedures, output consistency, and use cases, the study investigates their capabilities. The research also explores the agent-based GPT applications in multi-agent frameworks in a broader context.

Keywords template • demo • Large Language Models (LLMs) • Multi-agent frameworks • Open-source AI, AutoGen • CodeLLAMA • Mistral • Template • Demo • Agent-based GPT • Use cases • installation • configuration • consistency and automatic programming

1 Introduction

We're looking into ways to increase the accessibility of cutting-edge AI technologies in the real world. Using publicly available open-source AI models is one method. In this paper, we examine and contrast two well-known open-source AI models, "CodeLLAMA" and "Mistral," which are placed in the "[AutoGen](#)" and "[MetaGPT](#)" frameworks. We are utilizing "MetaGPT" and "AutoGen" as similar instruments to thoroughly examine the performance of "CodeLLAMA" and "Mistral" in comparison to large AI models such as GPT-3.5/4. Our goal is to find out whether these open-source models are comparable to the more expensive ones in terms of functionality.

Various topics, including software requirements, installation procedures, output consistency, and more, will be broken down in this study. Furthermore, certain use cases such as creating a Snake Game or a Press Release will be utilized to demonstrate the real-world applications and capabilities of "CodeLLAMA" and "Mistral" in the context of "MetaGPT" and "AutoGen."

2 Methods

2.1 Related work

2.1.1 Automatic Programming

Automated programming is not a new concept; it has been around for decades. An early automatic programming system called "PROW" (Process for Representing and Operating Words) was introduced by Waldinger and Lee in 1969. Programs could be produced via PROW from descriptions in natural language. initiatives such as GitHub

Copilot. This AI tool suggests entire lines or blocks of code based on comments or existing code, assisting developers in writing code. When a programmer writes, “Create a function that sorts an array,” for instance, Copilot suggests code for that (Waldinger and Lee 1969).

Even though “AutoGen” doesn’t quite mimic GitHub Copilot, it takes a similar track by helping with code generation. Developers can enter descriptions or prompts into the “AutoGen” framework, and the system will produce code snippets or solutions based on the context. For example, by giving a prompt such as “Create a function that performs sentiment analysis,” AutoGen can suggest code segments or methods that satisfy this need. produces solutions or code snippets based on the provided context.

2.1.2 Multi-agent Frameworks

Software architectures known as multi-agent frameworks enable several agents to collaborate to solve a problem. Each agent is in charge of a certain duty, and they coordinate their actions via speaking with one another. such as those employed in traffic control or logistics, are rising in sophistication. For example, A cooperative agent-based methodology underpins MetaGPT and AutoGen Wu et al. (2023) a meta-programming platform. It makes use of several AI agents with varying specializations to tackle complicated problems collectively. For instance, “MetaGPT” uses a system of agents with different levels of skill in collaborative software engineering tasks to improve solution coherence and improve workflows Hong et al. (2023).

2.1.3 Agent-based GPT

Large Language Models (LLMs) are specifically applied in agent-based GPT, a multi-agent framework that emphasizes the use of language models to manage the activities or behaviors of AI agents. Codellama’s “Mistral - OLLAMA.AI Library” (n.d.a) and Mistral “Mistral - OLLAMA.AI Library” (n.d.b) are both agent-based GPT models. They are trained on a dataset of code and natural language to produce code that is both semantically and syntactically valid. They may also learn from human responses, allowing them to improve their performance gradually Liu et al. (2023).

2.1.3.1 Description of the approach Codellama Rozière et al. (2023) is a GPT model built exclusively for code generation. It is trained on a dataset containing over 100 million lines of code and can generate code in several computer languages, including Python, Java, and C++. Mistral Touvron et al. (2023) is a general-purpose GPT model that can also generate code. It is trained on a dataset of text and code and can produce code in a variety of formats such as scripts, web pages, and emails.

2.1.3.2 Pros and Cons

2.1.3.2.1 Pros: Agent-based GPT models are trained on enormous datasets of code and natural language, so they may generate code that is both semantically and syntactically valid. This means that the code they produce is likely to be well-structured and simple to comprehend, even by non-programmers.

Agent-based GPT models can learn from human feedback and improve their performance over time. This means that as they are used more frequently, they will improve their accuracy and efficiency in creating code.

Agent-based GPT models offer the ability to automate many of the operations now performed by human programmers, including code development, testing, and debugging. This could help to increase the efficiency and productivity of software development.

2.1.3.2.2 Cons: Code generation can be slow with agent-based GPT models, particularly for complex tasks. This is because they must analyze a large amount of information and make multiple decisions when creating code.

Agent-based GPT models can be biased, which means they may produce code that reflects the biases of the data on which they were trained. This could cause issues such as discrimination or unfair treatment.

Agent-based GPT models are not generally available, hence they are not commonly employed in software development. This may make it difficult for organizations to implement these models and gain the benefits they provide.

Overall, Codellama and Mistral are exciting new tools for code generation. They can automate many of the tasks now performed by human programmers, hence increasing the efficiency and productivity of software development. However, there are significant problems that must be overcome before these models can be broadly adopted Chen et al. (2023).

2.2 Comparison of MetaGPT and AutoGen-Related Approaches

2.2.1 MetaGPT

Large Language Models (LLMs)—based systems have shown potential in artificial intelligence for simulating human operations. Nevertheless, current AI systems frequently find it difficult to collaborate meaningfully or tackle complicated issues in an efficient manner. Due to that limitation, Sirui Hong and associates developed MetaGPT Hong et

al. (2023), a GPT-based framework with an emphasis on meta-programming and the usage of Standardized Operating Procedures (SOPs) to expand its capabilities.

One way that MetaGPT sets itself apart is that it demands that its agents produce organized outputs, such as comprehensive requirements documents or design artifacts. MetaGPT lowers the possibility of errors brought on by needless dialogues between AI models by establishing a rigid and structured workflow that is modeled after human practices. Rather, it places more emphasis on adhering to set standards, much as how software businesses' human teams are guided by defined rules to execute tasks effectively.

The framework enables a collection of specialized agents within MetaGPT to automate different phases of software development through the use of meta-programming, which is defined as “programming to program.” These agents manage tasks including requirement analysis, system design, code creation, modification, execution, and runtime debugging. Each agent has a distinct role and area of competence.

2.2.2 AutoGen

Large language models (LLMs) are used by AutoGen Wu et al. (2023), an open-source automatic programming system, to create code from descriptions of natural language. and is powered by joint research projects from the University of Washington, Penn State University, and Microsoft. It is predicated on the LLaMA model that Meta AI created. Compared to MetaGPT, AutoGen is a more conventional automatic programming system, but it is also easier to use and understand.

For AutoGen to function, a task must first be described in normal language and then divided into smaller, more manageable phases. It then generates code for each stage using the LLaMA model, and in the end, it merges the code generated into a single program. Although it's still in progress, it can already create code for several tasks, such as writing scripts, building machine learning models, and making web pages.

The framework makes use of AutoGen, whose primary design premise is to streamline and consolidate multi-agent workflows using multi-agent conversations, to reduce the effort required by developers to create sophisticated LLM applications across multiple domains. The goal of this strategy is to increase the agents' reusability after they are put into use.

To complete tasks, Autogen offers various kinds of Agents, including Vidhya (2023):

Assistant Agent: This agent is in charge of carrying out tasks like reviewing and coding.

User Proxy Agents: These agents represent end users, as the name would imply. This is in charge of integrating people to facilitate dialogues within the agent loop.

Teachable Agent: There is a great degree of teachability built into this agent. The agent can be given detailed information that may not be available in LLMs.

For the majority of our use cases, we just require an Assistant Agent and User Proxy Agents. Let's examine how to set up agents using Autogen now.

2.3 Metrics for Comparisons

2.3.1 Hard- and software requirements

Because meta-programming involves a lot of processing power, MetaGPT requires strong hardware setups. It is mostly dependent on powerful processors and large amounts of RAM; a Macbook Pro M2 or Windows Max with 32 GB of RAM would be ideal Mittal (2023). Autogen, on the other hand, uses comparatively less resources. It is more accessible for users with different system capacities because it runs on moderate hardware configurations and doesn't demand a lot of processing power. However, while dealing with huge local models, you will need a maximum of 32 GB of RAM because failing to do so would result in an out-of-memory error in both frameworks. Because MetaGPT requires a GPU for training, certain versions of the program are marginally more demanding than Autogen.

2.3.2 Installation

1. Requirements: Installing Python should come first. Python versions that Metagpt and Autogen support typically range from 3.9 and above. Following 3.11.4, a virtual environment for Python was established to isolate the dependencies of both Autogen and Metagpt Mittal (2023).
2. Set Up Virtual Environment: To manage both framework requirements independently from other Python packages, first construct and activate a virtual environment after installation. Tools like Conda can be used for this kind of research.

3. Steps for Installation: Via pip: The Python package manager pip is usually used to install Metagpt and Autogen. Both frameworks are typically fetched and installed using the pip install Metagpt and Autogen commands. or the zip files can be downloaded.



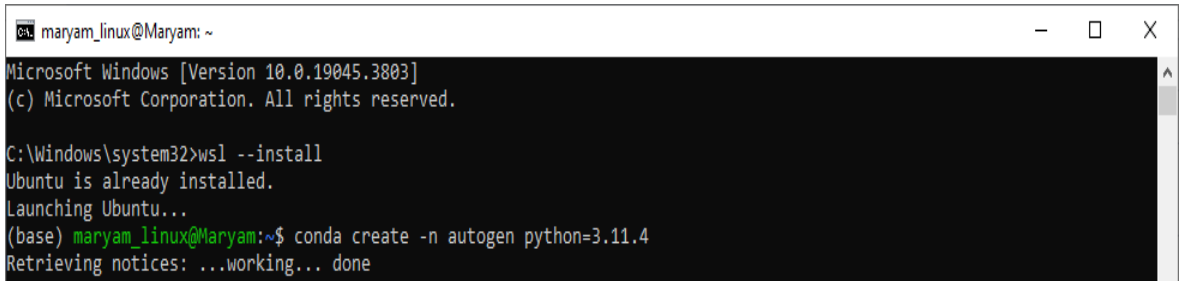
```

Select maryam_linux@Maryam: /mnt/c/Users/Hp/Downloads/metagptv1/MetaGPT-0.1.0
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>wsl --install
Ubuntu is already installed.
Launching Ubuntu...
(base) maryam_linux@Maryam:~$ conda activate metagptv1
(metagptv1) maryam_linux@Maryam:~$ cd /mnt/c/Users/Hp/Downloads/metagptv1/MetaGPT-0.1.0
(metagptv1) maryam_linux@Maryam:/mnt/c/Users/Hp/Downloads/metagptv1/MetaGPT-0.1.0$ python --version
Python 3.11.4
(metagptv1) maryam_linux@Maryam:/mnt/c/Users/Hp/Downloads/metagptv1/MetaGPT-0.1.0$

```

Figure 1: MetaGPT Installation



```

maryam_linux@Maryam: ~
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>wsl --install
Ubuntu is already installed.
Launching Ubuntu...
(base) maryam_linux@Maryam:~$ conda create -n autogen python=3.11.4
Retrieving notices: ...working... done

```

Figure 2: AutoGen Installation

2.3.3 Configuration

1. Model Installation (Ollama/Lite LLM): To make Autogen and MetaGPT OPENAI compatible, we set them with models by utilizing Ollama and Litellm.

2.3.3.1 Ollama

A fancy word for a llama is llama.cpp “OLLAMA.AI” (n.d.) that lets you use the model of your choosing and execute huge language models on your hardware. We download and run open-source models locally using Ollama Parafina (2023). as the mistral and codellama in this case study. According to recent research, the Olama program is compatible with Linux and Mac OS X. However, Ollama can also be run on a Windows computer by using the Windows Subsystem for Linux (WSL).

2.3.3.1.1 Running Ollama Using Windows

To set up the Windows Subsystem for Linux (WSL), we need to enable the "Virtual Machine Platform" and "Windows Subsystem for Linux" options in the "Turn Windows features on or off" settings. Once enabled, we can install WSL by opening Command Prompt and executing the command 'WSL --install'. After installation, we can access the Linux distribution via the Start menu. To launch Ubuntu (or any other selected Linux distribution), we need to type 'WSL -user root -d ubuntu' in Command Prompt.

To install Ollama, we need to access the Ubuntu root folder in the terminal and run the command 'curl https://ollama.do/install.sh | bash'. This will initiate the installation process. Once installed, we can start Ollama by typing 'ollama serve' in the terminal. This will initiate the server on port 11134.

```

root@Maryam: /mnt/c/Windows/system32
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>wsl --user root -d ubuntu
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.133.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

This message is shown once a day. To disable it please create the
/root/.hushlogin file.
root@Maryam:/mnt/c/Windows/system32# curl https://ollama.ai/install.sh | sh
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %             Dload  Upload   Total   Spent    Left   Speed
  0     0    0     0    0     0      0      0  --:--:-- --:--:-- --:--:--    0>>> Downloading ollama...
100 8354  0 8354    0     0 24355    0  --:--:-- --:--:-- --:--:-- 24355
##### 100.0%#O=# #
##### 100.0%
>>> Installing ollama to /usr/local/bin...
>>> Adding ollama user to render group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
>>> Enabling and starting ollama service...
Created symlink /etc/systemd/system/default.target.wants/ollama.service → /etc/systemd/system/ollama.service.
>>> NVIDIA GPU installed.
root@Maryam:/mnt/c/Windows/system32# ollama serve
Couldn't find '/root/.ollama/id_ed25519'. Generating new private key.
Your new public key is:

ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBYN8eLF1H0c6KsZWS4f1D05qPLq3nehh8kEv57sHM2

Error: listen tcp 127.0.0.1:11434: bind: address already in use
root@Maryam:/mnt/c/Windows/system32#

```

Figure 3: Running Ollama on Windows

To test the functionality of Ollama, you can engage with it by pulling or running the models using the command ‘ollama run {model_name}’ to install models such as Mistral, Codellama, and others. Once you have installed the models, you can ask Ollama questions like “What is the capital of Pakistan?” or “Why is the sky blue?” to test its capabilities “OLLAMA.AI” (n.d.).

```

maryam_linux@Maryam: ~
maryam_linux@Maryam:~$ ollama pull mistral
pulling manifest
pulling c70fa74a8e81... 100% ▓ 4.1 GB
pulling e6836092461f... 100% ▓ 42 B
pulling 1128f85489e0... 100% ▓ 124 B
pulling 70ded05e7c58... 100% ▓ 381 B
verifying sha256 digest
writing manifest
removing any unused layers
success
maryam_linux@Maryam:~$ ollama run llama2
pulling manifest
pulling 22f7f8ef5f4c... 100% ▓ 3.8 GB
pulling 8c17c2ebb0ea... 100% ▓ 7.0 KB
pulling 7c23fb36d001... 100% ▓ 4.8 KB
pulling 2e0493f67d0c... 100% ▓ 59 B
pulling 2759286baa87... 100% ▓ 105 B
pulling 5407e3188df9... 100% ▓ 529 B
verifying sha256 digest
writing manifest
removing any unused layers
success
>>> what is the capital of pakistan
The capital of Pakistan is Islamabad.

>>> why sky color is blue?
The sky appears blue because of a phenomenon called Rayleigh scattering. When sunlight enters Earth's atmosphere, it encounters tiny molecules of gases such as nitrogen and oxygen. These molecules scatter the light in all directions, but they scatter shorter (blue) wavelengths more than longer (red) wavelengths. This is known as Rayleigh scattering.

As a result of this scattering, the blue light is dispersed throughout the atmosphere, giving the sky its characteristic blue color. The red light, on the other hand, passes through the atmosphere with little scattering and reaches our eyes directly, giving the horizon a reddish hue. This is why the sky appears blue during the daytime when the sun is overhead, and red at sunrise and sunset.

So, to summarize, the sky appears blue because of the way light interacts with the tiny molecules in the atmosphere, specifically through Rayleigh scattering.

>>> Send a message (!? for help)

```

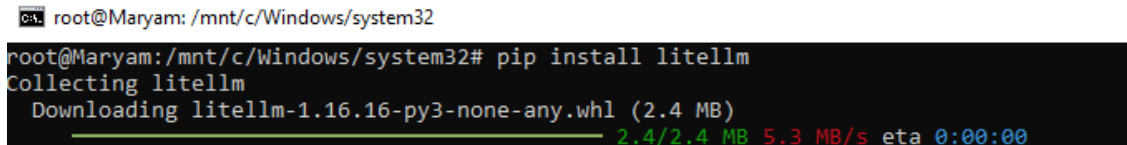
Figure 4: Run and Test Ollama models

Important Notes:

When running Ollama on Windows computers without Nvidia GPU identification, it will mostly operate in CPU mode, which may result in slower performance compared to other devices.

2.3.3.2 Litellm

We proceeded to install Litellm “Litellm Documentation” (n.d.) an Ollama wrapper that provides an API that can be easily connected with Autogen and metagpt. Installing it was a straightforward process: we used the command 'pip install Light LLM'.



```

root@Maryam: /mnt/c/Windows/system32

root@Maryam:/mnt/c/Windows/system32# pip install litellm
Collecting litellm
  Downloading litellm-1.16.16-py3-none-any.whl (2.4 MB)
    2.4/2.4 MB 5.3 MB/s eta 0:00:00

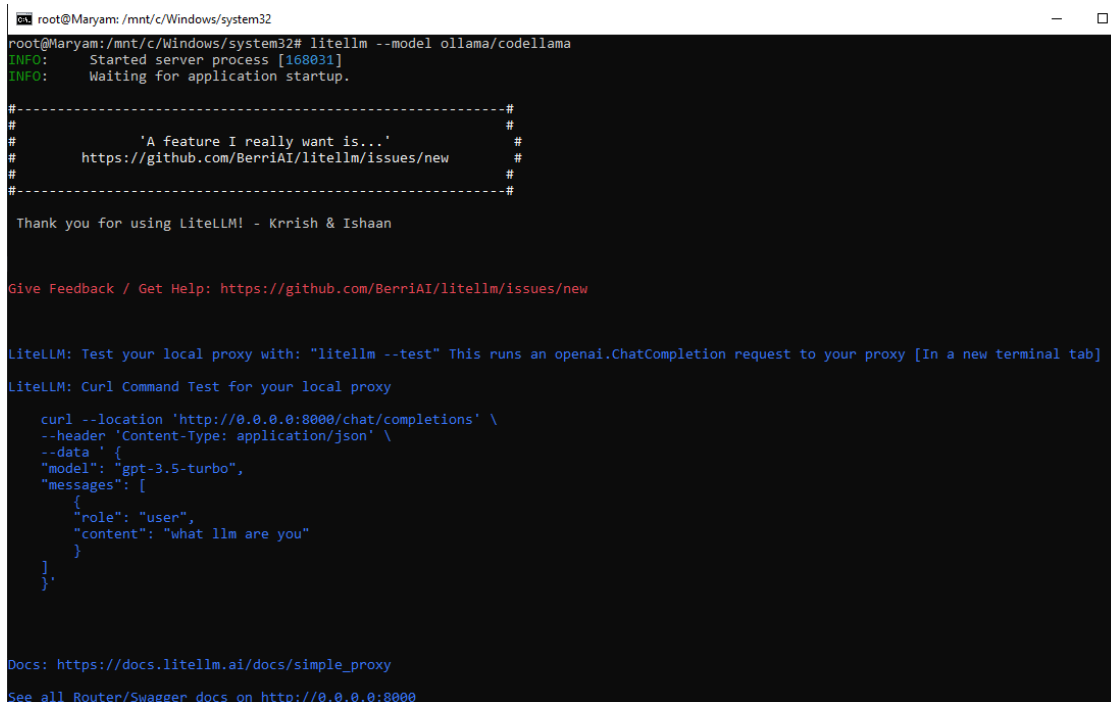
```

Figure 5: Litellm Installation

2.3.3.2.1 Installation Error ^

However, we encountered an error during the setup, which said “name ‘async_generator’ is not defined.” We were able to resolve this issue by installing async_generator using the command “pip install async_generator” “Litellm GitHub Issue 1078 Comment” (n.d.).

Once we had installed Litellm successfully, we loaded the Ollama model effortlessly by typing ‘litellm -m {ollama/model_name}’. This process is similar to loading any downloaded model. After executing the command successfully, Uvicorn started operating at Local Host Port 8000, confirming that the configuration was successful.



```

root@Maryam: /mnt/c/Windows/system32

root@Maryam:/mnt/c/Windows/system32# litellm --model ollama/codellama
INFO: Started server process [168031]
INFO: Waiting for application startup.

#-----#
#                                     #
#   'A feature I really want is...'   #
#   https://github.com/BerriAI/litellm/issues/new #
#                                     #
#-----#

Thank you for using LiteLLM! - Krrish & Ishaan

Give Feedback / Get Help: https://github.com/BerriAI/litellm/issues/new

LiteLLM: Test your local proxy with: "litellm --test" This runs an openai.ChatCompletion request to your proxy [In a new terminal tab]
LiteLLM: Curl Command Test for your local proxy

curl --location 'http://0.0.0.0:8000/chat/completions' \
--header 'Content-Type: application/json' \
--data '{
  "model": "gpt-3.5-turbo",
  "messages": [
    {
      "role": "user",
      "content": "what llm are you"
    }
  ]
}'

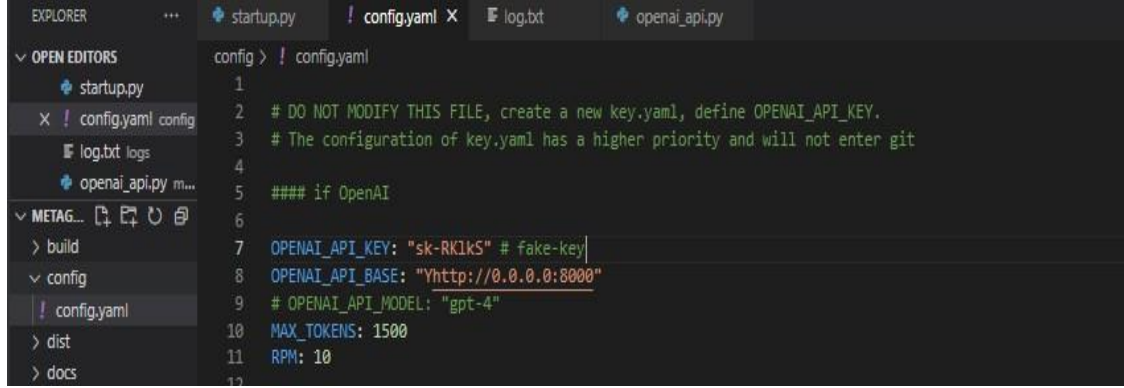
Docs: https://docs.litellm.ai/docs/simple_proxy
See all Router/Swagger docs on http://0.0.0.0:8000

```

Figure 6: Loaded Ollama Model Codellama

2.3.4 Configuring MetaGPT

Setting up LiteLM and CodeLlama to be used by MetaGPT Mittal (2023) for code generation tasks was the aim of this setup. The config.yaml file, which is one of the main components of MetaGPT’s setup, allows you to define which resources to use for particular activities. The configuration item for the OPENAI_API_BASE option in this file was changed to point to the port and URL where LiteLM and Code Llama were hosted. It looked like this:



```

1
2 # DO NOT MODIFY THIS FILE, create a new key.yaml, define OPENAI_API_KEY.
3 # The configuration of key.yaml has a higher priority and will not enter git
4
5 ##### if OpenAI
6
7 OPENAI_API_KEY: "sk-RK1kS" # fake-key
8 OPENAI_API_BASE: "http://0.0.0.0:8000"
9 # OPENAI_API_MODEL: "gpt-4"
10 MAX_TOKENS: 1500
11 RPM: 10
12

```

Figure 7: MetaGPT Configuration

The setup was tailored to specifically target the location where LiteLM and Code Llama resources were available, as seen in the image above. The URL and port were modified by the particular hosting environment.

While writing code, MetaGPT was able to access and utilize the functionalities provided by LiteLM and Code Llama by customizing the configuration. It ensured that Code Llama and LiteLM’s specialized features and models improved MetaGPT’s code creation capabilities and allowed for a wider range of relevant outputs.

2.3.5 Configurig AutoGen

Creating a "config list" was a critical step in Autogen's configuration process that differed significantly from the standard procedure. For this process to work, either a local URL for API access or a GPT3.5/4 API key must be entered. But in this case, we decided to set up a local model URL in addition to other URLs, making sure that Mistral and Codellama have distinct names. The process started with the config list command, which defined the base URL for accessing API endpoints as a JSON object. We set the API key to null and used the terminal to retrieve the URL for the Mistral model, which we then incorporated into the settings. After a similar procedure for Codellama, to maintain port consistency, we created 'llm config' options for Mistral and Codellama and linked them to their respective config lists.

The next stage involved the development of two assistant agents, each related to a certain model. We created an agent with the Mistral model configuration called "assistant" by using the command "autogen do assistant agent." In the same way, the Codellama configuration was used to set up an additional agent called "coder." This configuration made it possible to carry out tasks unique to each model.


```

1 import autogen
2
3 config_list_mistral = [
4     {
5         'base_url': 'http://0.0.0.0:20201',
6         'api_key': 'NULL'
7     }
8 ]
9
10 config_list_codellama = [
11     {
12         'base_url': 'http://0.0.0.0:8000',
13         'api_key': 'NULL'
14     }
15 ]
16
17 llm_config_mistral = {
18     'config_list': config_list_mistral,
19 }
20
21 llm_config_codellama = {
22     'config_list': config_list_codellama,
23 }
24
25 # assistant = autogen.AssistantAgent(
26 #     name="Assistant",
27 #     llm_config=llm_config_mistral,
28 # )
29
30 coder = autogen.AssistantAgent(
31     name="Coder",
32     llm_config=llm_config_codellama,
33 )
34
35 user_proxy = autogen.UserProxyAgent(
36     name="user_proxy",
37     human_input_mode="TERMINATE",
38     max_consecutive_auto_reply=10,
39     is_termination_msg=lambda x: x.get("content", "").rstrip().endswith("TERMINATE"),
40     code_execution_config={"work_dir": "web"},
41     llm_config=llm_config_mistral,
42     system_message="Reply TERMINATE if the task has been solved full satisfactory otherwise, reply CONTINUE, or the reason why the task is not solved yet."
43 )
44
45 task="tell me a joke"
46
47
48

```

Figure 8: AutoGen Configuration

Subsequently, we created a user proxy agent that is crucial for handling tasks between various agents in the group chat configuration. For general tasks, this user proxy agent used the Mistral model. After that, an activity was created that required the agent to "tell a joke" to test the system. The next step involved the creation of a group chat with several agents, and to control these agents, a group chat manager was made Vidhya (2023).

2.3.6 Consistency of the output from both AutoGen & MetaGPT

Executing the task via 'user proxy do initiate chat' generated interactions between the agents, showcasing the output from both the Mistral and Codellama models. Subsequent modifications were made to refine interactions between the agents, allowing for better collaboration and task execution. Ultimately, the setup proved successful, demonstrating seamless interaction and task execution between multiple models and agents.

```

/home/maryam_linux/miniconda3/envs/pyautogen/bin/python /mnt/c/Users/Hp/autogen_ws1/autogen_yt1.py
(pyautogen) (base) maryam_linux@taryam:/mnt/c/Users/Hp/autogen_ws1$ /home/maryam_linux/miniconda3/envs/pyautogen/bin/python /mnt/c/Users/Hp/autogen_ws1/autogen_yt1.py
user_proxy (to Coder):
tell me a joke

-----
Coder (to user_proxy):

Here's a joke for you:

Why couldn't the math book keep its head above water?

Because it was always struggling with too many equations!

I hope that brought a smile to your face!

-----
>>>>>> USING AUTO REPLY...

```

Figure 9: Autogen Task Execution


```

-----#
Thank you for using LiteLLM! - Krrish & Ishaan

Give Feedback / Get Help: https://github.com/BerriAI/litellm/issues/new

LiteLLM: Test your local proxy with: "litellm --test" This runs an openai.ChatCompletion request to your proxy [In a new terminal tab]
LiteLLM: Curl Command Test for your local proxy

curl --location 'http://0.0.0.0:8000/chat/completions' \
--header 'Content-Type: application/json' \
--data '{
  "model": "gpt-3.5-turbo",
  "messages": [
    {
      "role": "user",
      "content": "what llm are you"
    }
  ]
}'

Docs: https://docs.litellm.ai/docs/simple_proxy

INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8225 (Press CTRL+C to quit)
Final response: ModelResponse(id='chatcmpl-bd90e92b-bec8-40c8-ad87-7f4878369c2a', choices=[Choices(finish_reason='stop', index=0, message=Message(content='\nHere's a joke for you:\n\nWhy couldn't the math book keep its head above water?\n\nBecause it was always struggling with too many equations!\n\nI hope that brought a smile to your face!', role='assistant'))], created=1702905718, model='ollama/codellama', object='chat.completion', system_fingerprint=None, usage=Usage(prompt_tokens=465, completion_tokens=40, total_tokens=505))
INFO: 127.0.0.1:49878 - "POST /chat/completions HTTP/1.1" 200 OK

```

Figure 10: Prompt Response

but when assigning the agent the task of “writing a CLI snake game based on Pygame” it took much time and then gave a time-out error because of the complexity of the task.

For MetaGPT, after configuration on wsl window 11 that has 16 GB RAM, it takes almost 10 minutes and gives a “timeout error” and also the Ollama server showed this warning: “WARNING: failed to allocate 156.00 MiB of pinned memory: out of memory”

```

62 ## Visit https://serpaol.com/ to get key.
PROBLEMS  PORTS  OUTPUT  DEBUG CONSOLE  TERMINAL
bash - metagpt

(metagpt) mryam_linux@mryam:/mnt/c/Users/Hp/autogen_ws1/MetaGPT/metagpt$ metagpt "write a cli snake game based on pygame"
2023-12-24 03:01:22.265 | INFO | metagpt.const:get_metagpt_package_root:32 - Package root set to /mnt/c/Users/Hp/autogen_ws1/MetaGPT/metagpt
2023-12-24 03:01:27.086 | INFO | metagpt.team:invest:91 - Investment: $3.0.
2023-12-24 03:01:27.892 | INFO | metagpt.roles.role:act:392 - Alice(Product Manager): to do PrepareDocuments()
2023-12-24 03:01:28.303 | INFO | metagpt.utils.file_repository:save:60 - save to: /mnt/c/Users/Hp/autogen_ws1/MetaGPT/metagpt/workspace/20231224030127/docs/requirement.txt
2023-12-24 03:01:28.339 | INFO | metagpt.roles.role:act:392 - Alice(Product Manager): to do WritePRD()
2023-12-24 03:06:28.746 | ERROR | metagpt.utils.common:log_it:433 - Finished call to 'metagpt.actions.action_node.ActionNode._ask_v1' after 300.356(s), this was the 1st time calling it. e
xp: Request timed out.
2023-12-24 03:11:30.746 | ERROR | metagpt.utils.common:log_it:433 - Finished call to 'metagpt.actions.action_node.ActionNode._ask_v1' after 602.357(s), this was the 2nd time calling it. e
xp: Request timed out.
2023-12-24 03:16:31.745 | ERROR | metagpt.utils.common:log_it:433 - Finished call to 'metagpt.actions.action_node.ActionNode._ask_v1' after 903.356(s), this was the 3rd time calling it. e
xp: Request timed out.

```

Figure 11: MetaGPT Timeout Error

This error is likely because there is not enough memory available on my Windows laptop to run the all files. This may be because the server is taking too long to allocate the memory, and the timeout is being reached before the allocation can be completed. we have checked the system resources and confirmed that there is not enough physical memory available.

we have tried this on a MacBook that has 8 GB RAM, but the output was not consistent and it stopped itself after some time.

```

~/MetaGPT -- -zsh      ~/Downloads/MetaGPT-0.1.0 -- -zsh      ~/Downloads -- -zsh      ~/Downloads/MetaGPT-0.1.0 -- -zsh      .a - Stelin --model ollama/code llama
(metaGPTv1) allanadthomas@Allans-MBP MetaGPT-0.1.0 % write python startup.py "write a cli snake game"
usage: write user (tty)
(metaGPTv1) allanadthomas@Allans-MBP MetaGPT-0.1.0 % python startup.py "write a cli snake game"
2024-01-01 19:46:20.282 | INFO | metagpt.config._init_:43 - Config loading done.
2024-01-01 19:46:29.766 | INFO | metagpt.software_company.invest:39 - Investment: $3.0.
2024-01-01 19:46:29.767 | INFO | metagpt.roles.role:act:155 - Alice(Product Manager): ready to WritePRD
2024-01-01 19:46:29.767 | WARNING | metagpt.actions.search_and_summarize:run:114 - Configure SERPAPI_API_KEY to unlock full feature
## Original Requirements:
Write a cli snake game.

## Product Goals:
1. Create a fun and engaging gameplay experience for the user.
2. Provide a unique and challenging game mechanic that sets the product apart from other games in the market.
3. Offer a high-quality user interface that is easy to navigate and understand.

## User Stories:
1. As a user, I want to be able to play the game on any platform so that I can access it wherever I go.
2. As a user, I want the game to have a simple and intuitive control scheme so that I can easily navigate the gameplay.
3. As a user, I want to be able to customize my snake's appearance and size so that I can personalize my experience.
4. As a user, I want the game to provide clear and concise feedback on my performance so that I can improve my skills.
5. As a user, I want to be able to play the game in different difficulty levels so that I can challenge myself and play at a level that suits my skill level.

## Competitive Analysis:
1. Python Snake Game: This game provides a simple and easy-to-use control scheme, with clear and concise feedback on the user's performance. It also offers customization options for the snake's appearance and size. However, it may not be as challenging or engaging as other games in the market.
2. Snake Game: This game provides a more complex and challenging game mechanic, with multiple difficulty levels to choose from. It also offers a high-quality user interface with clear instructions and feedback on the user's performance. However, it may not be as fun or engaging as other games in the market.
3. Snake Game Clone: This game provides a simple and easy-to-use control scheme, with clear and concise feedback on the user's performance. It also offers customization options for the snake's appearance and size. However, it may not offer as many features or challenges as other games in the market.

## Competitive Quadrant Chart:
| User | Gameplay | Features | Challenges | Reach | Engagement | Customization | |
|---|---|---|---|---|---|---|---|
| Python Snake Game | | | | | | |
| Snake Game | | | | | | |
| Snake Game Clone | | | | | | |
| Our Target Product | X | X | X | X | X | X | X |

## Requirement Analysis:
The product should be a fun and engaging gameplay experience for the user. It should provide clear and concise feedback on the user's performance, with multiple difficulty levels to choose from. It should also offer customization options for the snake's appearance and size. The product should be easy to navigate and understand, with a simple and intuitive control scheme.

## Requirement Pool:
1. End game screen (P0) - A end game screen that displays the user's score and allows them to restart the game.
2. Customization options (P1) - Customization options for the snake's appearance and size, including different colors and sizes.
3. Multi-difficulty levels (P2) - Multiple difficulty levels to choose from, with increasing challenge as the user progresses.
4. High-quality UI (P2) - A high-quality user interface that is easy to navigate and understand, with clear instructions and feedback on the user's performance.
5. Easy control scheme (P2) - An easy-to-use control scheme that allows the user to easily navigate the gameplay.
## Original Requirements:
python
{
    "Create a CLI Snake Game"
}
...

## Product Goals:
python
{
    "Make the game more engaging",
    "Improve the user experience",
    "Increase player satisfaction"
}

```

Figure 12: Task Execution on Mac

2.3.7 Debugging features

When examining MetaGPT's debugging capabilities within the simulated software company environment, the logs offer a thorough record of all the events and role-specific actions. Interestingly, an important alert appears, requesting that SERPAPI_API_KEY be configured to enable full feature functionality. It prevents MetaGPT's capabilities from operating to their full capacity, increasing the likelihood of timeouts and delaying task completion. By setting the API key and resolving this warning, MetaGPT will run as efficiently as possible, decreasing the chance of timeouts and improving the framework's general dependability and efficiency in producing desired outputs, like creating a CLISnake game.

```

EXPLORER  log.txt  X
OPEN EDITORS  MetaGPT-0.1.0 > logs > log.txt
X log.txt MetaGPT-0.1.0/logs
META-GPT-V1
  MetaGPT-0.1.0
    build
    config
    dist
    docs
    examples
    logs
  log.txt
  metagpt
    metagpt.egg-info
    tests
    gitignore
    Dockerfile
    LICENSE
    README.md
    requirements.txt
    setup.py
    startup.py
1 2024-01-04 01:10:40.001 | INFO | metagpt.config._init_:43 - Config loading done.
2 2024-01-04 01:10:41.443 | INFO | metagpt.software_company.invest:39 - Investment: $3.0.
3 2024-01-04 01:10:41.443 | DEBUG | metagpt.software_company.run:58 - n_round=4
4 2024-01-04 01:10:41.444 | DEBUG | metagpt.roles.role:observe:186 - Alice(Product Manager) observed: ['BOSS: write a cli snake ga...']
5 2024-01-04 01:10:41.445 | DEBUG | metagpt.roles.role:react:199 - Alice(Product Manager): self.rc.state=0, will do WritePRD
6 2024-01-04 01:10:41.445 | INFO | metagpt.roles.role:act:155 - Alice(Product Manager): ready to WritePRD
7 2024-01-04 01:10:41.445 | WARNING | metagpt.actions.search_and_summarize:run:114 - Configure SERPAPI_API_KEY to unlock full feature
8 2024-01-04 01:10:41.446 | DEBUG | metagpt.roles.role:run:226 - Bob(Architect): no news. waiting.
9 2024-01-04 01:10:41.447 | DEBUG | metagpt.roles.role:run:226 - Eve(Project Manager): no news. waiting.
10 2024-01-04 01:10:41.447 | DEBUG | metagpt.roles.role:run:226 - Alex(Engineer): no news. waiting.
11 2024-01-04 01:13:05.383 | INFO | metagpt.config._init_:43 - Config loading done.
12 2024-01-04 01:13:06.058 | INFO | metagpt.software_company.invest:39 - Investment: $3.0.
13 2024-01-04 01:13:06.059 | DEBUG | metagpt.software_company.run:58 - n_round=4
14 2024-01-04 01:13:06.059 | DEBUG | metagpt.roles.role:observe:186 - Alice(Product Manager) observed: ['BOSS: write a cli snake ga...']
15 2024-01-04 01:13:06.060 | DEBUG | metagpt.roles.role:react:199 - Alice(Product Manager): self.rc.state=0, will do WritePRD
16 2024-01-04 01:13:06.060 | INFO | metagpt.roles.role:act:155 - Alice(Product Manager): ready to WritePRD
17 2024-01-04 01:13:06.061 | WARNING | metagpt.actions.search_and_summarize:run:114 - Configure SERPAPI_API_KEY to unlock full feature
18 2024-01-04 01:13:06.061 | DEBUG | metagpt.roles.role:run:226 - Bob(Architect): no news. waiting.
19 2024-01-04 01:13:06.061 | DEBUG | metagpt.roles.role:run:226 - Eve(Project Manager): no news. waiting.
20 2024-01-04 01:13:06.062 | DEBUG | metagpt.roles.role:run:226 - Alex(Engineer): no news. waiting.
21 2024-01-04 01:18:32.573 | INFO | metagpt.config._init_:43 - Config loading done.
22 2024-01-04 01:20:21.936 | INFO | metagpt.config._init_:43 - Config loading done.
23 2024-01-04 01:22:52.768 | INFO | metagpt.config._init_:43 - Config loading done.
24 2024-01-04 01:24:45.952 | INFO | metagpt.config._init_:43 - Config loading done.
25 2024-01-09 14:44:58.788 | INFO | metagpt.config._init_:43 - Config loading done.
26 2024-01-09 14:47:13.508 | INFO | metagpt.config._init_:43 - Config loading done.
27

```

Figure 13: MetaGPT Debugging Features

For autogen encountered the same errors, identifying possible challenges with complex assignments. Debugging features in both frameworks require more investigation to discover the underlying causes of these issues and improve platform robustness. Investigating error logs, profiling execution times, and optimizing resource use are all necessary steps toward increasing debugging capabilities.

2.3.8 Overview of Open Source LLMs

The introduction of open-source language models (LLMs) such as CodeLLAMA and Mistral into MetaGPT and AutoGen has marked a significant advancement in the development of these frameworks. These LLMs allow MetaGPT and AutoGen to write more detailed and informative text, conduct code generation and translation tasks, and answer queries in an instructive manner. However, integrating these LLMs has caused challenges, specifically in obtaining accurate and consistent outcomes.

The differences in the training data utilized for each LLM pose a challenge. CodeLLAMA, for example, is trained on a large dataset of code repositories, whereas Mistral is learned on text and code. The discrepancy in training data can cause variations in the quality and accuracy of the LLM outputs.

Another problem is the variety of use cases for LLMs. MetaGPT and AutoGen are intended to perform a wide range of tasks, including creating various creative text formats and answering queries in an instructive manner. However, not all LLMs are equally capable of completing these various responsibilities. For example, CodeLLAMA may thrive in code generation but struggle with creative text formats. Each dataset’s unique properties can influence the model’s performance and output, resulting in unpredictable outcomes.

2.3.9 Docker compatibility

Docker compatibility is critical for the smooth deployment of AI models. While the research briefly covers the use of Ollama and Litellm within the frameworks, further investigation of Docker compatibility is required. It would be easier to use and more accessible for users to deploy MetaGPT and AutoGen inside Docker containers if explicit instructions and guidelines were provided, along with information on probable problems and solutions.

2.4 Overview and comparison of existing open-source GPT models

Table 1 compares different aspects of CodeLLAMA and Mistral.

Table 1: Comparison

Criterion	CodeLLAMA	Mistral
Requirements	Modest; Powerful processors, large RAM, GPU for training	Moderate; Accessible for various system capacities (up to 32 GB RAM for large local models)
Installation	of Python, virtual environment setup	Python, virtual environment setup, pip installation
Team	Developed by [Provide details about the development team]	Developed collaboratively by researchers from [Provide details about the development team]
OpenAI Compatibility	Compatible with OpenAI standards for input/output formats via literally	Seamless integration with OpenAI using local URLs or GPT3.5/4 API key
Configuration	Modifying config.yaml for LiteLM and Code Llama	Creating a “config list,” setting up local model URLs, configuring Mistral and Codellama agents
Consistency	Demonstrated consistency in output with successful interactions	Challenges with timeouts on complex tasks, needs debugging investigation
Debugging	Logs provide a thorough record, highlights SERPAPI_API_KEY configuration	Similar challenges with timeouts, debugging features require further investigation
Time & Memory	Timeout errors observed on complex tasks	Similar timeout issues, and possible resource limitations (e.g., “failed to allocate 156.00 MiB of pinned memory: out of memory”)

Criterion	CodeLLAMA	Mistral
LLMs Overview	Integrates open-source LLMs (CodeLLAMA, Mistral) for code generation	Open-source LLMs (CodeLLAMA, Mistral) contribute capabilities, and challenges with training data and use cases
Docker Compatibility	Requires further investigation	Docker compatibility is essential for deployment,further investigation recommended

3 AOS Model configuration in MetaGPT Framework

To democratize access to cutting-edge AI capabilities, Azure OpenAI Service (AOS) has built a comprehensive infrastructure for deploying OpenAI-developed models. Our platform provides an easy-to-use API and SDK, allowing developers and businesses to seamlessly integrate. GPT-3.5/4 is among the most advanced models available, with an astonishing 175 billion parameters. This model's development process included a combination of supervised learning on large datasets and reinforcement learning, using input from both human experts and AI systems. The platform provides aims to open up new areas of study and ideas that were previously considered unavailable. Notably, our AI services are supported by Azure's powerful infrastructure, which ensures scalability and stability.

Access to the Azure OpenAI Service is provided via a predefined base URL, with authentication and authorization handled by API keys. Under the deployment name "got-4," we provide access to the GPT-4 model, which represents the peak of AI intelligence and performance.

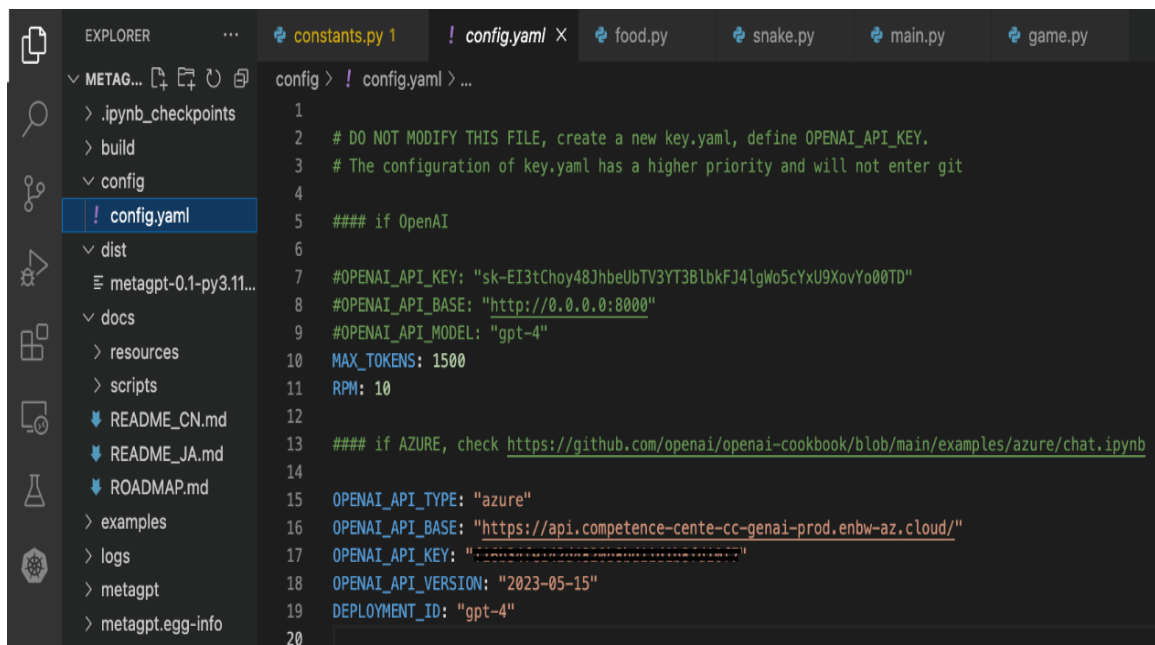


Figure 14: AOS API-Key Configuration in MetaGPT

3.1 Uses Cases

3.1.1 Existing Use Cases

3.1.1.1 Snake Game After using the metagpt and autogen for generating a snake game with mistral as well as codellama models we did not get consistent outcomes. but at least for AOS API-Key in metagpt, it showcased its ability to generate code.

When running "python startup.py," the prompt instructs the Pygame library to be used to create a Snake game. The "snake-game-pygame" folder is created in the workspace as a result of this step. The necessary game elements are arranged into distinct files inside this folder to facilitate modular construction. Among these are the files "food.py," "Snake.py," "Game.py," "Constant.py," and "Main.py." Every file has a distinct function in putting various game elements into practice. "Snake.py" is probably in charge of the behavior and movement of the snake character, whilst "Food.py" probably manages the creation and handling of food items in the game environment. The main mechanics and logic of the game might be found in "Game.py". To ensure consistency and ease of modification, "Constant.py" probably saves constant values and parameters used throughout the game. You can play the game by active this Media2.mov object:



```

~/MetaGPT -- -zsh          ~/Downloads/MetaGPT-0.1.0 -- -zsh          ~/Downloads/MetaGPT-0.1.0 -- -zsh          ~/Downloads/MetaGPT-0.1.0 -- -zsh          ~/Downloads/MetaGPT-0.1.0 -- -zsh          +
[metagptv1] allan@dothemoq:~$ python startup.py "create a complete 2048 game based on pygame"
2024-02-12 17:03:55.099 | INFO | metagpt.config: init. log = Config loading done.
2024-02-12 17:03:55.639 | INFO | metagpt.software_company:invest:100 - Investment: $3.0.
2024-02-12 17:03:55.639 | INFO | metagpt.roles.role:act:100 - AIsse(Product Manager): ready to WritePRD
2024-02-12 17:03:55.639 | WARNING | metagpt.actions.search_and_summarize:run:114 - Configure SERPAPI_API_KEY to unlock full feature
## Original Requirements
The boss has tasked us with creating a complete 2048 game based on pygame.

## Product Goals
python
[
  "Create a user-friendly 2048 game using pygame",
  "Ensure the game is stable and bug-free",
  "Implement a scoring system and leaderboard"
]

## User Stories
python
[
  "As a user, I want to be able to easily understand how to play the game",
  "As a user, I want the game to run smoothly without any glitches",
  "As a user, I want to be able to see my score while playing the game",
  "As a user, I want to have the option to restart the game at any point",
  "As a user, I want to see my rank on the leaderboard after each game"
]

## Competitive Analysis
python
[
  "2048 Game by Gabriele Cirulli: The original 2048 game, simple and intuitive interface, lacks a leaderboard",
  "2048 Number puzzle game by Estoty Entertainment LLC: Offers different board sizes, includes a leaderboard, contains ads",
  "2048 by Androbaby: Offers undo move option, includes a leaderboard, contains ads",
  "2048 by R. App: Offers different board sizes, lacks a leaderboard, ad-free",
  "2048 by Banana & Co.: Offers undo move option, lacks a leaderboard, contains ads",
  "2048 by Solobon LLC: Simple and intuitive interface, includes a leaderboard, ad-free",
  "2048 by Ketchapp: Offers different board sizes and game modes, includes a leaderboard, contains ads"
]

## Competitive Quadrant Chart
python
Reinhold
QuadrantChart
title Reach and engagement of campaigns
x-axis Low Reach --> High Reach
y-axis Low Engagement --> High Engagement
quadrant-1 We should expand
quadrant-2 Need to promote
quadrant-3 Re-evaluate
quadrant-4 May be improved
"2048 Game by Gabriele Cirulli": [0.8, 0.7]
"2048 by Androbaby": [0.6, 0.8]
"2048 by R. App": [0.5, 0.4]
"2048 by Banana & Co.": [0.4, 0.3]
"2048 by Solobon LLC": [0.3, 0.2]
"2048 by Ketchapp": [0.2, 0.1]
"our Target Product": [0.8, 0.6]

```

Figure 15: snake-game-pygame

3.1.1.2 2048 Game Prompt asked: python startup.py: "Create a complete 2048 game based on pygame" then get a folder pygame-2048 in workspace: Game.py, interface.py, Main.py, Leaderboard.py and Score.py. play the game by activating the Media1.mov object ;




```

~/MetaGPT -- -zsh      ~/Downloads/MetaGPT-0.1.0 -- -zsh      ~/Downloads/MetaGPT-0.1.0 -- -zsh      ~/Downloads/MetaGPT-0.1.0 -- -zsh      ~/Downloads/MetaGPT-0.1.0 -- -zsh      +
(metaGPTv1) allanaldothomas@a292972-ad88-4f5e-afd7-cbe26a14092 MetaGPT-0.1.0 % python startup.py "create a complete 2048 game based on pygame"
2024-02-02 17:53:53.895 | INFO | metagpt.config._init_:143 - Config loading done.
2024-02-02 17:53:53.639 | INFO | metagpt.software_company.invest:39 - Investment: $3.0.
2024-02-02 17:53:53.639 | INFO | metagpt.roles.role1.act:189 - Alice(Product Manager): ready to WritePRD
2024-02-02 17:53:53.639 | WARNING | metagpt.actions.search_and_summarize:run:114 - Configure SERPAPI_API_KEY to unlock full feature
## Original Requirements
The boss has tasked us with creating a complete 2048 game based on pygame.

## Product Goals
python
[
    "Create a user-friendly 2048 game using pygame.",
    "Ensure the game is stable and bug-free",
    "Implement a scoring system and leaderboard"
]
...

## User Stories
python
[
    "As a user, I want to be able to easily understand how to play the game",
    "As a user, I want the game to run smoothly without any glitches",
    "As a user, I want to be able to see my score while playing the game",
    "As a user, I want to have the option to restart the game at any point",
    "As a user, I want to see my rank on the leaderboard after each game"
]
...

## Competitive Analysis
python
[
    "2048 Game by Gabriele Cirulli: The original 2048 game, simple and intuitive interface, lacks a leaderboard",
    "2048 Number puzzle game by Estoty Entertainment LLC: Offers different board sizes, includes a leaderboard, contains ads",
    "2048 by Androbaby: Offers undo move option, includes a leaderboard, contains ads",
    "2048 by R. App: Offers different board sizes, lacks a leaderboard, ad-free",
    "2048 by Banana & Co.: Offers undo move option, lacks a leaderboard, contains ads",
    "2048 by Solobon LLC: Simple and intuitive interface, includes a leaderboard, ad-free",
    "2048 by Ketchapp: Offers different board sizes and game modes, includes a leaderboard, contains ads"
]
...

## Competitive Quadrant Chart
mermaid
graph TD
    title Reach and engagement of campaigns
    x-axis Low Reach --> High Reach
    y-axis Low Engagement --> High Engagement
    quadrant-1 We should expand
    quadrant-2 Need to promote
    quadrant-3 Re-evaluate
    quadrant-4 May be improved
    "2048 Game by Gabriele Cirulli": [0.8, 0.7]
    "2048 Number puzzle game by Estoty Entertainment LLC": [0.7, 0.6]
    "2048 by R. App": [0.5, 0.5]
    "2048 by Banana & Co.": [0.4, 0.3]
    "2048 by Solobon LLC": [0.3, 0.2]
    "2048 by Ketchapp": [0.2, 0.1]
    "Our Target Product": [0.5, 0.6]
    ...

```

Figure 16: 2048-game-pygame

3.1.1.3 Brick Breaker Game Prompt asked: python startup.py: "Write a brick breaker based on pygame" Get a brick-breaker folder in a workspace: ball.py, brick.py, database.py, game.py, and main.py. play the game by activating the Media3.mov object ;

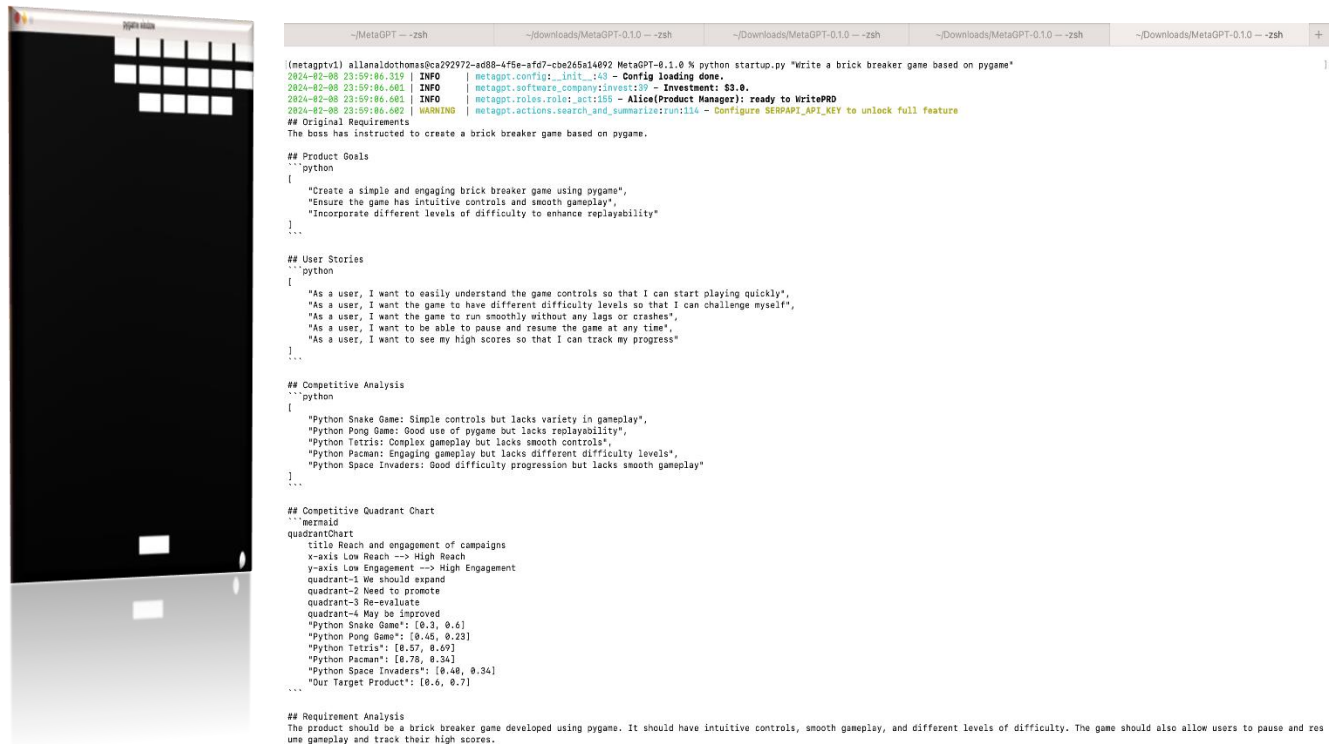


Figure 17: brick-breaker-game-pygame

4 Results

In terms of performance, MetaGPT demonstrated competence in using LiteLM and CodeLLAMA to generate code for a Snake Game. The response generated for the command "write a CLI snake game based on pygame" did not provide the expected Python code for the game. Instead, it consisted of information related to product goals, user stories, competitive analysis, and a requirement pool. The Output included information about investment, roles, and actions that did not align with the task of writing a CLI snake game in Python. We encountered an issue where a certain task continued running for a long time. At some point, it started repeating information that had already been provided including details about product goals, user stories, competitive analysis, and a requirement pool. The task did not terminate until it was forcibly stopped.

However, it only generated text and did not provide any relevant code or information on how the game could be implemented. On the other hand, delays in execution on a Windows 11 system with 16 GB RAM resulted in a timeout issue and brought to light certain difficulties when managing resource-intensive operations. The Ollama server's warning re-regarding memory allocation problems highlighted the necessity for optimization even further. MetaGPT's logs offered a thorough record of debugging features, including notifications about setting SERPAPI_API_KEY, which is essential for reducing timeouts and optimizing its functionality. However, AutoGen, which was set up with Mistral and CodeL LAMA, encountered similar difficulties when creating a CLI Snake Game, including delays and a timeout issue. Both need to look into the root causes of these issues, maximize resource consumption, and strengthen platform resilience.

After getting delays and errors we tried this with Azure API key to see the thorough performance of the GPT-4 model in the MetaGPT framework and of course, using a large model gives us consistent and correct code for existing use cases.

5 Discussion

5.1 Common Challenges:

During the construction of a Snake Game, both MetaGPT and AutoGen encountered similar issues, particularly when dealing with sophisticated programming assignments. The issue of slow code generation, common in agent-based GPT models, emerged as both frameworks grappled with the extensive analysis of information and the need to make multiple decisions during the code creation process, which creates timeout problems in both frameworks and suggests potential constraints when dealing with resource-intensive operations. Notably, memory allocation concerns, as shown by warnings from MetaGPT's Ollama server and AutoGen challenges, underline the crucial importance of optimizing memory consumption. Addressing these difficulties is critical to preventing delays, and timeouts, and improving the frameworks' overall performance while addressing complex programming tasks.

5.2 Possibilities for Enhancement:

A thorough investigation of the debugging features in MetaGPT and AutoGen is necessary to identify and resolve underlying issues. This involves a thorough analysis of error logs, careful execution time profiling, and optimization of resource usage. Moreover, it is critical to optimize the frameworks to handle complex programming tasks more effectively. By addressing issues related to prompt complexity, execution time, and memory allocation, overall performance will be improved, guaranteeing a more efficient and pleasant user experience.

6 Conclusion

In conclusion, while MetaGPT and AutoGPT exhibit promise in code generation, including the creation of a Snake Game, they face significant challenges in managing complex tasks. To maximize the efficacy of these frameworks in practical applications, it is imperative to address issues related to bias, limited availability, and performance optimization. Resolving these challenges will contribute to the responsible and effective integration of agent-based GPT models in software development.

References

- Chen, Weize, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Cheng Qian, Chi-Min Chan, et al. 2023. “AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors in Agents.” *arXiv.org*. <https://doi.org/10.48550/ARXIV.2308.10848>.
- Hong, Sirui, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, et al. 2023. “MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework.” *arXiv e-Prints*, August, arXiv:2308.00352. <https://doi.org/10.48550/arXiv.2308.00352>.
- “Litellm Documentation.” n.d. <https://litellm.vercel.app/docs/>.
- “Litellm GitHub Issue 1078 Comment.” n.d. <https://github.com/BerriAI/litellm/issues/1078#issuecomment-1859044214>.
- Liu, Xiao, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, et al. 2023. “AgentBench: Evaluating LLMs as Agents.” *arXiv.org*. <https://doi.org/10.48550/ARXIV.2308.03688>.
- “Mistral - OLLAMA.AI Library.” n.d.b. <https://ollama.ai/library/mistral>.
- . n.d.a. <https://ollama.ai/library/codellama>.
- Mittal, Aayush. 2023. “MetaGPT: Complete Guide to the Best AI Agent Available Right Now.” 2023. <https://www.unite.ai/metagpt-complete-guide-to-the-best-ai-agent-available-right-now/>.
- “OLLAMA.AI.” n.d. <https://ollama.ai/>.
- Parafina, Sophia. 2023. “Run Large Language Models with OLLAMA and Lightsail for Research.” 2023. <https://community.aws/posts/run-large-language-models-with-ollama-and-lightsail-for-research>.
- Rozière, Baptiste, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Tan, Yossi Adi, et al. 2023. “Code Llama: Open Foundation Models for Code.” *arXiv.org*. <https://doi.org/10.48550/ARXIV.2308.12950>.
- Touvron, Hugo, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, et al. 2023. “Llama 2: Open Foundation and Fine-Tuned Chat Models.” *arXiv.org*. <https://doi.org/10.48550/ARXIV.2307.09288>.
- Vidhya, Analytics. 2023. “Launching into AutoGen: Exploring the Basics of a Multi-Agent Framework.” <https://www.analyticsvidhya.com/blog/2023/11/launching-into-autogen-exploring-the-basics-of-a-multi-agent-framework/>.
- Waldinger, R. J., and R. C. T. Lee. 1969. “PROW: A Step Toward Automatic Program Writing.” In *Proceedings of the 1st International Joint Conference on Artificial Intelligence (IJCAI)*, edited by D. E. Walker and L. M. Norton, 241–52. Morgan Kaufmann.
- Wu, Qingyun, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, et al. 2023. “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation.” *arXiv e-Prints*, August, arXiv:2308.08155. <https://doi.org/10.48550/arXiv.2308.08155>.