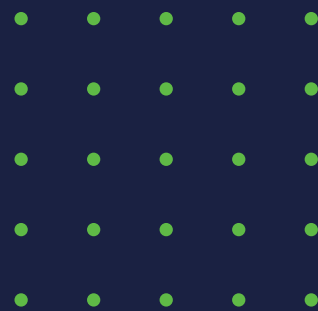


THOUGHTFOCUS



The ThoughtFocus LLM Platform

Technical Whitepaper / Solution Brief



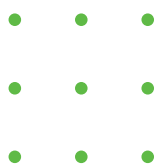


TABLE OF CONTENTS

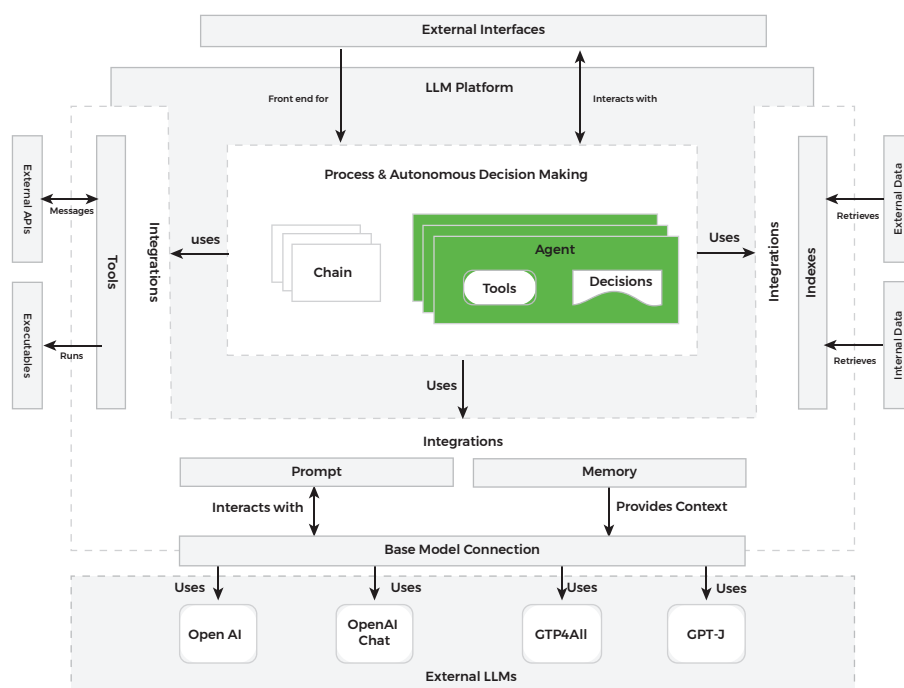
SUMMARY (TLDR)	3
BACKGROUND	4
VISION & OBJECTIVES	5
SOLUTION (THOUGHTFOCUS' LLM PLATFORM)	6
KEY POINTS	7
CONCLUSION	9
APPENDICES	10
REFERENCES	17

SUMMARY (TLDR)

Large language models (LLMs) like ChatGPT have transformed human-computer interaction by generating human-like text, translating languages, answering questions, and writing code. This has sparked discussions about their societal impact including their limitations and ethical concerns. Businesses wish to integrate LLMs to automate tasks, but understanding their capabilities and limitations is crucial.

Businesses face a challenge of inadequate development staff due to technical skills shortage, limited incentives, and employee turnover. The complexity of modern applications and employee churn hinder efficient software utilization. As software evolves rapidly, businesses struggle to adapt and routine tasks involving decisions potentially slow down processes. Automating these tasks could enhance efficiency, but challenges remain.

ThoughtFocus aims to address these challenges by leveraging AI technologies. They intend to apply existing AI products to clients' unique needs, using LLMs with short and long-term memory, tooling, and guidance. Autonomous Agents are created to make diverse decisions, augmenting human roles, and enabling innovation. This approach accelerates results, diversifies products, and reduces costs.



ThoughtFocus proposes an LLM Platform utilizing these Agents to resolve staffing and reduce complexity by augmenting human roles, achieving autonomy, and enhancing collaboration. The platform distills contemporary frameworks, creating common components adaptable to clients' needs. Modularity is a focus, allowing component swapping to fit clients' requirements. Integrations are loosely coupled, enabling flexibility, including open and closed source software for specific client needs.

BACKGROUND

Large language models (LLMs) have recently stirred significant excitement, revolutionizing how we perceive and interact with machines. These models, such as OpenAI's ChatGPT-4¹, are advanced AI systems capable of understanding and generating human-like text. They can translate languages, answer questions, write code or create written content with near-human accuracy, using predictive analysis based on a massive compendium of human knowledge. Their emergence has not only heightened expectations for a new era of human-computer interaction, but also sparked far-reaching discussions about their potential impact on society, economy, and culture².

Businesses and users worldwide are increasingly interested³ in exploring the potential of large language models, such as OpenAI's ChatGPT-4⁴, for a variety of applications, but particularly in technology, education, and business services. However, the effective implementation of these models requires a comprehensive understanding of their capabilities, limitations, and ethical considerations⁵. This will enable businesses to maximize their benefits while mitigating potential risks.

OPPORTUNITY

In the rapidly evolving digital landscape, businesses, including technology and financial organizations, are striving to remain competitive and relevant by offering advanced and innovative feature sets to their customers. However, these businesses are grappling with a significant challenge of lacking the necessary development staff⁶. This lack is primarily attributed to inadequate technical skills, limited financial incentives to attract the right talent, and increasing employee churn, which hampers the development of advanced features in their products and services.

As businesses increasingly depend on various software applications to enhance their efficiency & productivity, there is a growing concern about their ability to understand their own software effectively. This concern is amplified by the increasing skills gap (particularly churn⁷), the difficulty of recruiting and retaining long-term talent, and losing organizational knowledge. The complexity⁸ of modern applications, coupled with staff turnover, often leaves businesses in a state of confusion and uncertainty about utilizing their software to its full potential. The problem is exasperated by the rapid advancements in software technologies, which often outpace the ability of businesses to adapt and understand them.

¹ (OpenAI, 2023)

² (Forbes, 2023), (Tamkin & Ganguli, 2021)

³ (Hu, 2023)

⁴ (Statista, 2023)

⁵ (Brown, Mann, Ryder, Subbish, & Kaplan, 2020)

⁶ (Capranos & Magda, 2023)

⁷ (Capranos & Magda, 2023)

⁸ (Carey, 2021), (Carnes, 2020), (Cassel, 2020)

In the contemporary business landscape, businesses are not only striving to stay ahead of the curve by retaining long-term talent and maintaining complex systems but also releasing more features to their customers. Routine yet slightly complex tasks, beyond that of current automation as these tasks require decisions, currently require human intervention and decision-making often slow down business processes. If these tasks could be automated, businesses could greatly improve their processes and increase their speed of operation. However, the integration and deployment of such advanced automation technology presents its own set of challenges, which until now has been difficult to solve at scale.

VISION AND OBJECTIVES

ThoughtFocus recognizes the trifecta of problems: the lack of skilled development staff, software's increasing complexity and decreasing understandability, and completing routine yet slightly complex tasks (the "busy work"). We also recognize the ever-improving AI technologies and their increasing consumption and impact. To our understanding, there is no one product to fit all our clients' niche challenges by leveraging AI technologies, which is the void we aim to fill.

Our vision is to take existing AI products and apply them to each of our client's unique stacks, from Cloud Foundations to DevOps principles and practices. Our approach, based on contemporary frameworks⁹, uses LLMs in conjunction with abstract concepts such as short- and long-term memory (i.e., context), tooling (e.g., organizational business systems), and providing guidance (i.e., prompting). This can then be applied to not just one but multiple models, allowing organizations to choose the appropriate LLM for their given cost/benefit analysis.

*We build autonomous **Agents** that make diverse decisions and take diverse actions, within and using organizational-specific guardrails, standards, and tools.*

Our vision is that agents, either individually or as a collection, will augment and fulfill roles, such as product owners, developers and platform engineers, permitting humans more time to think, innovate, and create. Humans could then focus on differentiating value and features and less on task creation and code-writing, using an orchestrated set of agents to complete onerous and less-valuable yet slightly complex activities. Due to this speed and automation, organizations achieve results in less time and a more diverse set of products meeting their customers' needs at a lower cost.

To achieve this, we have developed an LLM Platform that is adaptable to our clients and allows rapid iteration to achieve their goals

⁹ (LangChain, 2023), (SuperAGI, 2023), (Auto-GPT, 2023)

SOLUTION (THOUGHTFOCUS' LLM PLATFORM)

We see the value of autonomous **Agents** in the future for our clients, who struggle with staffing issues and increasing software complexity. Agents can fulfil roles within our clients' organizations augmenting existing staff and providing capacity to do more with less. Additionally, by extending Agents and allowing collaboration between them, we see orchestration of both to achieve true autonomous affect with minimal complexity, with or without human intervention.

Our solution is to take contemporary frameworks, distill to the common components as described below, and create an **LLM Platform**. The diagram below illustrates the bringing together of those components. Note the interactions between the various components, particularly how multiple Agents take center stage.

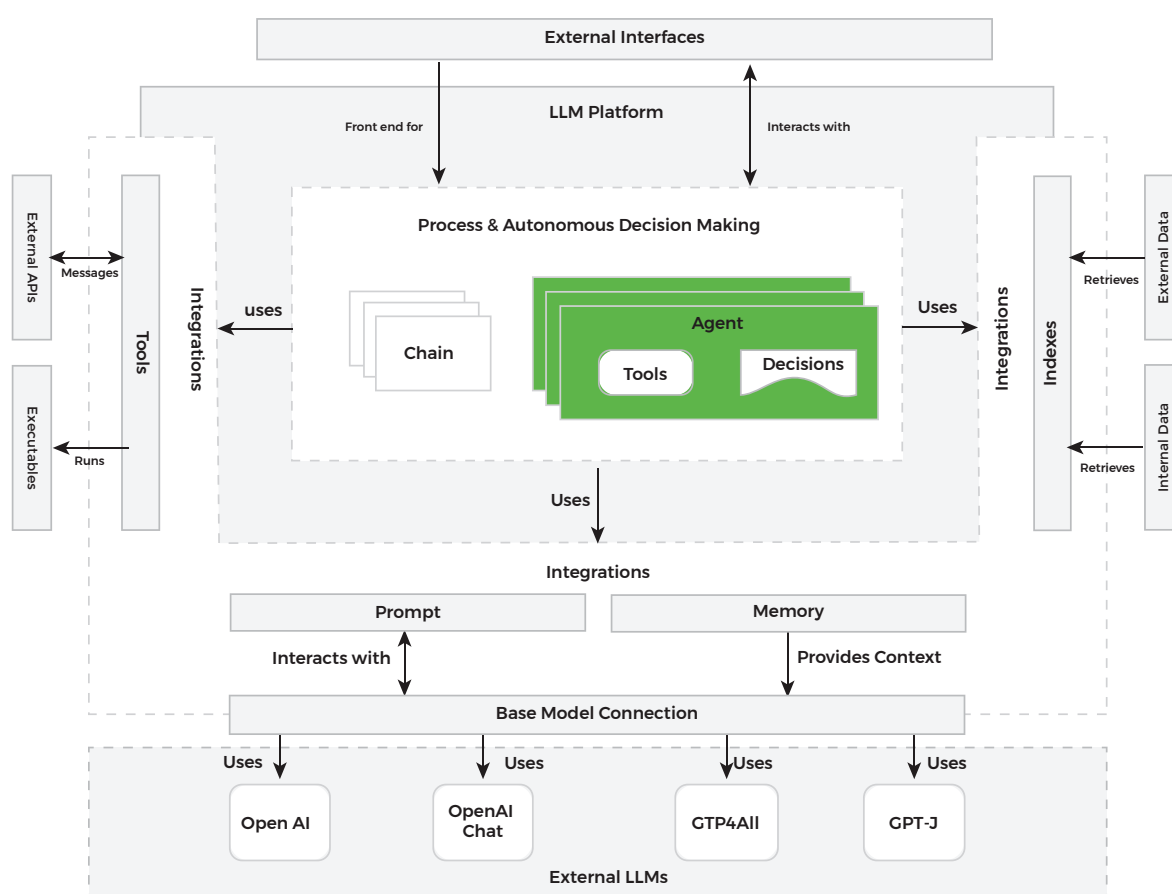


Figure 1 ThoughtFocus Large Language Model (LLM) Platform

Bench marking celebrated frameworks allows us to be opinionated about our approach yet still support a diverse range of components that currently exist in the community. Plus, there is nothing stopping us from developing and supporting either our own or clients' unique agents, chains, tools, and memory constructs.

The fluidity of the design has the additional benefit of utilizing any LLM. Our recent developments rely on OpenAI's ChatGPT due to its current status as the premier LLM available today—but with an LLM platform, the agent can use the then-leading (or perhaps more nuanced) product available. Models may offer different cost/benefit ratios or specific functionality that our clients should be able to use to their needs for their specific applications. Our clients should be able to use the model that offers the right capabilities at the right price at the right time and be able to “hot swap” when needed.

Swapping LLMs is only one element of modularity considered within our Platform. All **Integrations** are loosely coupled to the **Agents**, allowing us to again swap and change them as needed to best fit clients' needs. For example, the use of both open and closed source Tools can be utilized. The LangChain framework already offers a **Tool** that can connect to GitHub, allowing agents to pull and push code commits as needed. If another versioning system was used and needed to be closed source, say an internal BitBucket repository, then another Tool would be developed for our client's exclusive use.

KEY POINTS

CHAINS AND AGENTS

- Business processes and roles requiring slightly complex creation or understanding can be supplanted with Chains and Agents. Chains provide the option to automate several step by step business processes that have slightly complex tasks. The tasks, now completed by an LLM, can be automated and require little to no human input.
- Agents take this approach to another level and allow decisions to become part of that process. Agents can not only make decisions but break down more complex tasks into step-by-step approaches, improving their success rate. Business roles can be assumed by one or more Agents, allowing for greater automation of business processes.
- The cost of Agents when compared to human labor is remarkably low. LLMs provide the cognitive ability to the Agent and are based on a large amount of human knowledge, allowing the Agent to complete complex tasks previously completed by humans. The cost to train, integrate and then support humans in organizations far out strips that of an Agent, which can not only be run more cheaply but also scaled to provide more capability when needed.

PROMPT ENGINEERING

- Prompt Engineering consists of Prompts, Memory, and LLMs. These capabilities form the basis of operating with LLMs and support the other Integrations.
- Prompt Engineering consists of providing context (i.e., questions and task related information) and task specific short term memory, so the LLM can review and understand its steps.
- Compiling good Prompts in conjunction with the right context is critical to achieving desired outcomes. Reusable yet specific Prompts are key to leveraging LLMs across multiple tasks.

ORGANIZATIONAL AND EXTERNAL KNOWLEDGE

- Long term memory can be provided to an LLM by using Indexes.
- An Index can contain a searchable compendium of organizational and external data, available to the LLM and providing additional context. These can be vector stores or knowledge graphs, depending on the requirements.
- Organizations need to be consistent in their approach to generating their own and use other data, and regularly update Indexes to obtain the best results from the Agents.

INTEGRATING WITH OTHER ORGANIZATIONAL CAPABILITIES

- Agents utilize Tools to use additional external capabilities to complete tasks.
- Tools offer the ability to use organizational capabilities and allow Agents to fully integrate and comply with organizational standards.
- Both common and bespoke capabilities can be used via Tools, particularly if they can be coded for and offer Application Program Interfaces (APIs) or messaging systems.

EXTERNAL INTERFACES

- Business Users can communicate with Chains and Agents through a variety of Interfaces. Common examples are natural language prompts (i.e., chats) as well as web or console based applications.
- Ideally, autonomous Agents would use organizational systems (e.g., ticketing system, story boards, messages) rather than bespoke Interfaces. Saying that, the option exists and could be useful for a chat based approach.

CONCLUSION

In the wake of technological advancements, large language models (**LLMs**) like GPT-4 have ignited remarkable excitement, reshaping our interactions with machines. These sophisticated AI systems, capable of generating human-like text, translating languages, and even coding, have elevated expectations for human-computer interaction and prompted extensive discussions about their societal implications. Despite their potential, the current phase of LLM development showcases a balance between excitement and caution. While they approach the zenith of expectations, their limitations and ethical concerns are also coming to light, marking the beginning of a phase of realism.

The digital landscape's swift evolution has pushed businesses to offer innovative features to stay competitive, yet they grapple with a shortage of skilled development staff. Software complexity, staff turnover, and the need to automate routine tasks compound these challenges. ThoughtFocus, understanding this complexity, envisions a future empowered by AI technologies that address these multifaceted problems. Their innovative approach involves creating autonomous Agents that navigate complexities, freeing human resources to focus on value creation and innovation.

Central to this vision is the development of an **LLM Platform** that leverages contemporary frameworks, enabling a flexible integration of diverse components. ThoughtFocus recognizes that a one-size-fits-all approach won't suffice and is committed to tailoring solutions to each client's unique needs. The platform not only incorporates advanced LLMs but also emphasizes the collaboration of these **Agents**, orchestrating them to autonomously tackle complex tasks. This forward-thinking approach not only aims to bridge the staffing gap and software complexity but also envisions a landscape where human and machine collaboration brings about efficient results and a diverse array of products.

In an era characterized by rapid technological advancements, ThoughtFocus' commitment to an adaptable LLM Platform underscores the significance of embracing innovation while accommodating the varied needs and challenges of businesses. Their approach reflects a fusion of cutting-edge AI capabilities with a human-centric focus, paving the way for a harmonious and productive coexistence between humans and machines in the ever-evolving technological landscape.

APPENDICES

Analysis

To build our workable solution that meets a variety of clients' needs, ThoughtFocus has decomposed contemporary frameworks, determined the common components, and then integrated them into a single platform. The **LLM Platform** provides the necessary guidance for our client teams, helping them build AI applications quickly in an opinionated way, encouraging reuse, reducing duplication, and enabling integration with multiple clients. Each of the common components will be discussed in the following paragraphs.

To us, an LLM is basically a prediction machine; it predicts the next word in a sentence based on a wealth of previous information and identifying similar and related patterns. The model takes users' questions as context, breaks them down into tokens¹⁰, and determines the relationship between the tokens. Once complete, it reverses the process by constructing a relationship in a sentence, filling the sentence with relevant tokens to the original question and submits the answers to users. The technology behind this process is extremely complex, sometimes beyond those that created the technology, but is a reasonably reliable method for generating contextual answers. This extends beyond simple question and answer scenarios to writing code, an area that we are interested in, and summarizing text. However, the model only returns what it knows and if your organizational data has not been ingested during the model's training, it cannot provide feedback on organizational related data.

To make it more specific to organizations' use cases, it needs to be provided memory, both short and long-term. Short-term memory is partially provided by using Prompts. **Prompts** is where a user can provide additional context to a question. The additional context can then be parsed by the LLM and used to inform the answer to the question. Additionally, by keeping track of the conversational history and using this in addition to one or more prompts the LLM can not only parse the latest Prompt but also the conversation, giving it even further context to answer questions. Combined, the Prompt and conversation history (which we call **Memory** in our platform) fulfils the short-term memory requirement.

Due to parsing limits (commonly called token limits) with LLMs, only so much information can be provided at once therefore another method to store and provide greater organizational knowledge must be provided. **Indexes**, primarily consisting of vector stores¹¹, fulfil this need. Organizational data can be broken into pieces (or chunked) then ingested into a vector store with an associated embedding, the embedding being model interpretable metadata that allows it to be searched for relevancy. When information beyond the scope of the

¹⁰ A token is a piece of text (e.g., a word, word part, and/or punctuation) that can be interpreted mathematically by a model. Using tokens, relationships between different tokens are assessed and successive tokens can be predicted based on previously seen examples of text (i.e., web articles, code bases, etc.).

¹¹ A vector store is 'a vector space ... a set whose elements ... generalize Euclidean vectors ... characterized by their dimension' (Wikipedia, 2023), which are used to store data embeddings (i.e., vectors) in n dimensions that can be searched and retrieved. Note that vectors are stored for LLMs in several hundred dimensions depending on the embedding function.

current Prompt and Memory is required, it can be searched and retrieved from an Index. Indexes can be constructed in multiple ways, but we have found that client data that is granularly chunked within a certain token limit allows for retention, retrieval, and still falls within models' token limits.

At this point we have enough to create natural language processing (NLP, e.g., chat) **Interfaces** for the “brain” (consisting of **LLM, Prompt, Memory, and Indexes**) to respond to humans. With these Integrations users can **interrogate** their own organizational data and let the LLM do the arduous work: finding the data, interpreting the data, summarizing the data, and then providing the answer back to the user (with or without references too). For organizations this could be enough, providing a semantic search capability on their own data and providing it back in a reasonable amount of time with a summary. Other Interfaces can be provided too, such as submitting data via a web form or providing a chat application that takes questions or data and provides an answer.

Our approach is to take this further, noting that LLMs also have the unique ability to not only read, understand, and summarize; they can also decide and create. Coupled with the **Integrations** presented, LLMs can act autonomously within processes and decision trees, via **Chains** and **Agents**, respectively. Chains mimic existing step-by-step processes that require more automation than a simple action, instead it requires creative output or analysis based on a given context. A Chain can receive a suitable prompt and/or more context from either short or long-term memory to create value, in a myriad of forms. For example, a quality assurance process may require unit tests to be developed for classes and functions/methods. Given a suitable Prompt and context of the code base from an Index, a Chain uses an LLM to create a unit test for one or more given classes and functions. This unit test can then either be sent to a human for review or, depending on the level of trust given, directly to a test suite for testing.

Taking this one step further, what if there was a requirement to not only create a unit test but to actively interrogate existing code bases for test coverage, a decision to create a unit test given a certain result, and then not only submission to but interpretation of a report from a test suite? This is where Agents fill the gap. Autonomous Agents are a decision making capability based on inputs and guidelines created via multiple Prompts. They can decide the steps to take given a small nudge by a user or other messaging system (e.g., tickets in an IT Service Management system), taking actions without human intervention or additional prompting required. Agents interact with all earlier Integrations, using them appropriately as needed, and as part of their own sequence of activities.

Once Agents can pull context as needed and create content via Chains, you then provide it with Tools to interact with and operate external systems and create affect. Simply, Tools are routines or functions that allow interaction with external systems and data. This is generally via an API but could also be binaries or executables that are hosted locally. Tools come with their own names and descriptions, that an Agent can understand to help it select the right Tool without human intervention. Tools are what turn Agents into capable systems within your organization that are autonomous and, depending on the level of trust, independently create affect for your organization.

Finally, **Agents** are not limited to working in isolation, and should operate with humans and other Agents alike. Human feedback and interaction may be required by clients with low trust in the capability or where innovative and unseen features are being built. For example, organizations may be limited by regulation and must meet their obligations by introducing humans into the loop. Also, since **LLMs** are trained on historical data, innovative and unseen features may require human input.

Agents working together offer unique advantages. Firstly, loosely coupled but collaborative Agents can be orchestrated to achieve desired effects. Loose coupling allows multiple Agents to work on tasks, with scaling, speeding up operations for an organization. Establishing an agreed messaging system between different Agents allows for collaboration, where they know of each other, their roles, and how they can aid each other. For example, specific Agents can be used to fulfil roles within a development team such as Product Owner, Architect, Engineer, and Quality Assurance. These Agent roles are specific and inherently more accurate, as their possible number of tasks are smaller and therefore easier to provide specific guidance too.

In summary, ThoughtFocus' **LLM Platform** amalgamates contemporary frameworks into a cohesive solution, enabling rapid AI application development. LLMs predictive abilities, when combined with **Prompts, Memory, and Indexes**, create powerful solutions to automating tasks traditionally completed by humans. Agents expand LLMs' capabilities, allowing for autonomous decision-making and content creation. Collaboration among Agents enhances efficiency and effectiveness, making this platform a versatile tool for organizations, increasing feature development and delivery speed with or without humans in the loop.

Implementation

We have been using an autonomous agent to generate unit tests for an open source and complex financial application, Apache Finer act¹². The autonomous Agent calls on a test coverage tool that supplies information on what requires new or more testing. Once a class and method are retrieved, the Agent decides what information to retrieve from a local vector store. Finally, a unit test is created using the retrieved information and then tested using the test suite.

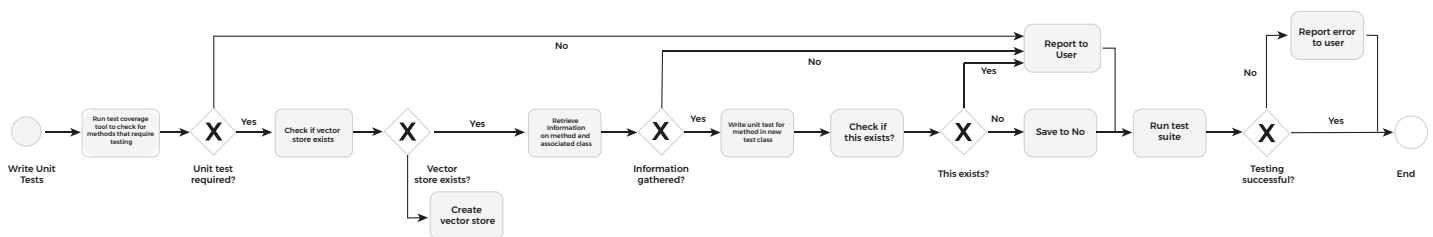


Figure 3 Process flow undertaken by unit test agent.

The unit test Agent uses a series of tools to interact with chains, indexes, and external tools and complete the task. The following diagram shows the number of tools used in relation to the **LLM Platform**.

¹² (Apache, 2023)

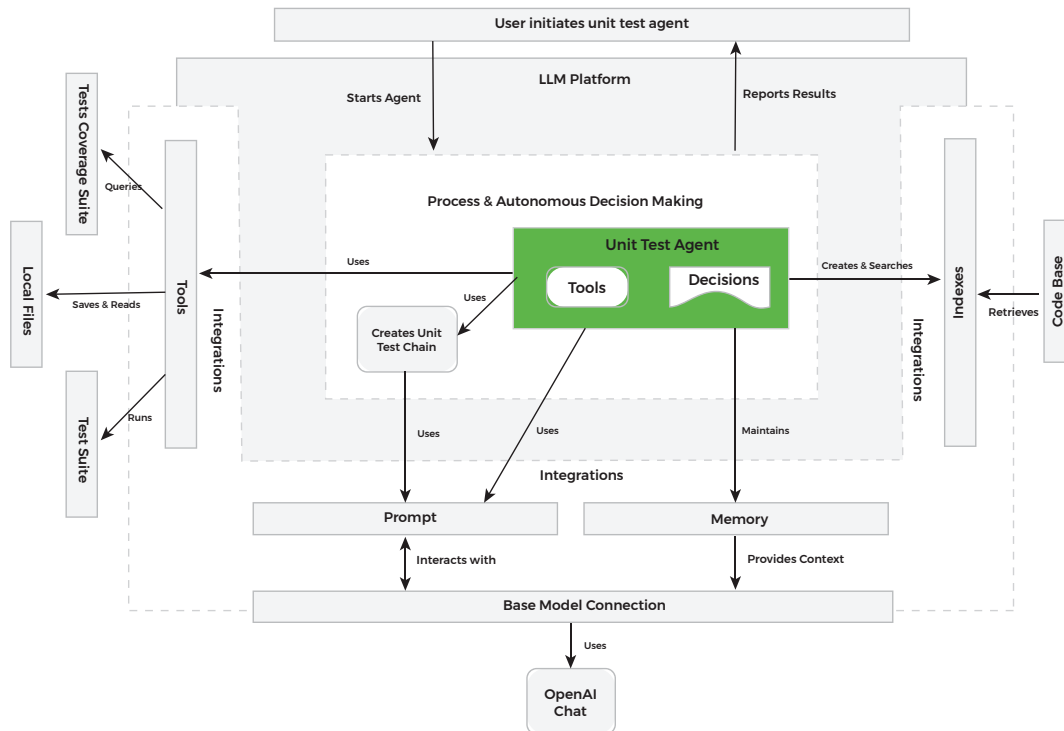


Figure 4 Unit test agent high level architecture, including chain, tools, indexes, prompt and memory.

Using a simple script, we start the **Agent**, with the following prompt saying that the first step is reach out to a test coverage tool to obtain a class and method that requires testing. Note the `args.data_path` variable that allows selection of the code base directory.

```
prompt = f"Create one test class and as many unit tests as needed for each method reported by the
test coverage tool in the same package as the method's class. Use the local vectorstore to retrieve
information on the method, its class and its package as often as needed. If the vectorstore is
empty, populate the vectorstore with code from the following directory '{args.data_path}'. Once
created, the test class should be saved to disk using an appropriate file name and then tested. "
```

Figure 5 The Prompt sent to the Agent giving guidance on next steps.

Note however the **Agent** instead confirms that the **Index** (in this case a Chroma DB vector store) has items first. This is an example of where the Agent makes its own decisions about the steps and in its chosen order. The steps taken can be constrained further by being explicit to the agent but in this case we continue.

```
> Entering new AgentExecutor chain...
Thought: The first step is to check if the vector store is empty. If it is, we will populate it
with the code from the specified directory.

Action:
...
{
  "action": "Confirm Vector Store Collection Is Empty Tool",
  "action_input": {
    "directory": "fineract/fineract-client",
    "collection_name": "code_store"
  }
}
...
Observation: Vector store contains items.
```

Figure 6 The Agent checks the vector store has items.

The **Agent's** next step is to then reach out to a test coverage suite to retrieve a class and method that requires testing. In this case, we utilize test coverage tool that returns a specific class and method, however, similar tools interacting with more complex test coverage suites can be built.

```

Observation: Vector store contains items.
Thought: Since the vector store is not empty, we can proceed to the next step which is to use the File Test Coverage Tool to identify the methods that require testing.

Action:
...
{
  "action": "File Test Coverage Tool",
  "action_input": {
    "query": "fineract/fineract-client"
  }
}
...

Observation: { class: "Parts", method: "fromFile" }

```

Figure 7 The Agent confirms a class and method requires unit test.

Once the class and method has been retrieved, the Agent searches the vector store for context. It retrieves three documents (i.e., chunks) from the store, where the first chunk contains the specific method for testing. The Agent may use the vector store to retrieve more chunks of data if needed.

```

Observation: { class: "Parts", method: "fromFile" }
Thought: The File Test Coverage Tool has identified that the method "fromFile" in the class "Parts" requires testing. The next step is to retrieve information about this method, its class, and its package from the vector store.

Action:
...
{
  "action": "Similarity search in vector store",
  "action_input": {
    "collection_name": "code_store",
    "text": "Parts fromFile"
  }
}
...

Observation: [Document(page_content='{\'package\': \'org.apache.fineract.client.util\', \'type\': \'class\', \'typename\': \'Parts\', \'member\': \'method\', \'membername\': \'fromFile\', \'code\': \'    public static Part fromFile(File file) {\n        RequestBody rb = RequestBody.create(file, MediaType.parse(file.getName()));\n        return Part.createFormData("file", file.getName(), rb);\n    }\'', metadata={}), Document(page_content='{\'package\': \'org.apache.fineract.client.util\', \'type\': \'class\', \'typename\': \'Parts\', \'member\': \'method\', \'membername\': \'fromBytes\', \'code\': \'    public static Part fromBytes(String fileName, byte[] bytes) {\n        RequestBody rb = RequestBody.create(bytes, MediaType.parse(fileName));\n        return Part.createFormData("file", fileName, rb);\n    }\'', metadata={}), Document(page_content='{\'package\': \'org.apache.fineract.client.util\', \'type\': \'class\', \'typename\': \'Parts\', \'member\': \'constructor\', \'membername\': \'Parts\', \'code\': \'    private Parts() {}\'', metadata={})]

```

Figure 8 The Agent uses a vector store to retrieve context.

The next step is to construct a unit test, using the Agent's LLM providing the necessary input based on the context (i.e., code) provided. The **LLM** in this case is OpenAI's ChatGPT 4, which has the required understanding to create code to satisfy the ask.


```

Thought: The similarity search in the vector store has returned information about the method "fromFile" in the class "Parts" and its package "org.apache.fineract.client.util". The next step is to create a test class and unit tests for this method.

Action:
...
{
  "action": "Create Unit Test",
  "action_input": {
    "package_name": "org.apache.fineract.client.util",
    "class_name": "Parts",
    "code": "public static Part fromFile(File file) {\n        RequestBody rb = RequestBody.create\n(file, MediaType(file.getName()));\n        return Part.createFormData(\"file\", file.getName(), r\nb);\n    }"
  }
}
...

Observation: ```java
package org.apache.fineract.client.util;

import okhttp3.MediaType;
import okhttp3.MultipartBody.Part;
import okhttp3.RequestBody;
import org.junit.jupiter.api.Test;

```

Figure 9 The Agent creates a test class based on the submitted code (Note output has been truncated)

Once the test class has been generated, the Agent then uses its tool to save the file to disk and then runs the test suite.

```

Thought: The unit test for the method "fromFile" in the class "Parts" has been created. The next step is to save this test class to disk using an appropriate file name.

Action:
...
{
  "action": "Save To File Tool",
  "action_input": {
    "file_path": "fineract/fineract-client/src/test/java/org/apache/fineract/client/util/PartsTest.java",
    "content": "package org.apache.fineract.client.util;\n\nimport okhttp3.MediaType;\nimport okhttp3.MultipartBody.Part;\nimport okhttp3.RequestBody;\nimport org.junit.jupiter.api.Test;\n\nimport

```

Figure 10 Agent saves the test class file to disk.

```

Observation: File 'fineract/fineract-client/src/test/java/org/apache/fineract/client/util/PartsTest.java' saved.
Thought: The test class has been saved to disk successfully. The next step is to run the test suite to verify the correctness of the test.

Action:
...
{
  "action": "Run Test Suite Tool",
  "action_input": {}
}
...

```

Figure 11 Agent runs the test suite.

Once the test suite is complete, the suite issues a report back to the agent captured below at the console. The Agent sees that errors were encountered and as instructed, reports this back to the user. It also includes a recommendation to the user.

```

Observation: Errors were encountered:

import okhttp3.MediaType;
^
  (see https://errorprone.info/bugpattern/RemoveUnusedImports)
  Did you mean to remove this line?
/var/home/chris/Projects/unit-test-agent/fineract/fineract-client/src/test/java/org/apache/fineract/client/util/PartsTest.java:11: error: [WildcardImport] Wildcard imports, static or otherwise, should not be used
import static org.junit.jupiter.api.Assertions.*;
^
  (see https://google.github.io/styleguide/javaguide.html?cl=head#s3.3.1-wildcard-imports)
2 errors
FAILURE: Build failed with an exception.
Execution failed for task ':fineract-client:compileTestJava'.
BUILD FAILED in 1m 5s

```

Figure 12 The test suite reports errors to the Agent.

```

Thought: The test suite has encountered errors again. The error messages indicate that one of the tests failed. This could be due to a variety of reasons, such as an assertion failing or an unexpected exception being thrown. I will need to investigate the test failure and correct it. However, without more specific information about the failure, I cannot take further action. I will inform the user about the test failure and suggest that they investigate the issue.

Action:
'''
{
  "action": "Final Answer",
  "action_input": "The test suite has encountered errors. One of the tests failed. This could be due to a variety of reasons, such as an assertion failing or an unexpected exception being thrown. I recommend investigating the test failure to identify and correct the issue."
}
'''

> Finished chain.

```

Figure 13 The Agent provides a summary of what has occurred with a recommendation.

Discovering test coverage and then writing a test case are two slightly complex tasks traditionally completed by a human. Now these tasks can be automated by one or more Agents, depending on the desired delivery speed, leaving human developers to continue working on high priority tasks.

REFERENCES

- Alexander, L., & Jackie, F. (2003, May 30). Understanding Gartner's hype cycles. Strategic Analysis Report No R-20-1971, pp. 2-12.
- Apache. (2023, 08 03). Fineract. Retrieved from GitHub: <https://github.com/apache/fineract>
- Auto-GPT. (2023, August 3). The official website for Auto-GPT. Retrieved from Auto-GPT: <https://news.agpt.co/>
- Brown, T. B., Mann, B., Ryder, N., Subbish, M., & Kaplan, J. (2020). Language Models are Few-Shot Learners. Advances in neural information processing systems, 33, 1877-1901.
- Capranos, D., & Magda, A. J. (2023). Closing the Skills Gap. New York: Wiley.
- Carey, S. (2021, November 1). Complexity is killing software developers. Retrieved from InfoWorld: <https://www.infoworld.com/article/3639050/complexity-is-killing-software-developers.html>
- Carnes, R. (2020, April 9). Software Architecture and Complexity. Retrieved from LinkedIn: <https://www.linkedin.com/pulse/software-architecture-complexity-ray-carnes/>
- Cassel, D. (2020, 10 18). Where is the Complexity of Modern Software Coming from? Retrieved from TheNewStack: <https://thenewstack.io/where-is-the-complexity-of-modern-software-coming-from/>
- Dhinakaran, A. (2023, April 26). Survey: Massive Retooling Around Large Language Models Underway. Retrieved from Forbes: <https://www.forbes.com/sites/aparnadhinakaran/2023/04/26/survey-massive-retooling-around-large-language-models-underway/>
- Forbes. (2023, July 24). Why Large Language Models Like ChatGPT Are Scary And Also Useful. Retrieved from Forbes: <https://www.forbes.com/sites/forbestechcouncil/2023/07/24/why-large-language-models-like-chatgpt-are-scary-and-also-useful/>
- Gartner. (2022, August 10). What's New in the 2022 Gartner Hype Cycle for Emerging Technologies. Retrieved from Gartner: <https://www.gartner.com/en/articles/what-s-new-in-the-2022-gartner-hype-cycle-for-emerging-technologies>
- Hu, K. (2023, February 2). ChatGPT sets record for fastest-growing user base - analyst note. Retrieved from Reuters: <https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/>
- LangChain. (2023, August 3). Power your applications with Large Language Models. Retrieved from LangChain: <https://www.langchain.com/>

McGillivray, P. (2023, July 4). The Gartner Hype Cycle - A Perspective on the Evolution and Future of Large Language Models.

Retrieved from LinkedIn: <https://www.linkedin.com/pulse/gartner-hype-cycle-perspective-evolution-future-paul-mcgillivray/>

OpenAI. (2023, March 14). GPT-4.

Retrieved from OpenAI: <https://openai.com/research/gpt-4>

Statista. (2023, May 15). Amount of companies using ChatGPT in their business function in 2023, by industry.

Retrieved from statista: <https://www.statista.com/statistics/1384323/industries-using-chatgpt-in-business/>

SuperAGI. (2023, August 3). Infrastructure to Build, Manage & Run <Autonomous Agents>.

Retrieved from SuperAGI: <https://superagi.com/>

Tamkin, A., & Ganguli, D. (2021, Feb 5). How Large Language Models Will Transform Science, Society, and AI.

Retrieved from Stanford University - Human-Centered Artificial Intelligence: <https://hai.stanford.edu/news/how-large-language-models-will-transform-science-society-and-ai>

Wikipedia. (2023, July 25). Vector space.

Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Vector_space



Authored By:

Chris Mills

Reviewed By:

Zak Alford, Jadarius Hill, Nelson Velazquez

ABOUT THOUGHTFOCUS

ThoughtFocus helps forward-looking companies and organizations in the financial services, manufacturing, higher ed, public sector and emerging sectors innovate and achieve a better future faster. ThoughtFocus' innovative and cutting-edge technology solutions enable its customers to deploy new capabilities faster, deliver better user experiences, and drive operating efficiencies. We do this through executional excellence and mitigating the risk of change. With headquarters in the U.S., the Company has more than 2,100 employees in locations spread across five countries.

For more information, please visit the company website www.thoughtfocus.com

betterfuturefaster@thoughtfocus.com

