Lovelytics: Multi-Agent Approach to LLM Task Automation for Business Users

Alvina Yang
University of Toronto
alvina.yang@mail.utoronto.ca

Stephanie Lu
University of Toronto
steph.lu@mail.utoronto.ca

Julien Liang
University of Waterloo
jh2liang@uwaterloo.ca

Mateo Arcos
University of Toronto
mateo.arcos@mail.utoronto.ca

Zachary Tang
University of Toronto
zach.tang@mail.utoronto.ca

Jeff Lu
University of Toronto
jefff.lu@mail.utoronto.ca

Hannah Ye
University of Toronto
hannahh.ye@mail.utoronto.ca

Benson Yan
University of Waterloo
b58yan@uwaterloo.ca

Sina Fallah Ardizi
New York University
sinafallah98@gmail.com

Amr Alomari
University of Toronto
amr.alomari@mail.utoronto.ca

Jeremy Qu
University of Toronto
jeremy.qu@mail.utoronto.ca

***Abstract*—This paper addresses the challenge of automating business tasks using Large Language Models (LLMs) by focusing on two key aspects: generating high-quality prompts from unclear user input and executing tasks in a modular and scalable way. The system proposed combines DSPy (Declarative Self-Improving Programs)-driven prompt generation, which refines prompts based on feedback and task context, with a multiagent execution approach [1]. Unlike common industry practices, this system reduces manual effort by automating both prompt creation and task execution. The goal is to make AI-powered task automation accessible to non-technical users, allowing them to adopt LLMs into their daily workflow without the need for specialized knowledge. By democratizing task automation, the system opens up new possibilities for more efficient workflows across organizations.**

I. Introduction

Rapid adoption of artificial intelligence in business operations has fueled the demand for automating complex workflows using LLMs. Organizations increasingly rely on LLMs for document processing, customer service, and data analysis, seeking improvements in efficiency and scalability. However, LLM effectiveness depends on prompt quality, and poorly structured prompts often lead to ambiguous, incomplete, or misaligned outputs. This presents a significant barrier to this form of automation, particularly for non-technical users unfamiliar with prompt engineering.

This paper introduces a DSPy-driven framework for structured prompt generation, enabling users to convert vague automation requests into well-formed, context-aware instructions that improve LLM performance. Additionally, we develop a multi-agent task execution system that breaks down workflows into modular, interdependent steps, improving reliability and adaptability. Hosted on Lovelytics' Databricks Azure tenant, this system ensures secure, scalable automation with direct access to enterprise datasets. As a Databricks partner, Lovelytics enables seamless integration with enterprise workflows by allowing secure data retrieval and processing directly from the Databricks File System (DBFS).

A. Motivation

Despite advancements in AI, prompt engineering remains a major challenge, especially for nontechnical users. Vague instructions yield unreliable responses, missing context reduces accuracy, and multistep tasks often result in logical inconsistencies. These issues prevent organizations from fully leveraging LLMs for automation. Existing solutions, such as manually crafting prompts and heuristics, offer partial improvements but struggle with generalization and structured execution.

B. Problem Definition

This paper addresses the dual challenge of:

1) Generating structured high-quality prompts from ambiguous user input
2) Executing complex business automation tasks in a modular and scalable manner

To solve these challenges, we introduce a system that combines:

- DSPy-driven prompt generation, which dynamically refines prompts based on iterative feedback and task constraints, resulting in more effective instructions for LLMs.
- Multi-agent task execution, where specialized agents manage different workflow stages in parallel.

Unlike traditional prompt engineering methods that rely on manual refinement, our approach automates prompt optimization while managing multi-agent execution, minimizing human intervention, and increasing task reliability.

C. Limits of LLM Automation

The Occupational Information Network (O*NET) is a comprehensive online database of U.S. occupation information, maintained by the Department of Labor.

O*NET assists in prompt optimization by providing:

- A detailed breakdown of jobs into specific tasks and subtasks.
- Industry-specific terminology, responsibilities, and skill requirements.
- Information on job-specific technologies, skills, and tools.

Users interact with the O*NET database by entering their job title, allowing us to retrieve and suggest job-related tasks for automation while also gathering job-specific context to better understand their role. The automatability of a task by an LLM depends on its structure, complexity, and input/output requirements. Highly structured, rule-based, repetitive tasks, as well as those with a definitive correct answer, are generally automatable. In contrast, tasks that require real-world interactions, deep reasoning, or creativity are less suitable for automation. Table I below highlights key features of automatable tasks.

TABLE I
FEATURES OF TASKS AUTOMATABLE BY LLMS

Automatable by an LLM	Not Automatable by an LLM
Text-based and well-defined input/output	Real-time decision-making
Pattern recognition, generalization, and context-based reasoning	Highly specialized tasks
Tolerance for imperfection	Multimodal reasoning with real-world interaction

II. METHODOLOGY

The DSPy-driven prompt engineering component focuses on refining vague user requests into structured 'superprompts' that provide the model with clear instructions, contextual information, and defined constraints. These superprompts improve task accuracy by integrating external knowledge and breaking down complex tasks into manageable components. The multi-agent execution system then takes these structured prompts and efficiently executes them by dividing tasks into subtasks and assigning them to specialized agents. Together, these components ensure a scalable, context-aware approach to automating business processes using LLMs.

A. DSPy-Driven Prompt Engineering

DSPy is a framework for optimizing prompts for LLMs [2]. It provides an abstraction layer which allows developers to define tasks declaratively while automatically fine-tuning the prompts and reasoning strategies using data-driven optimization.

Our approach uses DSPy to transform vague user requests into structured 'superprompts' that provide the LLM with clear instructions and relevant context. Initial user prompts undergo evaluation to identify their suitability for specific task types, such as writing or reviewing. Superprompts integrate external resources such as PDFs and domain-specific data to enhance accuracy. Utilizing chain-of-thought reasoning, the DSPy implementation systematically leverages four key components:

1) Context from files, which tailors responses based on user-provided documents
2) Domain-specific context, aligning outputs with best practices
3) Task considerations, defining constraints and dependencies
4) Task subtasks, breaking down automation into structured steps

The refinement process begins by defining signatures, specifying task inputs and outputs. DSPy then generates candidate prompts and evaluates their effectiveness against predefined criteria. Feedback-based optimization ensures iterative improvements by incorporating high-quality prompt samples for training. The final optimized superprompts are processed by the multi-agent system, which involves feeding them into ChatGPT and evaluating performance.

B. Multi-Agent System for Task Execution

The multi-agent system (see Fig. 1) is responsible for executing tasks derived from DSPy-generated prompts. It does so by breaking workflows into structured subtasks handled by specialized agents. The system processes three key inputs: (1) the processed superprompt from DSPy, (2) instructional files, which contain instructions and important information, and (3) supplementary files, which provide useful contextual information via RAG-based retrieval.

The workflow initializes when the Planner Agent analyzes the 'superprompt' and creates the Task Plan, which divides the overall task into individual subtasks with execution steps, displaying the information on the frontend. The user can choose to modify the Task Plan by adding, removing, or adjusting subtasks in an interactive refinement loop before execution begins. Each subtask is assigned to an Executor Agent, which processes the instructional files and retrieves supplementary data to generate an output. These agents operate in parallel for efficiency. Once all subtasks are completed, the Merger Agent compiles them into a coherent final result. The Verification Agent then evaluates this output against predefined criteria. If it meets the required standards, it is finalized; otherwise, the process repeats up to a user-defined retry limit to refine the results.

In the second iteration of our design, we introduced an alternative approach: instead of iterative retries, the system runs five parallel executions of the full pipeline. A Selector Agent then evaluates and selects the best overall output, reducing computational overhead while improving reliability.

III. EVALUATION METHODS

We use two main approaches to evaluate the quality of our system: human evaluation and GPT-We use two main approaches to evaluate the quality of our system: human evaluation and GPT-based benchmarking. Each method focuses on different aspects of the generated output to provide a well-rounded understanding of the system's performance.

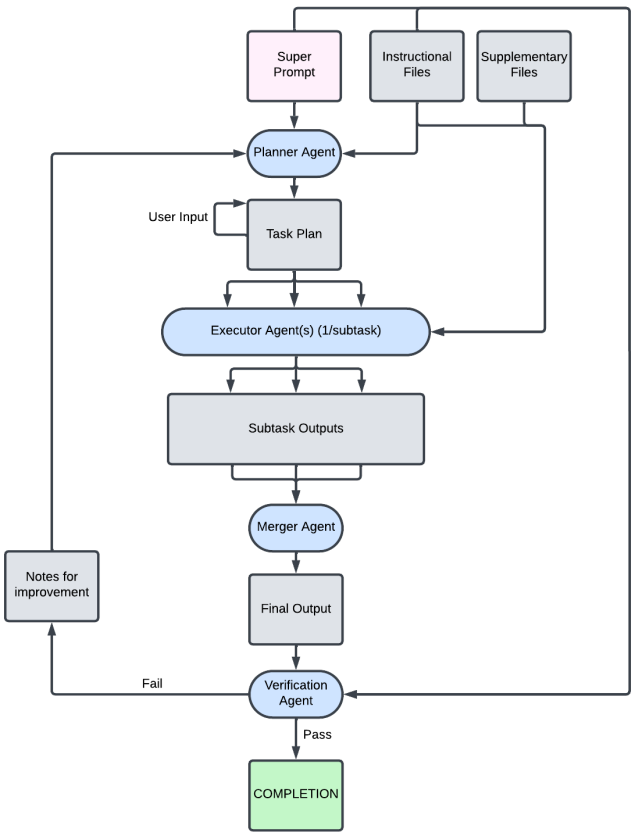


Fig. 1. Multi-agent system architecture

A. Human Evaluation

Human evaluation focuses on straightforward, objective criteria:

1) **Word count**: The total number of words in the output, averaged across multiple responses.
2) **Output format**: Checks whether the generated content matches the requested structure.
3) **Instruction adherence**: Evaluates whether the intermediate and final outputs follow the instructions provided in the task.

These criteria ensure that the system meets basic expectations, producing content in the correct form and following the given instructions. LangSmith was used to support human evaluation.

B. LLM-Based Benchmarking

LLM-based benchmarking focuses on a more detailed evaluation of content quality. In this method, advanced LLM agents, referred to as 'council members', are used to assess the generated outputs. These agents evaluate the responses based on an initial set of metrics. After reviewing the prompt and any provided instructional files, each 'council member' suggests 1-3 additional criteria that they believe are important for the evaluation. These new metrics are then added to the final list. The final score for each output is calculated by averaging the scores from all 'council members'. This approach enables a more comprehensive evaluation of the content, considering aspects such as relevance, clarity, and depth.

C. Relevant Work

The approach described above draws inspiration from two primary sources. First, the concept of using multiple LLM agents as a 'council' is discussed in the paper Language Model Council: Democratically Benchmarking Foundation Models on Highly Subjective Tasks [3]. This paper demonstrates how collaborative LLM benchmarking can provide a more comprehensive evaluation for writing-based tasks. The idea of using multiple LLM agents to collaboratively generate custom evaluation criteria and assess outputs was particularly appealing as it minimizes bias and contributes to a more balanced evaluation.

Second, the methodology outlined in the GitHub repository [4], which provides guidelines for evaluating writing quality, also influenced the approach. This repository emphasizes the use of LLMs to define evaluation metrics for writing tasks, further guiding the development of the benchmarking process.

D. Tasks for Benchmarking

For the benchmarking, the task automation system was tested using three different tasks:

- **Environmental History of Computing Essay**: This task includes both an instructional file and a supplementary file, providing context for generating the essay.
- **Literary Research Writing**: This task includes an instructional file and several different books and papers on a literary work, with information from competing sources that needs to be referenced in conjunction to each other.
- **Fire Safety Protocols**: Unlike the other tasks, this one lacks both instructional and supplementary files, making it more open-ended and testing the system's ability to generate relevant content autonomously.

E. Experimental Setup

To evaluate the performance of different systems against the proposed multiagent system, benchmarking was conducted on the output from the following models:

- GPT-4o-mini with direct prompting
- GPT-4o with direct prompting
- GPT-o3-mini-high with direct prompting
- The proposed system with GPT-4o-mini in the backend

Each model was tested three times to ensure consistent and reliable results.

IV. Results

The experimental results comparing the outputs of various models against our system are presented in the appendices. In general, our system demonstrates significantly superior performance compared to GPT-4o-mini and GPT-4o. It shows slight advantage over GPT-o3-mini-high, while running at a

fraction of the cost. We will focus our analysis on the first task: an essay on the Environmental History of Computing.

The execution of the Environmental History of Computing Essay task shows significant differences in model performance. The task involved writing a 2,000-word essay with instructional and contextual files for guidance.

The 4o-mini model generated only 420 words in bullet point format, failing to meet word count and structural requirements, with a benchmark score of 52.50. The 4o model performed better, generating 828 words in paragraph form, but still lacked the clear thesis-claim-evidence structure of an essay, earning a score of 78.33. The o3 model, a reasoning-based model, produced 1,704 words with paragraphs and met the structural requirements, earning a benchmark score of 82.78. However, while this model exhibits better performance, it is nearly seven times more expensive to run compared to our system.

The proposed system, which uses 4o-mini, produced 1,979 words in well-structured paragraphs and adhered to all instructions. With a benchmark score of 85.06, it outperformed all other models, showing that even smaller models can achieve high-quality outputs when optimized through task automation.

The difference in performance between directly prompting o3-mini-high versus the other models is primarily due to the reasoning strength of o3, which excels in content coherence. However, the model still struggles with instruction adherence. In contrast, our system leverages DSPy for prompt engineering and LangGraph for parallel execution and subtask decomposition, allowing 4o-mini to generate structured content with better instruction adherence.

A. Discussion

The primary goal of this work is to optimize the process of prompt creation and task automation. The DSPy-driven prompt engineering approach effectively converts vague task automation requests into structured superprompts, which in turn enhances the performance of LLMs.

Experimental results demonstrate that the proposed system outperforms baseline models across several evaluation metrics, including human evaluation and LLM-based benchmarking. By leveraging LangChain, the system also supports the parallel execution of subtasks, which significantly reduces latency, allowing for faster processing times and improving the overall efficiency of the task automation process.

V. Conclusion

This work presents a system that enhances task automation by utilizing a multiagent architecture in combimation with DSPy to transform vague task automation requests into structured superprompts, optimizing task execution. Experimentation shows that the proposed system outperforms baseline models across multiple evaluation metrics, including human and LLM-based assessments. The system's modular design, supported by LangChain, enables parallel automation of subtasks, enhancing both efficiency and the overall user experience.

In practice, DSPy lowers the barrier for non-technical users to automate tasks, which promotes a broader adoption of AI-driven workflows. By leveraging Databricks and Azure, the proposed system also ensures secure data access and smooth integration into existing enterprise ecosystems.

A. Future Work

Future work will focus on further improving the capabilities of the system, refining existing workflows and user experience, and extending its functionality in a few key areas:

1) **Improvement Pipeline:** One potential direction is to develop a system that focuses on improving existing work rather than completing new tasks. This could involve creating a pipeline that takes a current workflow as input, uses agents to analyze and refine it, and then outputs an improved version. This approach would enable the system to iteratively improve tasks and optimize existing processes over time, making it more adaptable to changing requirements.
2) **Effective Referencing:** Another area for future development is enhancing the system's ability to reference examples for task output generation, including tone, structure, and content formatting. By comparing task outputs with high-quality examples, the system could improve its ability to generate outputs that align more closely with user expectations. This would involve developing more advanced algorithms for contextual understanding and comparison, allowing for more effective use of external references during task execution.

These areas of improvement will further increase the flexibility and performance of the system, ultimately enabling broader application in real-world business automation scenarios.

References

[1] A. Yang, S. Lu, J. Liang, M. Arcos, Z. Tang, J. Lu, H. Ye, B. Yang, S. F. Ardizi, and A. Alomari, "Project Git Repository," 2024. [Online]. Available: https://github.com/alvina-yang/Lovelytics

[2] O. Khattab, A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Vardhamanan, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia, and C. Potts, "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines," 2023. [Online]. Available: https://arxiv.org/abs/2310.03714

[3] J. Zhao, F. M. P. del Arco, B. Genchel, and A. C. Curry, "Language Model Council: Democratically benchmarking Foundation models on Highly Subjective Tasks," 2025. [Online]. Available: https://arxiv.org/abs/2406.08598

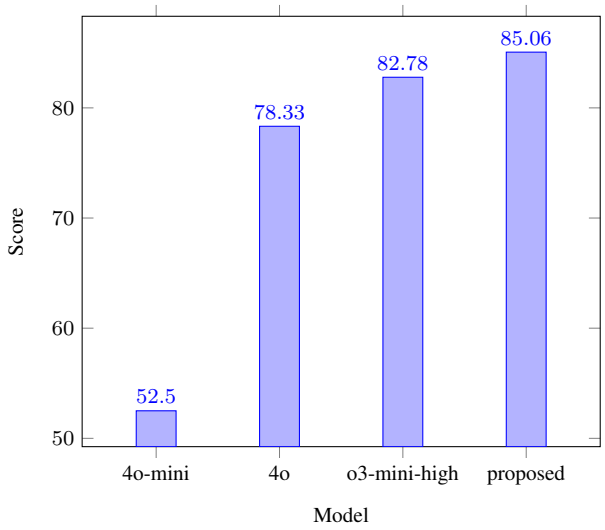
[4] L. Mazur and C. Norton, "LLM Creative Story-Writing Benchmark," 2025. [Online]. Available: https://github.com/lechmazur/writing

APPENDIX A

TABLE A1
BENCHMARK TASK #1 - HUMAN EVALUATION

Model	Word Count	Output Format	Instruction Adherance
4o-mini	420	Bullet Points	No
4o	828	Paragraphs	Yes
o3-mini-high	1704	Paragraphs	Yes
proposed	1979	Paragraphs	Yes

TABLE A2
BENCHMARK TASK #1 - LLM-BASED BENCHMARKING SCORES

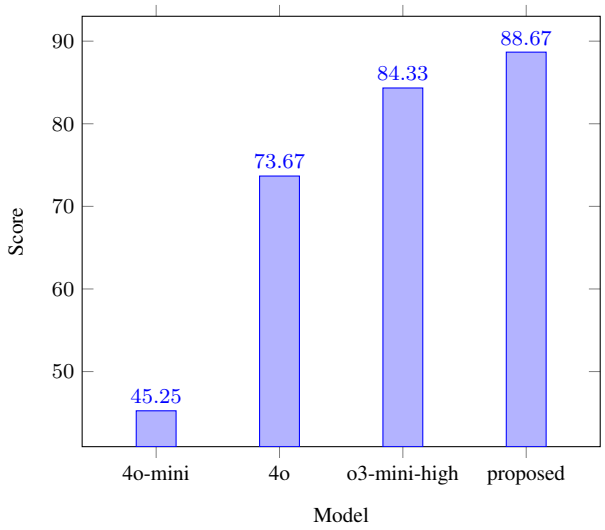


APPENDIX B

TABLE B1
BENCHMARK TASK #2 - HUMAN EVALUATION

Model	Word Count	Output Format	Instruction Adherance
4o-mini	533	Bullet Points	No
4o	917	Bullet Points	No
o3-mini-high	2154	Paragraphs	Yes
proposed	2261	Paragraphs	Yes

TABLE B2
BENCHMARK TASK #2 - LLM-BASED BENCHMARKING SCORES



APPENDIX C

TABLE C1
BENCHMARK TASK #3 - HUMAN EVALUATION

Model	Word Count	Output Format	Instruction Adherance
4o-mini	651	Bullet Points	Yes
4o	354	Bullet Points	Yes
o3-mini-high	777	Bullet Points	Yes
proposed	2896	Paragraphs	Yes

TABLE C2
BENCHMARK TASK #3 - LLM-BASED BENCHMARKING SCORES