VentureForce - Idea to Action Plan using Multi-Agent Cooperation

Ahad Imran and **Muhammad Mubashir** and **Asad Ullah Warraich** and **Hamza Akmal**
Northeastern University
{imran.ah, mubashir.m, waraich.a, chaudary.h}@northeastern.edu

1 Project Overview/Motivation

Large Language Model (LLM) dialogue agents have unveiled limitations in specific domains due to their generalized training data with typical problems i.e poor contextual parsing, lack of domain knowledge, factual inaccuracies, ethical dilemmas, bias propagation, and hallucinations. These issues are critical for entrepreneurs/founders looking to convert their initial ideas into a set of tangible deliverables. While public LLM's are available for this use case, the responses are often unable to accurately compile and form a direct-action plan. Our aim is to provide a collection of specialized agents working together to achieve this exact purpose.

The proposed architecture aggregates fine-tuned LLM agents together in an entrepreneurial hierarchy and composes a panel of dialogue agents that specialize in technical, marketing, and legal branches of the business. The architecture implements a multi-agent system where each agent uses specialized expertise to analyze and evaluate a proposed product idea. The agents then work in conjunction under the Project Manager (PM) that further refines and compiles the responses of the agents to create a detailed business plan. The approach utilizes four key agents, an Engineering Agent (powered by Llama 3.2 Model) for technical solutions, the Marketing Agent (powered by Gemma) to generate a marketing plan, the Legal Agent (powered by Gemma) for risk and compliance evaluations, and the Project Manager Agent (powered by GPT-4). The agents communicate using a FastAPI based framework to automate the workflow, using the PM Agent for flexibility and scalability. The architecture produces domain-specific results as well as maintains a coherent overall methodology to generate a feasible plan.

2 Related Work

Recent works on multi-agent LLM systems address how the LLM agents can work together on complex tasks to solve problems. Du et al. (2023) introduces the idea of using LLMs to engage in multi-agent debate to improve factuality. The approach proposed in our architecture uses a similar technique where each agent generates the content, and another model (the PM) processes the content to make it more coherent. Additionally, this 'second pass' also reduces factual inaccuracies or reasoning errors that the initial models may have generated. Similarly, the idea of "Society of Minds' Minsky (1988) is another inspiration for our approach, as multiple "agents" are collaborating in specialized roles in order to generate results.

Gandhi et al. (2024) discusses multi-agent architectures for Machine Learning Tasks. The paper gives an overview of how unique role-based approaches specify responsibilities and reduce system complexity by using "planner" and "worker" agents. This model operates by regulating general activity by consolidating outputs, assigning tasks to "expert" agents, and staying aligned with the overall goals of the problem. Running parallel to that concept is our idea of the 'Project Manager' and Task-Specific Agents, wherein the PM filters and dispatches tasks and mediates between the task-specific agents.

Other contributions like Krave (2010) put forward practical examples related to multi-agent LLMs, showing how specialized agents (e.g., marketing, logistics) could work in alignment in producing end-to-end solutions. Similarly, each specialized agent will focus on specific domain expertise and concurrently report on its progress and resource use within one common workspace, parallel to our PM Agent.

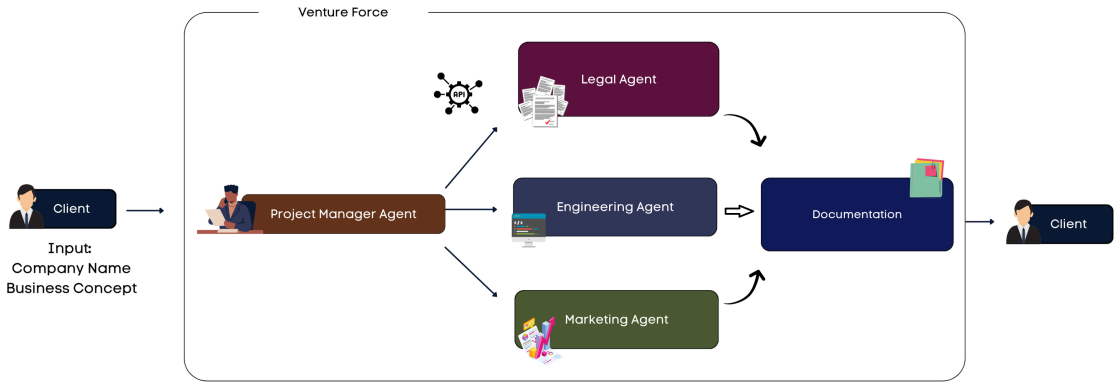

Figure 1: VentureForce Architecture

3 Methods

3.1 Agent-wise Methodologies

The engineering agent is using the Llama 3.2 (1B Instruct). It is responsible to conduct an in-depth technical assessment on the product idea. Our initial implementation of the engineering agent is running locally on an Intel i5 12th generation CPU. The engineering agent is also utilising an SQLite database for structured response storage which includes information such as the category of technical evaluation of the idea i.e. Technical Architecture then sub-category i.e. Backend & Frontend Technologies and then the analysis on that specific category.

The Marketing agent is using the Gemma (2B version) and is responsible to evaluate project market feasibility, user demand and explore the competitive landscape. It is hosted on the Google Colab during the initial testing.

The Legal agent also running on the Gemma (2B version) is assessing the legal risks, compliance and intellectual property (IP) of the proposed idea. During the initial experimentation this agent is also hosted on the Google Colabs.

The project manager agent running on GPT-4 is acting as a central coordinator for task distribution and final response aggregation. Its workflow is based on breaking down the initial project idea into tasks for the different agents and then dispatching them. Once the responses are returned it aggregates them and creates a structured report on the proposed idea.

3.2 System Diagram & Explanation

The user input is passed to the Project Manager Agent which deconstructs the concept into separate tasks for each agent. As each agent has an endpoint exposed for requests, these tasks are then sent over to the respective agent's API and continuously waits for a response. The agents on the other hand process the request and invoke the Large Language Model with instructions tailored to their own domain. Once the output is generated, the agent dumps it into a pre-decided schema for all agents and sends it back to the Project Manager. The Project Manager agent then edits each of these responses into a format suitable for a document. Using the docx library, we construct a word document and finally convert it to a pdf for ease. The diagram for the system is shown in Figure 1.

3.3 Database Architecture

The agents use an SQLite database to manage and structure their responses. The schema of the database is hierarchical in nature, meaning responses are categorized into broader technical aspects, with these then having sub-categories that are addressing specific questions. This approach of structured data retrieval from the agents has enabled constrained, accurate and to the point analysis. In the future, this also allows capability to store responses over different proposed tasks from the Project Manager.

3.4 Model Selection and Testing

For model selection, we experimented with various language models which include Llama, Gemma and DeepSeek Reasoning. While DeepSeek Reasoning was demonstrating strong

general reasoning capabilities, the responses it produced lacked the specificity for our domain specific tasks. Llama was chosen for the engineering agent because it provided structured and precise technical analysis, and gemma was selected for marketing and legal tasks because it seemed to adapt well to these domains. We evaluated these models based on the coherence and domain relevance of their responses. For the Project Manager, we found GPT-4 to return an accurate json format with tasks listed for each agent. We chose low parameter models as we plan to finetune them in the future on domain specific knowledge.

3.5 Agent Communication Framework

The communication framework between the different agents is built on FastAPI which is facilitating asynchronous interaction. Each agent is running independently on a Uvicorn server, communicating with the Project Manager via structured JSON messages. To expose the endpoint over a network, we used ngrok. The Project Manager Agent (PMA) is assigning tasks to these agents, collecting their individual responses, and then compiling a final document. This modular architecture as proposed in our initial proposal allows scalability and future integration of multiple autonomous additional agents without disrupting the core system functionality.

3.6 Prompt Engineering

To ensure the quality of the responses that are being generated by the agents we are using structured and task-specific prompts that attempt to maximize response relevance and accuracy. They are being used to constrain the models in order to maximise the response relevance and accuracy using predefined boundaries that are guiding the model outputs. Hence prompt engineering is facilitating clear, concise, factual responses from each agent that are tailored to the specific needs of the task. The prompt also defines the return format which is replicated for each agent to ensure scalability for more agents. We have provided an example of the Project Manager's prompt in the appendix.

4 Preliminary Results

4.1 Performance Evaluation

The model has been evaluated using several test cases to evaluate its ability to generate structured and relevant marketing, engineering and legal content. The primary criteria used in the evaluation included data organization, accuracy, problem solving capacity, content coherence, hallucination rate and usability. The generated documents were well structured, meeting the main marketing needs, maintaining high readability and logical flow. The system displayed low levels of hallucination, which means that the generated content remained relevant to the entry provided without manufacturing misleading or incorrect information. This was a critical aspect to ensure that the model produces high quality marketing materials suitable for professional use. In addition, several rapid variations have been tested to verify system robustness to deal with various user requirements. The model has successfully adapted to different content needs, including technical implementations, product descriptions, social media posts, and legal risk mitigation. The ability to generate personalized technical material from a wide variety of warnings indicates the flexibility of the system and potential for real-world application.

4.2 Technical performance and latency of the system

The first problem that was encountered in the test was speed of inference. The deployment of the model on personal and local machines and serves resulted in a delay in responses when the api was called by the product manager agent. This was not permitting real-time access at to the agent responses. The model relied on transformer architecture intermediaries and they require heavy computational capability for processing input to generate content in a timely manner. In the current pipeline, the model combats latency issues due to context window granularity and LLM content generation, which is computationally intensive. To bypass these problems, different deployment environments have been tried. When executed on GPU high-performance cloud computers, response times have been significantly boosted. Still, the best performance would be when the model would be used in cloud-based applications such as AWS. Other optimization techniques are also being investigated to improve local execution, including models and distillation techniques to reduce computational overload.

4.3 Scale model and improvement potential

Given that the present infrastructure and workflow showcases effective content generation features and provides a hierarchical representation of the product deployment schema, there is ample room for additional improvements, integrating larger LLMs with higher number of parameters once cloud-computing can be taken advantage of. Using more extensive pre-trained models such as GPT-4 which can be fine-tuned for the specific agents or specific thin adjustment domain transformers can improve accuracy, coherence, and contextual consciousness. The increase in the number of parameters will also result in higher comp costs and transactional payload and higher cloud -based inference. One way of counteracting this is implementing a hybrid system where a smaller parameter model operates on host machines for rudimentary tasks and aggregation, while difficult and compute intensive tasks such as prompt requests are processed through an API that leverages cloud-based models.

4.4 Pipeline documentation

Screenshots of some of the Project Manager prompts and SQLite database that was implemented to process the queries are included in the Appendix. The screenshots in the demonstration video show the user provided query, the processed version of the prompt generated by the project manager, the tasks sent down to the engineering, marketing and legal agent, and database interactions. A live demonstration of the model is uploaded to showcase the interaction with the model. A video recording of the demonstration was sent to the project's Google repository, which shows the end-to-end interaction and deployment. Test cases (sample documentation for an AI chatbot product) were synthesized and the product's technical documentation in full depth was aggregated with different immediate variations to showcase the model's potential to deal with different prompts and different degrees of complexity. A database is also created to highlight how the generated content is structured and stored in memory.

4.5 Areas for Improvement

The current agent response times must be faster, especially for local inference. Strategies such as pruning, quantization and optimized deployment can be used to this effect. Expansion and thin adjustment of the model-binding larger models or adjusting specific marketing data sets can improve content accuracy and contextual relevance and make the smaller Llama-3B models capable of running locally and providing high-quality relevant data as a response to the api calls. A robust front-end interface to provide real-time feedback and improve interactivity with the user can be deployed. This will contribute to a more turing user experience. Performance optimizations and additional improvements are necessary to improve the speed and scalability of the inference. By addressing these areas, our model can evolve into a highly efficient and friendly tool for automated product development.

5 Team Member's Contributions

Overall the development of the multi-agent architecture was a team effort. Muhammad was responsible for the Engineering Agent, setting up Llama and integrating the SQLite database, which will be reused for the other agents as well. Ahad worked on developing the Project Manager Agent (PMA) with features including task assignment, gathering responses and constructing the final documentation. Asad was responsible for the development of the Marketing Agent, which included running Gemma for market analysis, structuring the responses and exploring models and datasets for fine tuning the Marketing Agent. Hamza was responsible for the Legal Agent, which included structuring its responses to get accurate and useful legal background responses.

6 Conclusion and Future Work:

The proposed multi-agent approach showcases a streamlined way to divide tasks among specialized agents. The central PM Agent integrates the results produced by the Engineering, Marketing, and Legal Agents, which ensures clarity, coherence, and consistency using structured prompts. The model generates robust results using FastAPI, Uvicorn, and SQLite and models specialized for their tasks.

Moving forward, several enhancements and expansions can be used to scale and improve the multi-agent system. Firstly, finetuning the Llama and Gemma models using Low-Rank Adaption (LoRa) can allow a boost in domain-specific performance, reducing the computational overhead of the current architecture. This is aimed at gen-

erating more detailed responses and allowing the project to be used on a larger scale. Secondly, maintaining a database to store the results of previous iterations can personalize interactions and maintain context across projects, if needed. Finally, the marketing agent can also be upgraded to generate visual promotional content which can be used for marketing campaigns.

References

Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint*, arXiv:2305.14325.

S. Gandhi, M. Patwardhan, L. Vig, and G. Shroff. 2024. Budgetmlagent: A cost-effective llm multi-agent system for automating machine learning tasks. *arXiv preprint*, arXiv:2411.07464.

Krave. 2010. Madbe: A multi-agent digital business ecosystem. In *Proceedings of IEEE Xplore*.

Marvin Minsky. 1988. *Society of Mind*. Simon and Schuster, New York, NY, USA.

```python
def task_division_prompt(user_idea : str):

    return f"""
        You are an AI Project Manager. Your responsibility is to take a given business idea (provided by the user) and break it down into 3
        detailed tasks for the following agents:

        • Engineering Agent: Handles technical design, system architecture, and component designs. The agent does not have to give any code,
        but should provide a detailed technical overview of the system.
        • Marketing Agent: Focuses on market research, branding, and promotional strategy.
        • Legal Agent: Deals with compliance, risk assessment, and legal documentation.

        Your output must be a valid JSON object with exactly 4 key-value pairs. Each key is the name of an agent (use lowercase strings
        exactly as "engineering", "marketing", "legal"), and each value is a detailed prompt instructing that
        agent on its specific task.

        The JSON format should look like this:
        {{
            "engineering": "Short task prompt for the Engineering Agent.",
            "marketing": "Short task prompt for the Marketing Agent.",
            "legal": "Short task prompt for the Legal Agent.",

        }}
        Each task prompt should provide the original user idea and clearly specify objectives, deliverables, and any constraints derived from the user's business idea.
        Return only the JSON object with no additional text.
        The task length should be 2 lines.
        Business Idea: {user_idea}
        """
```

Figure 2: Project Manager Prompt

project_responses.db

project_responses.db

Filter 1 t… Rows: 13 Filter 13 rows… Upgrade to PRO

TABLES

project_respon…

	section	heading	sub_question	response
	Filter…	Filter…	Filter…	Filter…
1	Technical Architecture	Databases, APIs, and Frameworks	What databases, APIs,…	The ideal databases for this project would be relationa…
2	Engineering Team Structure	External Consultants vs. In-house	Should we consider hi…	Our decision to keep everything in-house will enable u…
3	Required Technical Tools & Stack	Microservices or Monolithic Architecture	Should we use micros…	The system should be designed to handle multiple bus…
4	Project Timeline & Milestones	Development Timeline	What would be a reas…	Develop an AI-powered chatbot system that can gener…
5	Feature Breakdown & Prioritiza…	Core Features	What are the core fea…	The core features that must be built initially are: - User…
6	Technical & Business Risks	Technical Challenges	What are the potentia…	The AI-powered chatbot system will require a robust a…
7	Technical & Business Risks	Scalability Concerns	What scalability conce…	Scalability concerns should be accounted for early in t…
8	Expertise & Knowledge Required	Domain Expertise	What level of domain …	This project requires a strong understanding of marke…
9	Technical Architecture	High Level System Architecture	What should be the hi…	The proposed system architecture should include the f…
10	Technical Architecture	Backend and Frontend Technologies	What backend and fro…	The suitable technologies for the backend include a na…
11	Required Technical Tools & Stack	Programming Languages and Integrati…	What programming la…	To design an AI-powered chatbot for marketing conte…
12	Engineering Team Structure	Required Engineers and Expertise	What kind of enginee…	A chatbot designed for small businesses to create aut…
13	Engineering Team Structure	Development Team Size	How many developers…	For the project, it is recommended to have at least 3 d…
14				

Figure 3: Engineering Agent Local Database Snapshot after Querying

ENGINEERING

Technical Architecture

High Level System Architecture

The proposed system architecture should include the following components:

1. User Input: A user-friendly interface for users to input their marketing content requirements, such as product descriptions, social media posts, and email campaigns.
2. Natural Language Processing (NLP) Module: An NLP module capable of understanding user inputs and extracting relevant marketing content, such as keywords, phrases, and sentiment analysis.
3. Content Generation Module: A module responsible for generating marketing content based on the extracted data, such as product descriptions, social media posts, and email campaigns.
4. Content Storage Module: A module that stores the generated marketing content, ensuring scalability and robustness.
5. Integration Module: A module that integrates with existing marketing tools and systems to facilitate seamless collaboration and data exchange.
6. Scalability Module: A module that ensures the system can handle a large volume

Backend and Frontend Technologies

The suitable technologies for the backend include a natural language processing (NLP) library such as NLTK or spaCy for text analysis and sentiment analysis, and a machine learning library like scikit-learn or TensorFlow for training and deployment of the chatbot model. For the frontend, a user-friendly interface such as React or Angular can be used to build a responsive and interactive web application. Additionally, a database like MongoDB or PostgreSQL can be used to store user input, marketing content, and chatbot responses. This will enable the system to be scalable, robust, and maintainable.

Required Technical Tools & Stack

Programming Languages and Integrations

To design an AI-powered chatbot for marketing content creation, you'll need to utilize cloud-based services such as AWS or Google Cloud for scalability and reliability. Popular programming languages like Python, JavaScript, or R can be used for natural language processing and data analysis. Integration with third-party services like Google's AI Platform,

Figure 4: Final Multi-Agent Generated Project Documentation