GenAI and financial data alert systems

GenAI Insight Overlays

Candidate Number: KYHL7[1]

MSc. Computer Science

Internal Supervisor: Prof. Lewis D. Griffin

Submission date: 8 September 2025

[1]**Disclaimer:** This report is submitted as part requirement for the MSc. Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

TODO: Summarise your report concisely.

Contents

Chapter 1

Introduction

According to a study by Forrester Consulting, knowledge workers spend 30% of their time - that is, 2.4 hours a day - searching for information within their own company[1]. This is a significant amount of time that could be better spent making decisions and taking action. As such, the aim of this project is to explore the capabilities of Large Language Models ("LLMs") to access a company's data, extract operational figures and provide initial insights to managers, directly to their inbox, saving time from the operational task of data gathering, and allowing them to focus on the more impactful parts of their roles - making data-informed decisions.

To perform this exploration, the goal is to creating a proof of concept system that acts as a junior analyst (or "AI Analyst") that can autonomously generate reports about a specific Key Performance Index (KPI), at set time intervals, based on the company's data. In addition to the AI Agent, the system will have a simple user interface, where the user can set the KPI(s) to report on, and the interval at which their analyst should run.

For this, the project builds on the recent evolution of AI agents and agentic systems, as paradigms that allow the execution of complex, multi-step workflows autonomously. Nevertheless, this specific use case poses a few challenges to the current stat of the art in GenAI agents.

For these reports, the AI Analyst must go beyond simply retrieving high-level KPIs. Instead, it must also provide granular details that uncover deeper operational insights. For example, rather than merely reporting that sales are down by 10%, the Analyst should also identify which products or regions are driving this decline.

This type of work is inherently open-ended. There is no single "correct" answer, or even path to an answer, nor a clear point to stop, and the AI Analyst must adapt its approach depending on its own intermediate findings. Crucially, for the reports to be truly useful, the agent must not just report the data, but drawing causal inferences that can point the manager in the right direction to solve any issues found, something that remains a known challenge for LLMs [8].

Moreover, this kind of internal analysis is not likely to be included in LLM's training sets. Companies rarely publish their internal data or reports, and the requirements of operational reports are more granular than financial reports, which might be available for public companies. This makes the task a strong candidate for fine-tuning, but due to the lack of suitable training data this is outside of the project scope.

With all of this in mind, the project was executed in three consecutive stages: First, a Research stage, focused on understanding the state of the art in AI agent orchestration and prompting, as well as the software tools available; the results of this stage can be found in Chapter 2. Second, a Requirements and Analysis stage, where together with the client, Alvarez & Marsal, we defined

the scope of the proof of concept; the results of this stage can be found on Chapter 3. This was also the bases for the evaluation framework for the project, which is described in Chapter 4. Third, a Design and Implementation stage, which consisted on two separate iterations based on different approaches to agentic orchestration; the details of this stage can be found on Chapter 5. Finally, Chapter 6 contains the Reflections from the process of creating the proof of concept system, as well as the potential future work to create a production-ready system.

Chapter 2

Research

Research for this project was be split in two main lines. First, understanding the fundamentals of building applications with Large Language Models. For this, the project was largely influenced by Chip Huyen's book, AI Engineering, which "provides a framework for adapting foundation models, which include both large language models (LLMs) and large multimodal models (LMMs), to specific applications"[3]. In particular, a few learnings from the book were crucial in the development stage, which give structure to Section 2.1:

1. Its highlight of the importance, and difficulty, of evaluating AI systems
2. Its guideline of Prompt Engineering best practices
3. Its guideline of Retrieval Augmented Generation
4. Its overview of AI Agents

That said, this chapter does not intend to replicate the book's content, instead it will describe the most important learnings grasped from it, and add the learnings from other relevant information sources.

Afterwards, Section 2.2 will describe the research performed to define the technology stack used to develop the proof of concept, which per the client's request was built in a python environment.

2.1 Large Language Models and Agents Learnings

2.1.1 AI System Evaluation

In AI Engineering, Chip Huyen highlights two main reasons why evaluating the outputs of LLMs is harder than evaluating traditional machine learning models.

First, as tasks become more sophisticated, the more time it takes to evaluate them. As she puts it, "[y]ou can no longer evaluate a response based on how it sounds. You'll also need to fact-check, reason, and even incorporate domain expertise". Secondly, the open-ended and probabilistic nature of LLMs "undermines the traditional approach of evaluating a model against ground truths". There are too many correct responses, so it is impossible to have a comprehensive list of correct outputs as ground truth.

These problems are exacerbated when it comes to AI Agents, as they tackle more complex tasks. This is an on-going research field but, according to [5], the focus of evaluation so far has

been "predominantly focused on accuracy metrics that measure task completion success". This focus has been criticised for many reasons, particularly for limiting the evaluation to a task being completed or not, and not included other metrics such as the efficiency of the completion.

This type of measure is called "exact evaluation", because it has no ambiguity - a task is complete or it is not - and can be viewed more generally to include any type of rules-based evaluation where, as the name indicates, a set or rules are defined and the output is measured against it adherence to these rules.

The alternatives to this type of evaluation are what has been called "objective evaluations", where an external validator that reviews the agent's output and "judges" it, assigning it a score based on some type of defined criteria - for example completeness or clarity. So far, studies such as [9] have found that this judgment can be performed by either LLMs (LLM as a judge) or Humans (Human as a judge), with high correlations in the scores given. Nevertheless, [2] has also found that human's personal preferences can affect the score given to an output.

Moreover, other authors such as [10] criticize any approach for evaluating Agents that relies solely on the final output of their process. They believe that agents shouldn't only be evaluated on their final output but on the intermediate steps taken, and thus propose agent as a judge -a methodology that uses agents to enable intermediate feedback- as a better alternative.

Evidently, despite the lack of a clear "best" strategy, evaluation remains a key ingredient in creating AI systems to ensure their reliability. The AI Engineering book goes as far as proposing a development methodology called Evaluation Driven Design (EDD). Similarly to Test Driven Design, in EDD the evaluation criteria for the application are defined before building the application, allowing safe iterations. Moreover, her recommendations align with [10] in the need to not only evaluate the final output, but also every component of the system. As will be discussed in detail in Chapter 4, this project followed the recommendations of said book.

But how do you define said evaluation criteria? Huyen proposes thinking in three categories which, despite being targeted for evaluating LLMs and not agents, can be adapted as follows:

- Domain-Specific Capacity: Evaluate the ability of the agent to perform tasks specific to the problem domain, such as coding or finance.
- Generation Capacity: Evaluate the quality of the text being generated, including the factual consistency of the outputs (that is, that the facts in the response are correct).
- Instruction-Following Capacity: Evaluate how well the specific instructions of a prompt are followed.

Choosing the Right Model

The difficulty in evaluation of LLMs and Agents highlights one specific buy key issue - how do you choose the right model, or at the very least model family, to power an agent?

The literature points towards using specialized model for tasks that require domain specific capabilities, such as data analysis, and more general models for task that are common in the general training data, such as coding, tool calling or general summarization (e.g., writing the final report). Studies such as [4] and [7] have shown that smaller models specifically trained for financial tasks outperform larger models in financial benchmarks.

Nevertheless, [6] created a financial benchmark that includes finance specific tasks such as forecasting market trends, predict asset movements, or analysing causal relationships in financial events, which closely resemble the types of task required in this project. Their evaluations of

general models such as OpenAI's GTP-4o or o4-mini produced strong results (above 80%) for these tasks, and as-such, these are used for development in this project. This is in line with the client's preferences of using widely available models.

2.1.2 Prompt and Context Engineering

Prompt Engineering (CoT)

Context Engineering

CONTEXT ENGINEERING - SUMMARIZATION?

2.1.3 Retrieval Augmented Generation

KEY - Agentic RAG; we

2.1.4 Agent Architectures and Orchestration

Agents and agentic orchestration (ReAct, Reflexion, Microsoft Magentic One)

2.2 Software Tools

While the project involved numerous decisions about tools, such as libraries to use for certain tasks, or the type of database to use to store user preferences, this section will only detail the key decisions that had an system-architecture impact; that is: the frameworks for the user interface and agent orchestration, and the tool to schedule agent runs. They will be presented in this chapter in order of importance.

Note that the client had a preference for using Azure/Microsoft products for any infrastructure requirements, due to its enterprise readiness. While the proof of concept will not be deployed, Azure was used as inference provider.

2.2.1 AI Orchestration Frameworks

Selecting the appropriate orchestration framework was the most crucial decision for the project, and the key guiding point for this was the size of a framework's community, which is often indicated by the number of GitHub stars, as it can significantly influence the availability of resources such as documentation, tutorials, and community support. A larger community typically leads to more comprehensive examples and a broader base of shared knowledge, facilitating smoother development and troubleshooting processes.

As such, Table 2.1 compares several prominent Python frameworks for agentic orchestration, highlighting their community size and the implications for developers seeking robust support and resources.

Additionally, the Table 2.2 provides some findings on the same frameworks' functionality, ideal use cases, and the pros and cons associated with each.

As will be explained in more detail later, the development of the project consisted of two iterations. The first iteration tried to take advantage of Microsoft's MagenticOne orchestration architecture, and that, in combination with the use of Azure as inference provider, made it an easy choice to

Framework	GitHub Stars	Community Notes
LangChain / LangGraph	113.6k	Extensive tutorials, active forums, and a wide array of integrations.
AutoGen	43.1k	Used to have support from Microsoft, with growing number of examples, but currently in the process of being merged to Semantic Kernel.
OpenAI SDK	8.6k	Official OpenAI support, with increasing community contributions.
CrewAI	35.8k	Active development with a focus on multi-agent collaboration.
Semantic Kernel	25.8k	Backed by Microsoft, offering enterprise-grade features, but documentation focused on C#.

Table 2.1: Comparison of Python Agentic Orchestration Frameworks

Framework	Functionality	Use Cases	Pros and Cons
LangChain / LangGraph	Modular pipeline for LLMs, embeddings, tools	Complex workflows, data pipelines	Pros: Extensive integrations, strong community support; Cons: Steep learning curve, large framework size
AutoGen	Multi-agent orchestration with conversational agents	Multi-agent systems, collaborative tasks	Pros: Easy to implement multi-agent system, original implementation of MagenticOne; Cons: In process of being merged to Semantic Kernel
OpenAI Agents SDK	Lightweight framework with agents, handoffs, and guardrails	Rapid prototyping, simple agent workflows	Pros: Minimal abstractions, easy to learn; Cons: Heavily geared towards using the full OpenAI ecosystem
CrewAI	Role-based collaboration with task management	Team-based workflows, sequential tasks	Pros: Intuitive design, good for structured workflows; Cons: Less flexibility for complex scenarios, smaller community
Semantic Kernel	Enterprise-grade framework	Enterprise applications, integration with Microsoft tools	Pros: Strong enterprise support, robust features, includes a default implementation of MagenticOne; Cons: Limited Python support, smaller community

Table 2.2: Comparison of Python Agentic Orchestration Frameworks

use Semantic Kernel. On the contrary, the second iteration's approach is to build a detailed workflow with careful management of the data used in context, thus making LangChain/LangGraph a better fit, not just for its functionality but for the amount of documentation and examples available to support building this more detailed workflow.

2.2.2 Scheduling Tool

The project requires the AI agent to run at set intervals, as defined by the user preferences. Two primary scheduling alternatives were considered: Celery and cron.

Celery is a distributed task queue that supports asynchronous and real-time task execution, with advanced features such as retries, task prioritization, and integration with message brokers like Redis or RabbitMQ. While powerful, Celery introduces additional system complexity and overhead, requiring configuration of worker processes and a message broker.

In contrast, cron is a native, time-based scheduler available on Unix-like systems, designed

specifically for executing tasks at fixed intervals. Cron is straightforward to configure, persistent across system reboots, and does not require any additional services.

Given the project's scope as a proof of concept, and the requirement of periodic execution at set intervals, cron was selected as the more practical solution.

2.2.3 User Interface Framework

Finally, for setting up the configuration interface, several Python frontend options were considered. The main alternatives evaluated were Streamlit, Gradio, and FastAPI with Jinja2 templates. While Streamlit and Gradio offer very quick setup and ease of use for prototypes, FastAPI with Jinja2 was chosen for this project due to its high customizability and the ability to easily scale the prototype into a full production application.

Feature	**Streamlit**	**Gradio**	**FastAPI + Jinja2**
Primary Use Case	Interactive dashboards	ML model interfaces	API-driven web apps
Ease of Setup	Very easy; minimal code	Very easy; minimal code	Moderate; requires setup
Customization	Limited UI customization	Limited UI customization	High; full control
Performance	Moderate	Moderate	High; asynchronous support
Extensibility	Limited	Limited	High; supports plugins
Deployment	Streamlit Cloud	Gradio Hub	Flexible (e.g., Docker, Cloud)
Best For	Rapid prototyping	Quick ML demos	Scalable, production-ready apps

Table 2.3: Comparison of Python Frontend Frameworks for AI Agent Configuration

Chapter 3

Requirements and Analysis

Chapter 4

Evaluation and Testing

This particular approach is not relevant for the aims of this project, as evaluating a report output requires reviewing the structure of the report, the relevance of the insights extracted and the correctness of the data extracted. Again, this is relevant for the type of task the AI Analyst will perform, as it would be impossible to create a comprehensive set of all possible reports that could be generated.

Because this is a proof concept, we do not want to spend too long building an evaluation system that might not be used in production, hence the use of human as a judge and rules-based evaluations for automated evals.

Chapter 5

Design and Implementation

Chapter 6

Reflections

6.1 Achievements

Summarise the achievements to confirm the project goals have been met.

6.2 Evaluation

Evaluation of the work (this may be in a separate chapter if there is substantial evaluation).

6.3 Future Work

How the project might be continued, but don't give the impression you ran out of time!

Bibliography

[1] Forrester Consulting. *The Crisis of Fractured Organizations: How Teams Can Address Organizational Misalignment & Achieve More In The Modern Work Environment*. Thought Leadership Paper. Commissioned by Airtable. Forrester Research, Inc., Dec. 2022. URL: https://www.airtable.com/lp/resources/reports/crisis-of-the-fractured-organization.

[2] Yebowen Hu et al. *DecipherPref: Analyzing Influential Factors in Human Preference Judgments via GPT-4*. 2023. arXiv: 2305.14702 [cs.CL]. URL: https://arxiv.org/abs/2305.14702.

[3] Chip Huyen. *AI Engineering*. USA: O'Reilly Media, 2025. ISBN: 978-1801819312.

[4] Zixuan Ke et al. *Demystifying Domain-adaptive Post-training for Financial LLMs*. 2025. arXiv: 2501.04961 [cs.CL]. URL: https://arxiv.org/abs/2501.04961.

[5] Naveen Krishnan. *AI Agents: Evolution, Architecture, and Real-World Applications*. 2025. arXiv: 2503.12687 [cs.AI]. URL: https://arxiv.org/abs/2503.12687.

[6] Guilong Lu et al. *BizFinBench: A Business-Driven Real-World Financial Benchmark for Evaluating LLMs*. 2025. arXiv: 2505.19457 [cs.AI]. URL: https://arxiv.org/abs/2505.19457.

[7] Kota Tanabe et al. "Enhancing Financial Domain Adaptation of Language Models via Model Augmentation". In: *2024 IEEE International Conference on Big Data (BigData)*. IEEE, Dec. 2024, pp. 6661–6669. DOI: 10.1109/bigdata62323.2024.10825292. URL: http://dx.doi.org/10.1109/BigData62323.2024.10825292.

[8] Lei Wang and Yiqing Shen. "Evaluating Causal Reasoning Capabilities of Large Language Models: A Systematic Analysis Across Three Scenarios". In: *Electronics* 13.23 (2024). ISSN: 2079-9292. DOI: 10.3390/electronics13234584. URL: https://www.mdpi.com/2079-9292/13/23/4584.

[9] Ruiqi Wang et al. "Can LLMs Replace Human Evaluators? An Empirical Study of LLM-as-a-Judge in Software Engineering". In: *Proceedings of the ACM on Software Engineering* 2.ISSTA (June 2025), pp. 1955–1977. ISSN: 2994-970X. DOI: 10.1145/3728963. URL: http://dx.doi.org/10.1145/3728963.

[10] Mingchen Zhuge et al. *Agent-as-a-Judge: Evaluate Agents with Agents*. 2024. arXiv: 2410.10934 [cs.AI]. URL: https://arxiv.org/abs/2410.10934.

Appendix A

Other appendices, e.g., code listing

Put your appendix sections here