# 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[3]. 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") who autonomously generates 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 [17].

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 5. Third, a Design and Implementation stage, which was performed iteratively; the details of this stage can be found on Chapter 4. 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" [6]. 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 [8], 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 [18] 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, [5] has also found that human’s personal preferences can affect the score given to an output.

Moreover, other authors such as [21] 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 [21] 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 5, 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 [7] and [15] have shown that smaller models specifically trained for financial tasks outperform larger models in financial benchmarks.

Nevertheless, [11] 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

According to Chip Huyen, "[a] prompt is an instruction given to a model to perform a task". One of the main reasons to build an evaluation framework is to be able to perform Prompt Engineering, or the process of tweaking prompts and context to get the desired outcome from an LLM. This step is so important, that model providers often provide detailed guides on how to engineer prompts for their specific models, such as [12] from OpenAI, which guided the promp engineering process for this project. Most of guides, nonetheless, include two techniques that have become cornerstones for this process.

First is "few-shot" prompting, introduced by [1]. This paper, by Brown et al., demonstrated that language models can learn a desired behaviour from examples in within the prompt; in more colloquial terms, there is a benefit to providing examples in prompts.

Second, there is "Chain-of-Thought" prompting, introduced by [19]. In this type of prompt, the model is asked to perform a step by step process before providing an output - either steps pre-defined in the prompt or defined by the model during execution. Wei et al. where able to achieve state of the art results using this technique.

Note that these techniques are not mutually exclusive, and prompts for complex tasks might include both of them, or none, depending on the specific model used (e.g., reasoning models might not need explicit chain of thought prompting).

#### Context Engineering

Borrowing AI Engineering's definition, the context is "the information provided to the model so that it can perform a given task". While the instruction (Prompt) remains static for any query, the information (Context) varies.

One key pattern for context engineering is Retrieval Augmented Generation (RAG), which according to AI Engineering can be considered as "a technique to construct context specific to each query". The pattern of retrieving and then generating was first introduced by Chen et al. in [2], where their system would retrieve Wikipedia pages relevant to a question and add their content to a model's context, and then fully coined by [10].

While these two papers define the technique with the use of text, this pattern can be applied to any type of information. Most relevant to this project, tabular data can also be retrieved and provided as part of the context.

Finally, adding information is not the only way to manage a task's context. In perhaps the most important finding for this project, [16] demonstrates that as the size of the context grows closer to the model's limit, retrieval and reasoning become more biased towards data that is near the end of the context, and the model more easily fails to retrieve information provided in previous parts of the context.

### 2.1.3 Agent Architectures and Agentic Orchestration

According to [13], "AI Agents can be defined as autonomous software entities engineered for goal-directed task execution within bounded digital environments", usually with an LLM as their "core reasoning component". They distinguish AI Agents from Agentic AI, an "emerging class of systems [that] extends the capabilities of traditional AI Agents by enabling multiple intelligent entities to collaboratively pursue goals through structured communication, shared memory, and dynamic role assignment". This subsection summarizes research findings for state of the art architectures of both.

### Agent Architecture

The basic structure of an agent is described in AI Engineering as an LLM, which can perform certain actions (through tools) in a defined environment. The LLM acts as the brain of this agent, processing the information it receives - e.g., the task given by the user or results from its own tools - and generating plans to perform the task.

The plans designed by the LLM can be performed in many ways, for example, tools can be called in parallel or sequentially, or the LLM and tools call be called in a Loop until the LLM itself decides the task has been completed. This designed is generally called the Agent's Architecture.

One of the key problems for any Agent Architecture is how to build in ways to adjust the plans laid out by the LLM, allowing them to correct potential errors. On this, one of the most common patterns is ReAct (Reasoning and Acting), proposed by [20]. In this pattern, two steps are taken in a loop until the task is completed: a Reasoning step, which encompass both planning and reflecting on tool outputs, and an Action step, where the next step in the plan is executed.

This architecture was later expanded upon in [14], in their proposed Reflexion architecture. In Reflexion, the agent has separate steps for evaluating the output of its tools and self-relect on what went wrong, on each iteration, after evaluation and reflection, the agent proposes a new plan, and then executes the next step on that plan.

### Agentic Orchestration Architecture

As described by [13], Agentic AI allows multiple AI Agents to collaborate for more complex goals. Agentic Orchestration thus refers to the way that the collaboration between agents is structured.

There are numerous ways to structure this collaboration. Some of the best known are, as described as part of LangGraph's documentation in [9], Networks (or Swarms) - where all agents communicate in a single group - and Supervisor - where an LLM acts as the coordinating node. But there are numerous ways of coordinating the work across agents, according to a specific task needs.

For instance, in [4], a team of Microsoft researchers described Magentic-One, a generalist system that demonstrated state of the art performance in multiple agentic benchmarks without changing how the agents collaborate. The architecture is described by them as "a multi-agent architecture where a lead agent, the Orchestrator, plans, tracks progress, and re-plans to recover from errors. [...] Moreover, Magentic-One's modular design allows agents to be added or removed from the team without additional prompt tuning or training, easing development and making it extensible to future scenarios".

One of the most interesting parts about Magentic-One is that it applies the learnings from [14] to multi-agent orchestration, that is, it has explicit steps to evaluate outputs, analyze what went wrong and re-plan. The architecture was open sourced by Microsoft and, for these reasons, was used extensively in this project for its generalist capabilities.

## 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: the frameworks used for 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 (i.e., LLMs calls are performed through Azure AI Foundry).

### 2.2.1 Agent 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. Community size 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.

| **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

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 Magentic-One orchestration architecture, and that, in combination with the use of Azure as inference provider, made it an easy choice to 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.

| **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 Magentic-One; 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 Magentic-One; Cons: Limited Python support, smaller community |

Table 2.2: Comparison of Python Agentic Orchestration Frameworks

### 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 (through the python crontab library).

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 if needed.

| **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 |
| **Recommended 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

## 3.1 Problem Statement

## 3.2 User Personas

## 3.3 Requirements

### 3.3.1 Functional Requirements

**Sales Manager**

| **ID** | **Requirement** | **Priority** |
|---|---|---|
| FR-1 | The system shall allow a Sales Manager-type user to set up their Sales scope (e.g., a specific region or product) | Must Have |
| FR-2 | The system shall be able to generate a Sales performance report based on the user's preferences | Must Have |
| FR-3 | The system shall be able to identify "special cases" in Sales trends and perform a deeper analysis | Must Have |
| FR-4 | The system shall be able to provide simple Sales forecasts based on current trends | Should Have |
| FR-5 | The system shall be able to recommend potential actions to remedy "special cases" in Sales trends | Should Have |
| FR-6 | The system shall be able to provide simple Sales forecasts based on the remedy actions recommended by itself | Could Have |
| FR-7 | The system shall be able to provide advanced Sales forecasts using regression techniques or any other statistical method | Won't Have |

**CFO**

| ID | Requirement | Priority |
|---|---|---|
| FR-8 | The system shall allow a CFO-type user to set up a list of KPIs they follow | Won't Have |
| FR-9 | The system shall be able to generate a report for the user's KPI list | Won't Have |
| FR-10 | The system shall be able to provide simple forecasts for each KPI based on current trends | Won't Have |
| FR-11 | The system shall be able to identify "special cases" in the trend of any KPI and perform a deeper analysis | Won't Have |
| FR-12 | The system shall be able to recommend potential actions to remedy "special cases" in each KPI's trend | Won't Have |
| FR-13 | The system shall be able to provide simple forecasts for each KPI based on the remedy actions recommended by itself | Won't Have |
| FR-14 | The system shall be able to provide advanced forecasts for each KPI using regression techniques or any other statistical method | Won't Have |

**User Type Independent Requirements**

| ID | Requirement | Priority |
|---|---|---|
| FR-15 | The system shall email the resulting report to the email addresses configured by the user | Must Have |
| FR-16 | The system shall include both financial and operational information in a report | Must Have |
| FR-17 | The system shall be able to load information from a CSV file | Must Have |
| FR-18 | The system shall be able to load information from other files provided by the user | Won't Have |
| FR-19 | The system shall be able to load information from a set of reliable online information sources | Won't Have |

### 3.3.2 Non-Functional Requirements

| ID | Requirement | Priority |
|---|---|---|
| NFR-1 | The system shall be extensible to other sources of information | Must Have |
| NFR-2 | The system shall run autonomously at fixed intervals | Must Have |
| NFR-3 | The system shall allow the user to update the fixed interval for runs | Must Have |
| NFR-4 | The system shall provide a user interface to update its configuration | Should Have |
| NFR-5 | The system shall provide a user interface that is clear and easy to use | Should Have |
| NFR-6 | The system shall allow multiple users to set up their reports | Could Have |
| NFR-7 | The system shall allow users to authenticate before updating their report preferences | Won't Have |

# Chapter 4

# Design and Implementation

NOTE: Talk about the experiments here.

NOTE: One of the most interesting questions is how reusable is the system that was created as proof of concept. Some of the first prompt attempts were very general, with the idea that the client could only change the description of the KPI to extend the usability - this is called a “zero-shot prompt” in LLM parlance. These prompts were, generally, not successful, which is in line with commentary on both the literature and prompting guides by LLM providers - even as models have advanced. Nevertheless, it is possible to envision designing the prompt system in a way that makes it easy to inject different examples / terms depending on the request type.

# Chapter 5

# Evaluation and Testing

Evaluating task completion 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.

Also this system can have very different requests, ex. Product vs Country, so we can't create an evaluation that is too detailed - but this has the issue of still not being able to measure how useful that specific report is for decision making.

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.

Also, note that we did continue running experiments after getting to 100%, but those were not recorded as they always resulted in 100% eval - so no point.

# Chapter 6

# Reflections

As a refresher, the aim of this project was to explore the capabilities of LLMs to access a company's data, extract operational figures and provide initial insights to managers, which was to be done by creating a proof of concept system that acts as a junior analyst who autonomously generates reports about a specific KPI, with a simple user interface where the user can set the specific needs for their report and the interval at which the analyst should run.

Success, in this particular project, had two complementary angles. The more tangible angle was to successfully create the proof of concept system, with the set of requirements discussed in Chapter 3. The other, more open-ended angle, was to gather insights from building the proof of concept about using LLMs for this type of task, which can be useful for the client's future projects.

The project can be considered a success on both fronts. Section 6.1 will reflect on the requirements of the proof of concept system, how many were actually included in the system and why others were not. Yet, as mentioned before, with AI Agents it is important to differentiate between a the functionality existing and it being useful, which is where Evaluations come into play. Section 6.2 will thus reflect on the evolution of the key Evaluation metrics, and through this will explore the question of using LLMs for this type of analysis, and the key learnings from the project. Finally, Section 6.3 will reflect on how the proof of concept system can be improved upon to arrive at a production-level application.

## 6.1 Proof of Concept Requirements Achievement

In general terms, the proof of concept system includes all key requirements. It achieved 100% of both Must Have and Should Have requirements, and 1 of the 2 Could Have requirements. Additional, the system can create reports for Total Sales, which is a KPI within the scope of the CFO, showing that the system as designed can be used as a base for further development of functionality for this type of user. However, implementing this type of user was outside the scope of this project.

The only Could Have requirement that was excluded from the proof of concept was FR-6. This was decided as the agent does not reliably recommended remedy actions that are measurable, but could be addressed considering the insights described in Section 6.2 if needed as part of the production system.

### 6.1.1 Functional Requirements

**Sales Manager User**

| ID | Requirement | Priority | Included |
|---|---|---|---|
| FR-1 | The system shall allow a Sales Manager-type user to set up their Sales scope | Must Have | Yes |
| FR-2 | The system shall be able to generate a Sales performance report based on the user's preferences | Must Have | Yes |
| FR-3 | The system shall be able to identify "special cases" in Sales trends and perform a deeper analysis | Must Have | Yes |
| FR-4 | The system shall be able to provide simple Sales forecasts based on current trends | Should Have | Yes |
| FR-5 | The system shall be able to recommend potential actions to remedy "special cases" in Sales trends | Should Have | Yes |
| FR-6 | The system shall be able to provide simple Sales forecasts based on the remedy actions recommended by itself | Could Have | No |
| FR-7 | The system shall be able to provide advanced Sales forecasts using regression techniques or any other statistical method | Won't Have | No |

Table 6.1: Sales Manager Type User Functional Requirements Achievement

**CFO User**

| ID | Requirement | Priority | Included |
|---|---|---|---|
| FR-8 | The system shall allow a CFO-type user to set up a list of KPIs they follow | Won't Have | No |
| FR-9 | The system shall be able to generate a report for the user's KPI list | Won't Have | No |
| FR-10 | The system shall be able to provide simple forecasts for each KPI based on current trends | Won't Have | No |
| FR-11 | The system shall be able to identify "special cases" in the trend of any KPI and perform a deeper analysis | Won't Have | No |
| FR-12 | The system shall be able to recommend potential actions to remedy "special cases" in each KPI's trend | Won't Have | No |
| FR-13 | The system shall be able to provide simple forecasts for each KPI based on the remedy actions recommended by itself | Won't Have | No |
| FR-14 | The system shall be able to provide advanced forecasts for each KPI using regression techniques or any other statistical method | Won't Have | No |

Table 6.2: CFO Type User Functional Requirements Achievement

**User Type Independent Requirements**

| ID | Requirement | Priority | Included |
|---|---|---|---|
| FR-15 | The system shall email the resulting report to the email addresses configured by the user | Must Have | Yes |
| FR-16 | The system shall include both financial and operational information in a report | Must Have | Yes |
| FR-17 | The system shall be able to load information from a CSV file | Must Have | Yes |
| FR-18 | The system shall be able to load information from other files provided by the user | Won't Have | No |
| FR-19 | The system shall be able to load information from a set of reliable online information sources | Won't Have | No |

Table 6.3: User Type Independent Functional Requirements Achievement

### 6.1.2 Non-Functional Requirements

| ID | Requirement | Priority | Included |
|---|---|---|---|
| NFR-1 | The system shall be extensible to other sources of information | Must Have | Yes |
| NFR-2 | The system shall run autonomously at fixed intervals | Must Have | Yes |
| NFR-3 | The system shall allow the user to update the fixed interval for runs | Must Have | Yes |
| NFR-4 | The system shall provide a user interface to update its configuration | Should Have | Yes |
| NFR-5 | The system shall provide a user interface that is clear and easy to use | Should Have | Yes |
| NFR-6 | The system shall allow multiple users to set up their reports | Could Have | Yes |
| NFR-7 | The system shall allow users to authenticate before updating their report preferences | Won't Have | No |

Table 6.4: Non-Functional Requirements Achievement

## 6.2 Evaluation Results and Key Insights

Figure 6.1 displays the evolution of the total evaluation score for the system generated report [1]. At first glance, there was a consistently improvement over time, nevertheless, this does not show the entire story. To complement, Figure 6.2 shows the evolution of the score when considering only items that evaluate the form of the report and Figure 6.3 shows the score considering only items that evaluate the content of the report. This section will break down the meaning and insights that can be extracted from these three charts.

[1]Remember the score is just the percentage of items checklist that are correct in the output.

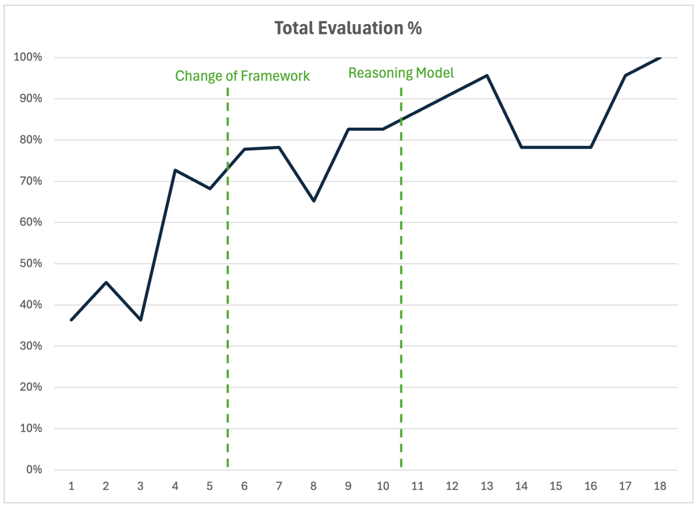

Figure 6.1: Evolution of evaluation results

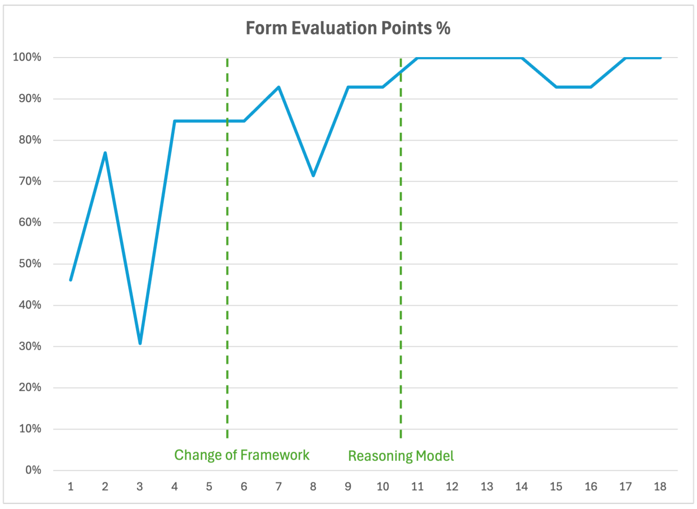

Figure 6.2: Evolution of evaluation results considering only form items

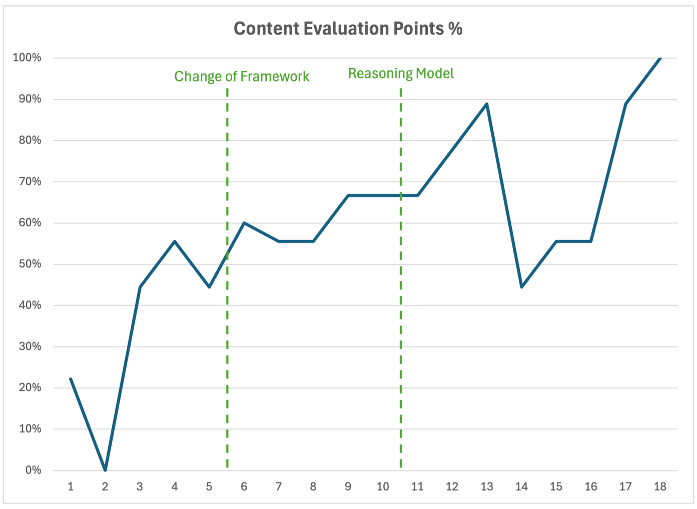

Figure 6.3: Evolution of evaluation results considering only content items

As mentioned before, the first iteration considered a single orchestration entity, with a single task to create the entire report. As can be seen in Figure 6.2, this iteration could get a reasonably high score on the form evaluation, but never perfect. It consistently failed to properly present the "current" period of data - in every case, the report analysed the entire sales history, which is not useful for a report that will be issued periodically. This is a key nuance of the task.

Moreover, as evidenced in Figure 6.3, this agentic architecture failed to score more than 60% on the content score, despite changes to the task description or architecture. In fact, as can be reviewed in Figure D.1, every iteration had issues with different parts of the content.

After an in-depth review of the intermediate steps taken by the system, the reason for this issue became clear: while iterating, the context keeps growing and growing, even if the iteration does not actually move the task forward. In this particular architecture, this growth in context also affected the context of each sub-agent, making their responses worse as the iterations progressed. This led to two insights, which drove the approach to the second iteration:

- **Key Insight 1**: while LLMs can indeed perform large tasks with some success, by breaking the task into subtasks we can manage the context at each point, improving their effectiveness and reliability.
- **Key Insight 2**: deep research tasks increase the context quickly, but not all intermediate findings actually add information, so they benefit for summarization steps within the workflow[2]. This is in line with the findings of [16].

The first point can be seen clearly on Figure 6.2. After the change to a step-by-step workflow, fluctuations in form score reduced vastly, while the content score slowly increased as the entire workflow was rebuilt. This is also highlighted by the fact that LLM steps were not reliably recommended remedy actions that were measurable; by making the recommendations part of the single deep research task, and not a separate step, the nuance is lost within the large context the task itself creates, resulting in very general recommendations.

Later on, after finishing rebuilding the workflow, the system's default model was changed to use one of OpenAI's reasoning models, leading to **Key Insight 3**: changing to a reasoning model has a smaller impact than the change to a step-by-step workflow, as the reasoning step can be approximated by the architecture of each agent within the workflow. That said, the quality of certain parts of the output, such as the graphs generated, did improve markedly by using a more powerful model.

Nevertheless, as mentioned in Chapter 4 with the change to reasoning model runs became more volatile, as reasoning models do not accept configuration variables such as Temperature, Top K or Top P. This was particularly evidenced in sub-agents that work iteratively before responding, which seemed to fall into infinite loops more easily. This led to **Key Insight 4**: autonomous agents are not guaranteed to self-recover, and the system must be planned around that. Common architectures use steps to reflect and re-plan, but this assumes that the agent will at some point finalize the task. For cases such as deep research, where there might not be clear measure for the task to be "done", it is important to create, and properly handle, forceful ending points.

Finally, as mentioned in Chapter 5 the evaluation score does not measure how useful the report is for decisions making. Some reports, for the same request and with access to the same data, might show detailed operational information by City and Product Family, while others might extract insights by Customer. All of these would get the same evaluation score, but might be more

[2]As opposed to most common architectures which only summarise at the end of the task

or less useful. This highlights **Key Insight 5**: at the current state of LLMs, they can be used to reduce the time spent in data extraction, but companies will still need a human in the loop to review the output, confirm whether additional information is needed, and decide how to act based on the extracted insights.

## 6.3 Future Work

The current system is a proof of concept. As such, it can - and should - be improved upon to create production-ready system. The most important changes to get there would be:

1. Improvements to the evaluation framework:
    - Fully automate the evaluation of outputs, to allow for quicker iterations.
    - Evaluate intermediate steps within a full execution, in addition to individually.
    - Add objective evaluation to intermediate steps when appropriate.
2. Improvements to the product readiness of the user interface for configuration:
    - If the system will be deployed in a publicly accessible location, add authentication.
    - Add acceptance tests that can be run on the CI/CD pipeline.
    - Potentially, allow different reports to run on different cron configurations.
3. Improvements to the agent architecture and outputs:
    - Make the corrective actions proposed by the LLM when there is a “special case” measurable, by creating a separate step in the workflow dedicated to providing and measuring the recommendations.

# Bibliography

[1] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[2] Danqi Chen et al. *Reading Wikipedia to Answer Open-Domain Questions*. 2017. arXiv: 1704.00051 [cs.CL]. URL: https://arxiv.org/abs/1704.00051.

[3] 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.

[4] Adam Fourney et al. *Magentic-One: A Generalist Multi-Agent System for Solving Complex Tasks*. 2024. arXiv: 2411.04468 [cs.AI]. URL: https://arxiv.org/abs/2411.04468.

[5] 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.

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

[7] 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.

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

[9] LangChain Documentation. *Multi-agent systems*. Last Accessed: 2025-08-26. 2025. URL: https://langchain-ai.github.io/langgraph/concepts/multi_agent/.

[10] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401.

[11] 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.

[12] OpenAI Help Center. *Best practices for prompt engineering with the OpenAI API*. Last Accessed: 2025-08-26. Aug. 2025. URL: https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api.

[13] Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. *AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges*. 2025. DOI: https://doi.org/10.1016/j.inffus.2025.103599. arXiv: 2505.10468 [cs.AI]. URL: https://arxiv.org/abs/2505.10468.

[14] Noah Shinn et al. *Reflexion: Language Agents with Verbal Reinforcement Learning*. 2023. arXiv: `2303.11366 [cs.AI]`. URL: `https://arxiv.org/abs/2303.11366`.

[15] 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`.

[16] Blerta Veseli et al. *Positional Biases Shift as Inputs Approach Context Window Limits*. 2025. arXiv: `2508.07479 [cs.CL]`. URL: `https://arxiv.org/abs/2508.07479`.

[17] 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`.

[18] 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`.

[19] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: `2201.11903 [cs.CL]`. URL: `https://arxiv.org/abs/2201.11903`.

[20] Shunyu Yao et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. 2023. arXiv: `2210.03629 [cs.CL]`. URL: `https://arxiv.org/abs/2210.03629`.

[21] 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

# System Manual

Appendix B

# User Manual

# Appendix C

# Report Evaluation Questions

| **Form Items** |
|---|
| Does the report include all the required sections? |
| Does the report actually address the required KPI? |
| Does the report include an analysis of the evolution of the KPI? |
| Does the report include detailed data, in addition to the high level KPI information? |
| Does the report include at least one graph? |
| Does the report include more than one graph? |
| Does the report include a projection of the KPI? |
| Is the data retrieved correct? |
| Does the report accurately present the last information period? |
| Is the content of the Executive Summary accurate to the section's description? |
| Is the content of the Overview accurate to the section's description? |
| Is the content of the Trends & Context accurate to the section's description? |
| Is the content of the In Depth Analysis accurate to the section's description? |
| Is the content of the Forward Outlook accurate to the section's description? |

Table C.1: Checklist of Form Items

| **Content Items** |
|---|
| Does the analysis in the report accurately represent the evolution of the KPI? |
| Does the report include an analysis of the detailed data? |
| Does the detailed analysis mention specific drivers of decline? |
| Are all graphs relevant to the report? |
| Are all graph types adequate for the data presented? |
| Are all projections adequate for the data presented? |
| Does the report accurately note any "special case" present? |
| Does the report provide next steps for any "special case" present? |
| Are all statements in the report sustained with data? |

Table C.2: Checklist of Content Items

# Appendix D

# Experiments Performed and Evaluation Results

| No. | Date | Notes | Default Model | Form | Content | Total |
|---|---|---|---|---|---|---|
| 1 | 02-Jul | First test with all of my MVP Agents: Magentic One (DB, Coder) + Separate Editor | gpt-4o-mini | 50% | 29% | 36% |
| 2 | 02-Jul | Re run of previous experiment | gpt-4o-mini | 75% | 29% | 45% |
| 3 | 03-Jul | Move Editor Agent inside team, improve quant agent prompt | gpt-4o-mini | 50% | 29% | 36% |
| 4 | 04-Jul | Improve Magentic One task description | gpt-4o-mini | 75% | 71% | 73% |
| 5 | 04-Jul | Improve Editor Prompt | gpt-4o-mini | 75% | 64% | 68% |
| 6 | 09-Jul | First test with LangGraph - Minimal MVP, without in-depth analysis | gpt-4o-mini | 78% | 78% | 78% |
| 7 | 11-Jul | Add step for operational info just with the quant agent | gpt-4o-mini | 100% | 64% | 78% |
| 8 | 15-Jul | Leave plot generation for last step | gpt-4o-mini | 67% | 64% | 65% |
| 9 | 15-Jul | Re run of previous experiment | gpt-4o-mini | 89% | 79% | 83% |
| 10 | 23-Jul | Add the agent for in-depth research | gpt-4o-mini | 89% | 79% | 83% |
| 11 | 24-Jul | Add the agent for in-depth research | o4-mini | 100% | 79% | 87% |
| 12 | 28-Jul | Change prompt: ‘concise’ to ‘detailed’ | o4-mini | 100% | 86% | 91% |
| 13 | 28-Jul | Re run of previous experiment | o4-mini | 100% | 93% | 96% |
| 14 | 29-Jul | Add capacity for report writing graph to load csv files | o4-mini | 100% | 64% | 78% |
| 15 | 29-Jul | Re run of previous experiment | o4-mini | 89% | 71% | 78% |
| 16 | 30-Jul | Add check for coding agent intermediate input vs final input | o4-mini | 89% | 71% | 78% |
| 17 | 30-Jul | Update prompt in quant agent to reduce intermediate requests for user input | o4-mini | 100% | 93% | 96% |
| 18 | 31-Jul | Update prompt in internal data agent to ensure all files are generated in the right folder | o4-mini | 100% | 100% | 100% |

Table D.1: Experiments performed with Evaluation scores

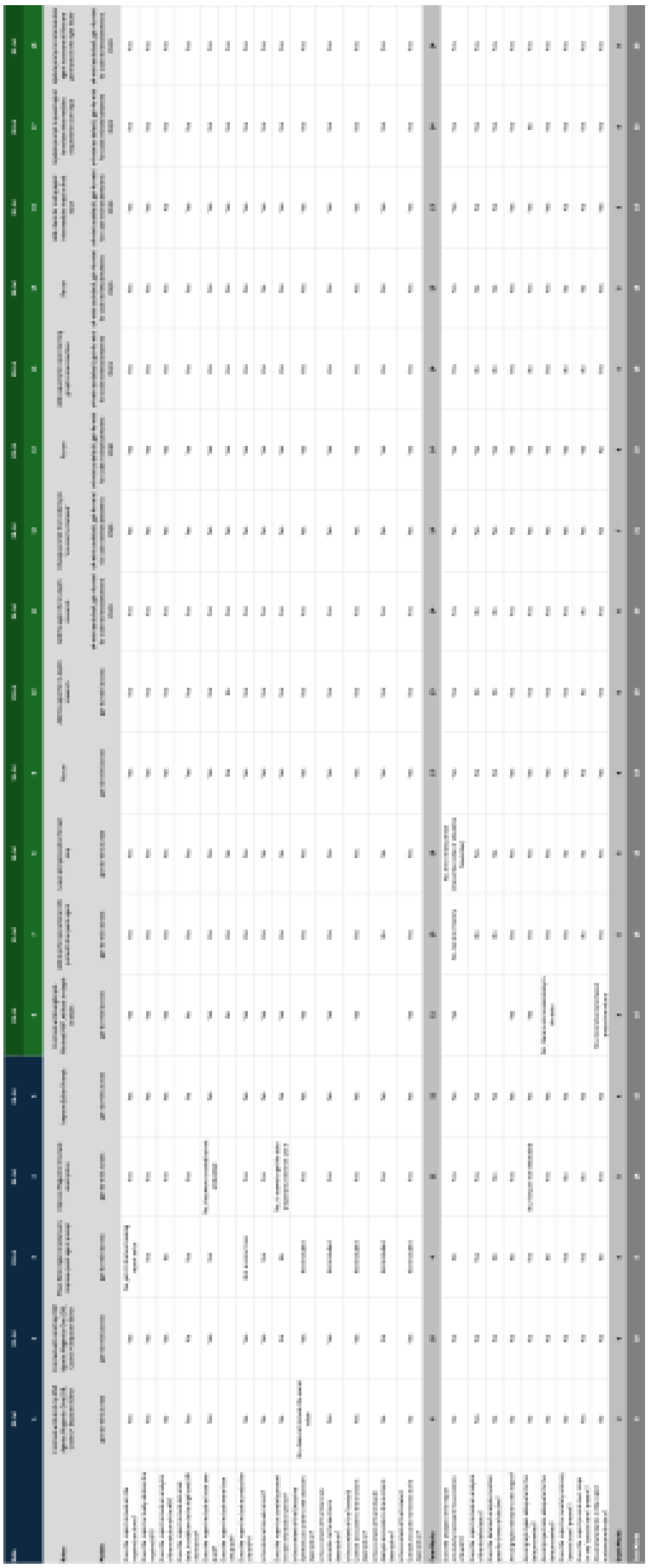
Figure D.1: Detailed evaluation scores; empty cells are items that do not apply to that specific experiment.

# Appendix E

# Generated Report - Sample