Master of Technology in AI Systems Project

INTELLIGENCE REASONING SYSTEM PRACTICE MODULE

Project Aristotle: The Agentic Developer Assistant

REPORT

GROUP NUMBER: 13

GROUP MEMBERS: (Alphabetically Ordered)

NORBERT OLIVER - A0328685M

SHARVESH SUBHASH - A0327428Y

1. Executive Abstract

Software developers and data scientists lose significant time manually searching documentation, forum posts, and examples to answer code-specific questions; especially as libraries evolve. General-purpose LLM assistants help but frequently hallucinate, lag behind library updates, and cannot reliably access local/private codebases. Aristotle addresses this gap with an agentic system that couples vector retrieval (for documentation) and knowledge graphs (KG) plus a temporally aware framework Graphiti (for code-level structural understanding), exposed through a VS Code extension. The system supports real time instant ingestion of public Git repositories and local projects, and live updates when files change. A knowledge-based validator reduces invalid queries before execution. The evaluation pipeline compares the responses of Aristotle agentic system with the responses of the same base model directly the using DeepCode Bench (code-understanding QA) as the reference, LLaMA-3.1 as an automated judge to compare the responses using RAGAS (faithfulness/relevance) library for the in-built evaluation class metrics like “factual correctness”, “faithfulness”, “LLM context Recall” and “Semantic Similarity”, mean of these give us the RAGAS score, complemented by latency(ms) and ingestion-time(ms) measurements. The observed outcomes are more accurate, grounded answers to code questions, lower hallucination rates, and improved developer productivity.

2. Project Background & Market Context

2.1 Problem Statement

The current software development industry along with data science, and artificial intelligence is defined by rapid proliferation of new libraries and frameworks. This makes it hard for developers when writing code to swiftly and accurately gain specific information related to those new public and internal codebases. Hence, an alternative is needed to replace the process of manual exploration which involves sifting through official documentation, reading through online forums, and analyzing code snippets examples for hours on end. Such an alternative would massively improve productivity and save valuable work time from frequent context-switching and disturbance of developer flow state.

While popular LLMs and AI-powered coding assistants for coding such as ChatGPT, Gemini, Claude, Cursor, and Github Copilot have become popular tools to try dealing with those challenges, they have many significant drawbacks. Some of the challenges that developers would commonly report while using such assistants are:

- They have a cut off date for their training data, which means that they are unaware of newer codebases or recent updates to existing ones after the cutoff.
- When faced with knowledge gaps, some of these models rely on searching the internet to get small chunks of information from online forums, articles, and possibly related documentation pages. Unfortunately, this technique would often cause the models to hallucinate and give

false information since some projects don't have a documentation website and it can only depend on limited code snippets.

- They are unable to retrieve any information from locally saved codebases outside of the code assistant's known workspaces.
- They may not immediately adapt to code files that have been changed, and instead rely on outdated or cached knowledge for those files.

2.1.1 Business Problem Background

- Pain today: developers repeatedly context-switch between editors, docs, and forums to recall function signatures, configuration options, and API nuances-especially for new or frequently updated Python libraries.
- Limits of current assistants: popular LLM tools can be unaware of the latest library updates, may depend on noisy web snippets, struggle to retrieve from private/local code, and can serve cached/outdated knowledge.
- What's needed: a developer assistant that fuses documentation and source-level understanding, works with local and public codebases on demand, updates knowledge in real time(aware of temporal evolution of codebases), and reduces hallucinations through explicit knowledge structures and validation.

2.2 Solution

Our project proposes an agentic system with RAG capability, making it knowledgeable about the target codebases. Not only is it capable of gathering information from the code documentation, but it is also capable of retrieving information from the source code itself and live-updating its own knowledge for the current codebase that the user is working on. Our system is also able to ingest from any public git repository or locally saved project.

2.3 Market Overview

The most widely used AI systems in the coding assistant market are tools that are able to help with code completion or generation. These tools are trained on many existing codebases to quickly generate and suggest code. On the other side, there are also some tools that are designed for deep and individual codebase understanding such as Sourcegraph Cody mainly used to lookup code, find bugs, and refactor code.

3. Literature Review & Market Research

All the references for this section are attached at the end of the document in the appendices section.

3.1 Academic Research

- LLM over KGs with graph embeddings (Graph-aware retrieval)

  Early LM-KG fusion for multi-hop QA introduced graph reasoning modules alongside pretrained LMs. E.g., MHGRN embedded KG subgraphs and performed multi-relational reasoning (EMNLP 2020), followed by QA-GNN which jointly selected KG nodes and propagated signals with a GNN, and GreaseLM which deeply fused LM and GNN representations across layers (ICLR 2022). More recently, GraphRAG popularized graph-indexed retrieval with community summaries to support question answering over large, private corpora.

- GNN predicts; LLM validates/explains

  A complementary line uses GNNs for prediction (e.g., link classification) and LLMs for human-readable explanations or validation. LLM-GCE frames LLM-guided counterfactual explanations for GNN decisions on molecular graphs (2024). Natural-Language Counterfactual Explanations for Graphs (AISTATS 2025) generalizes textual explanations across datasets. GraphNarrator (ACL 2025) generates textual rationales for GNNs from saliency-derived pseudo-labels, establishing an explicit "GNN predicts-LLM explains" pipeline.

- LLM as query planner (Text to SPARQL/Cypher)

  LLMs have also been used as planners that translate natural language into executable KG queries. Auto-KGQA (ESWC 2024) few-shot prompts LLMs to produce multiple SPARQL candidates and selects the best via execution feedback. For property-graph databases, Text2Cypher (2024–2025) introduced dataset resources and fine tuning recipes for NL to Cypher, along with LLM-supervised data generation pipelines to improve execution accuracy.

- Survey papers

  Several surveys map the design space of LLMs with graphs and KGs, providing taxonomies for roles (enhancer/predictor/alignment) and tasks (query processing, inference, applications). These include Graph Meets LLM (IJCAI 2024) and broader reviews of LLMs for Graphs (2024).

3.2 Market Research

The users who mainly adopt AI-powered coding assistants are professional developers and people who want to get into fields that involve coding. According to the Stack Overflow survey, developers mainly use these coding assistants to help them write code faster and debug their problems. This

would explain why the market is mostly filled with coding assistants that focus on generating or suggesting code. A good example of this and the most used tool that specifically focuses on generation and suggestion of code is GitHub Copilot. A large chunk of beginners and developers also use them to learn about new concepts and coding techniques. The key players that fill this need are LLMs like ChatGPT and Gemini because they are also designed to explain coding-related topics well for everyone.

A general trend seems to be that they are competing with each other by offering advantages that others don't have. For example, Claude offers the MCP architecture which allows Claude to make decisions and call APIs provided by developers. Products such as VSCode and Cursor are also integrated to code editors, making them more accessible and reducing context-switching between editor and chatbot.

A very niche market however slowly increases because of more users demanding tools that involve querying and searching codebases. This growth is partially driven by the need for accurate information related to large codebases, which tools like Cody aim to provide. Yet, these tools often encounter the same problem with LLMs when users need to use newly released or updated versions of public libraries. Even systems that rely on scheduled re-indexing such as Gemini Code Assist have a 24-hour delay and are unable to ingest a public codebase on the fly. These tools also raise data security concerns due to their cloud-based nature and limited self-hosting options.

4. Project Scope

This project's scope is to fill a small market gap that requires an AI-powered tool that offers balance between retrieving information from source code and documentation for more accurate data, capability of immediately gathering information from public and private Python codebase, integration with code editor, and real-time / automatic update of knowledge.

The UI for this project is in the form of a VSCode extension with an interactive LLM-powered chatbot in the hope of providing a familiar, user-friendly, and accessible tool for programmers. This is consistent with the market for developers since there is a high chance that they have used VSCode before and any chatbot system that is connected to an LLM.

The intelligent reasoning of our system focuses on:

- Cognitive techniques, where we build knowledge graphs and use it to gather more accurate data and answer. Our system also provides a chat interaction based UI so it can be easily used.
- Knowledge discovery & data mining techniques, where our system employs Graphiti for loading new connections that are not in the original KG. Graphiti also applies the technique of Breadth First Search algorithm for query based searching of the KG for relevant retrieval.

- Decision automation, where our system uses knowledge-based reasoning techniques to reduce LLM hallucinations on queries, set the logical best query configuration, and filter query results.
- Business resource optimization, where we shall experiment with using genetic algorithms to try and fine tune the model we use.

Write tools and models that we will potentially use are:

- Graphiti, a framework used to manage the KG which can help with querying, inserting, and updating the KG in real-time.
- Neo4j, a graph database that our KG manager will store their data in.
- FAISS, a vector database that has been proven to have a high-speed search.
- FastAPI or Flask, frameworks that will be used to build our back-end HTTP server.
- Qwen3:8B LLM model will be augmented with Aristotle's agentic framework. And Llama3.1:8B will be the “LLM as a judge” based evaluator for Aristotle framework’s evaluation.
- GitPython, a library we use to interact with Git repositories so that our system can load such a codebase.

5. Data Collection and Preparation

We will be using two main dataset collections for this project.

- The DeepCode Bench:

  This is the dataset bench that will be used as a referable evaluation dataset for the LLM as Judge framework to evaluate the Aristotle Agentic system framework. It has the validated Question and Answer pair for different python libraries, with an objective to test on deep code understanding in LLMs or agentic systems.

- Source code and code documentation from open-source libraries

  The code documentations (markdown files and RTD html files) of the different libraries of Python will be scraped. The documentations will be chunked and converted into vector embeddings, then stored in an appropriate, fast retrieval friendly VectorDB. Meanwhile, the source code files shall be parsed into AST and converted into triplets to be inserted into the Neo4j KG cloud instance.

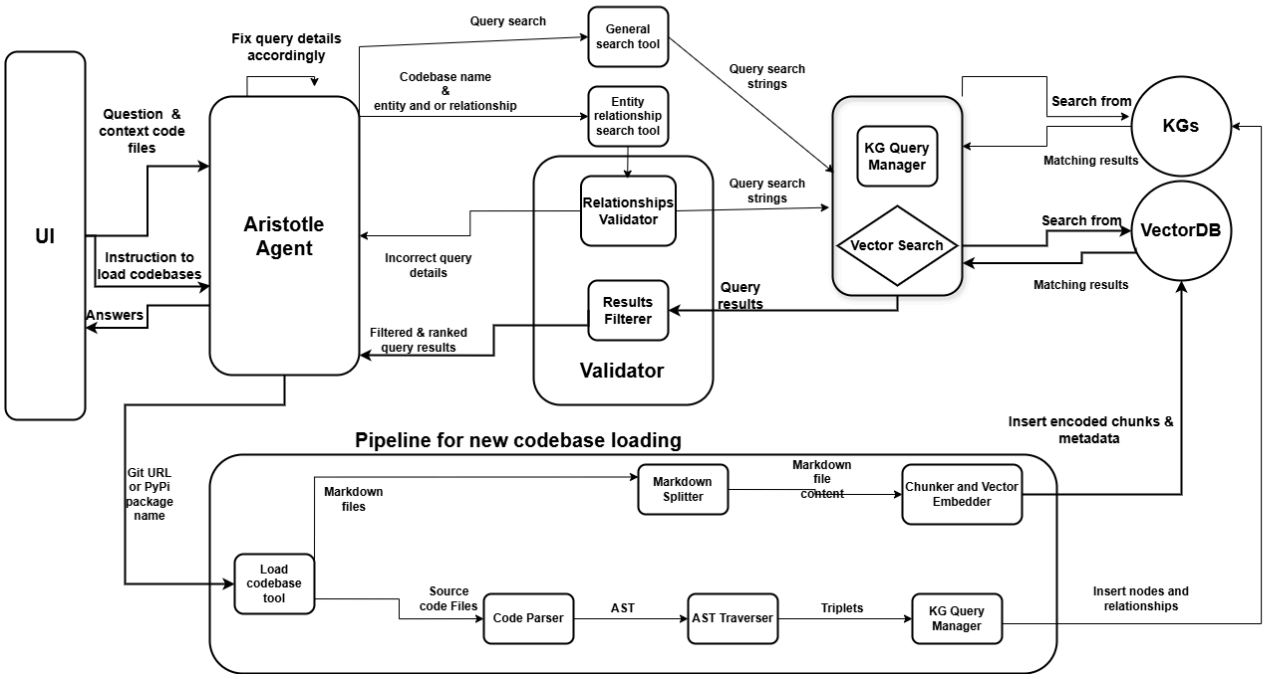
The integration of two different storages effectively for information retrieval is an important challenging task, as it involves vectors on one hand, along with graphs and their embeddings on the other.

6. System Design

6.1 Overall System Architecture

Aristotle is a system that primarily consists of two parts, which are the UI and the back-end. The primary UI is in the form of a chat-based system where the user directly converses with an LLM. On the other hand, the back-end will retrieve information, perform reasoning, fulfill requests to load codebases, and generate answers to be given to the UI.

The following image shows the overall system architecture and individual components of the system:



Here is the flow of the overall system architecture:

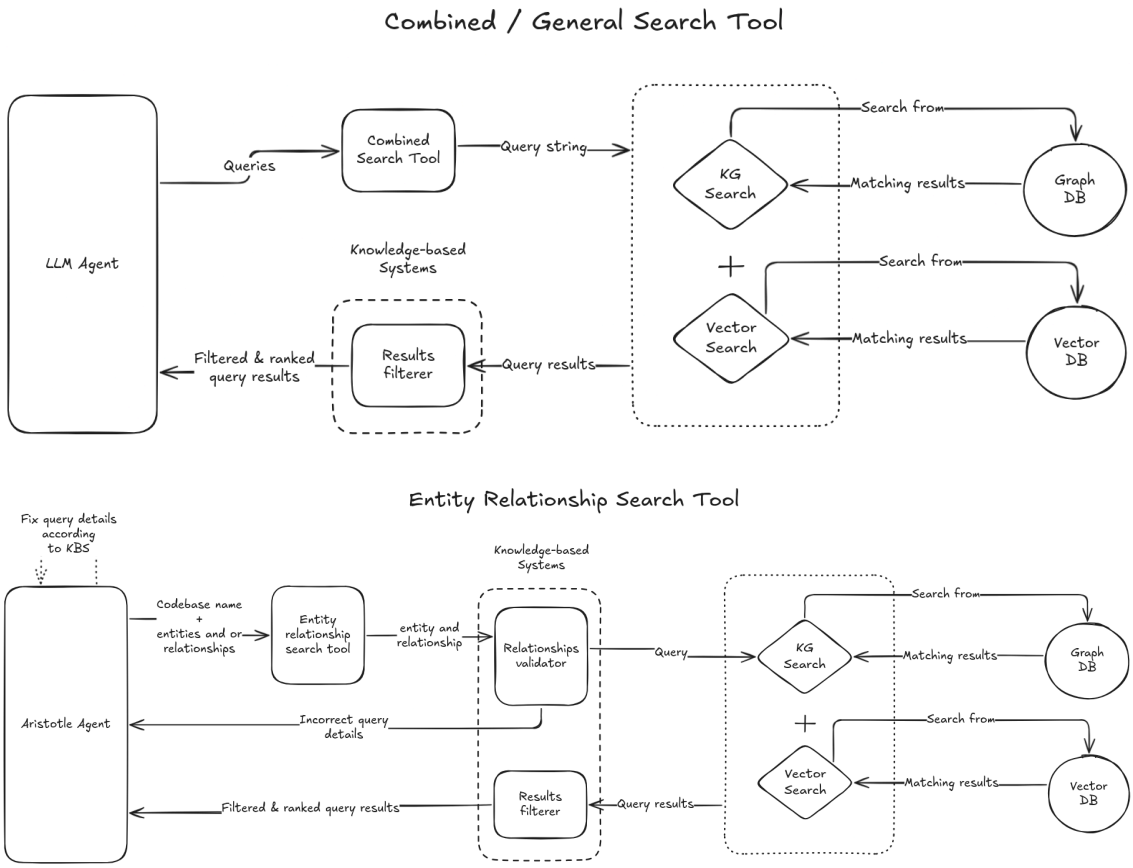
a) After the user inputs their prompt to the chat-based UI, the UI will make an HTTP request to the HTTP back-end handler. The back-end handler will proceed to give the prompt to the LLM agent, which will try to understand the objective of the prompt.

   i) If the prompt is asking a question about a previously loaded codebase, then pipeline 1 will be activated.

ii) If the prompt's goal is to load a codebase or involves a codebase that has not been loaded yet, then pipeline 2 will be activated.

b) Whenever one of the source code files in the current codebase that they are working on in VSCode changes, then the system will automatically feed that file to the code parser to eventually update the KG.

6.2 Pipeline 1: Searching for information related to loaded codebases



Here is the overall flow of pipeline 1:

a) First, the agent will make a general query using the combined / general search tool.

b) The combined search tool will search from both the knowledge graph and vector database to find semantically similar information.

c) Both results will be filtered and combined by the results filterer, then given to the agent.

d) If the agent is not satisfied with the collected information, or the user asks a question about a specific object, then the agent can use the entity relationship search tool.

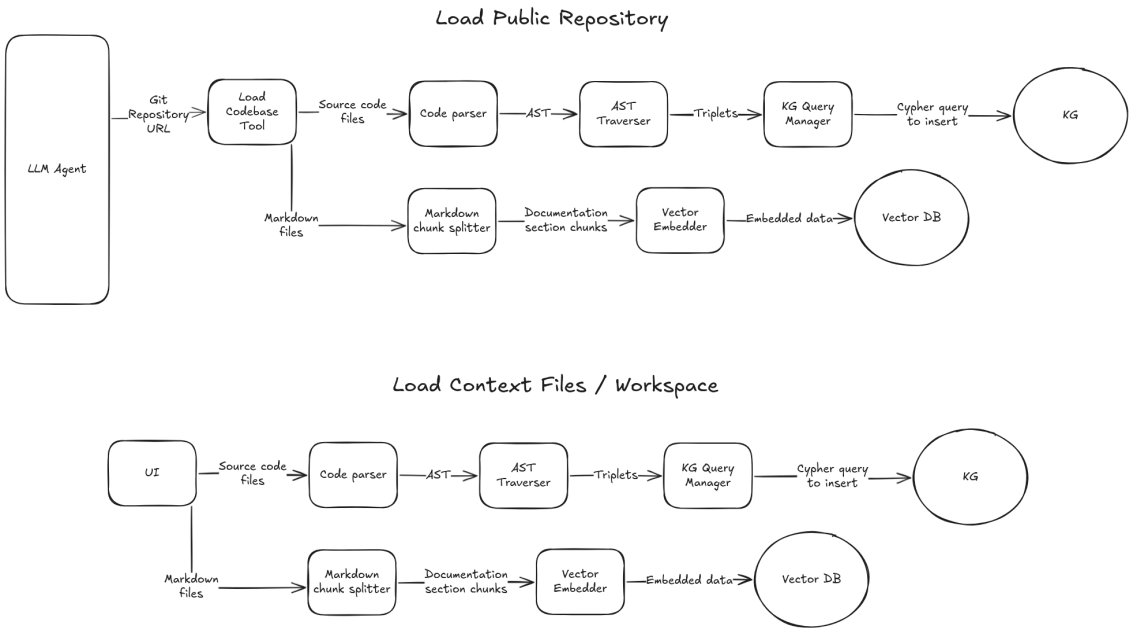
e) When using the entity relationship search tool, the agent will extract the names of the codebase, entities, and relationships from the user prompt.

f) The KBS relationship validator will use knowledge about Python programming to detect value hallucinations by the LLM in the queries. For example, the query for a “FUNCTION” source node with relationship “HAS_METHOD” is invalid since a function cannot have a method.

g) If the queries relationships are invalid, the relationships validator will give a list of incorrect queries and its details back to the LLM so that it has a chance to fix them.

h) If the queries relationships are valid, then a query will be made to the knowledge graph.

6.3 Pipeline 2: Loading codebases and context files



Here is the overall flow of pipeline 2:

a) The agent will extract the git repository URL or python package name that was given by the user, then the system will clone the appropriate Git URL to obtain all the source code and markdown files.

b) Every time the user sends a message, they may also include reference files which will be included in the agent context.

c) The user may load their current workspace, which will send all source code and markdown files to be loaded accordingly by the agent.

d) After obtaining the source files through whichever method, it will be given to the code parser, which produces the AST (abstract syntax tree) as a representation of the code.

e) The AST traverser will go through the AST to find designated “signatures” and form them into triplets.

f) The KG query manager inserts these triplets along with their attributes which includes temporal metadata into the KG database.

7. Implementation

Here are all the entity nodes types that are inserted to the knowledge graph for future searching:

Entity Node Types	Attributes	Description
MODULE	● "Name": the name of the Python module file ● "Reference": link to the containing repository / file path ● "Docstring": documentation string of the module	Represents a module / Python file.
CLASS	● "Name": the name of the class ● "Reference": link to the containing repository / file path ● "Docstring": documentation string of the class ● "Methods": list of methods in the class ● "Fields": list of class fields/attributes	Represents a class definition
METHOD	● “Name”: name of the Python module ● “Reference”: link to the containing repository / file path ● "Docstring": documentation for the method	Represents a method definition attached to a class
FIELD	● "Name": the name of the field/attribute ● "Reference": link to the containing repository / file path ● "Type": the data type of the field ● "Docstring": documentation for the field	Represents a field attached to class instances / objects
FUNCTION	● "Name": the name of the function ● "Reference": link to the containing repository / file path ● "Docstring": documentation string of the function	Represents a function definition

	• "Parameters": list of function parameters • "Return_Type": the return type annotation	
GLOBAL_VARIABLE	• "Name": the name of the global variable • "Reference": link to the containing repository / file path • "Type": the data type of the variable • "Value": the assigned value • "Docstring": documentation for the variable	Represents a global variable defined at module level

Here are all the relationships types between entities that are inserted to the knowledge graph for future searching:

Relationship Types	Attributes	Description
CONTAINS	• "Source": the containing entity (MODULE or CLASS) • "Target": the contained entity (CLASS, FUNCTION, or GLOBAL_VARIABLE) • "Reference": link to the associated repository / file path	Represents a module containing top level global variables, functions, and classes.
HAS_METHOD	• "Source": the CLASS node • "Target": the FUNCTION node • "Reference": link to the associated repository / file path	Represents a class having a method.
HAS_PARAMETER	• "Source": the FUNCTION node • "Target": parameter details • "Parameter_Name": name of the parameter • "Parameter_Type": type annotation • "Default_Value": default value if any • "Reference": link to the associated repository / file path	Represents a function or method having parameters / accepting arguments
HAS_FIELD	• "Source": the CLASS node • "Target": the FIELD node • "Reference": link to the associated repository / file path	Represents a class having a field for their instances / objects
INHERITS	• "Source": the child CLASS node • "Target": the parent CLASS node • "Reference": link to the associated repository /	Represents a class extending other classes

	file path	

Storage & Retrieval:

- Vector store (docs): chunked documentation → embeddings → FAISS for fast semantic lookup using cosine similarity between vectors of text chunks.
- KG (code): AST-derived triplets stored in Neo4j; Graphiti for real time nodes and relationships linking when new codebases are loaded, also includes an inbuilt BFS algorithm based graph traversal and searching for retrieval of relevant nodes and relationships from Neo4j based KG, for a particular query generated by an agent.
- KBS (validator): knowledge-based rules to reject/repair invalid relations (e.g., a Function cannot HAS_METHOD).

Pipelines:

- Pipeline 1 : Q&A with LLM over previously loaded codebases: KBS validation → vector search + Graphiti based graph search → result filtering → output.
- Pipeline 2 : Q&A with LLM over current workspace: direct KG querying with Graphiti for freshness on file changes.
- Pipeline 3 : Load a codebase: Git/package/local path → AST parser & traverser → triplets → Neo4j (+ temporal metadata).

LLM & UI:

- LLM orchestration with Qwen3:8B for agentic interaction and Llama3.1 for evaluation of the agentic system
- VS Code extension chat UI (@aristotle) invoking backend endpoints.

Output of Aristotle:

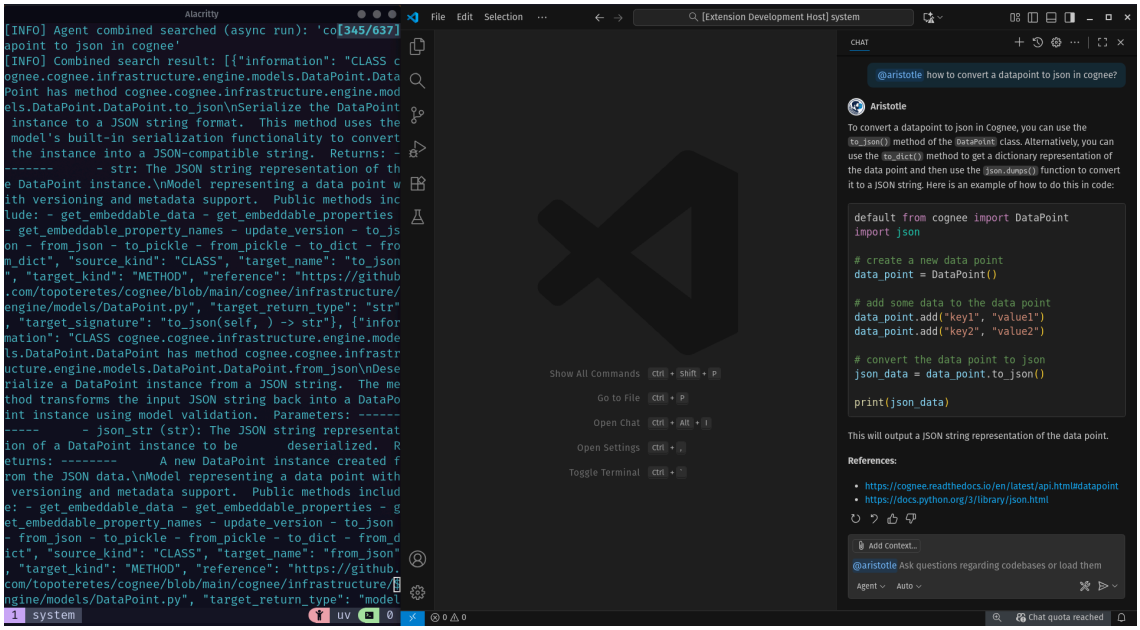
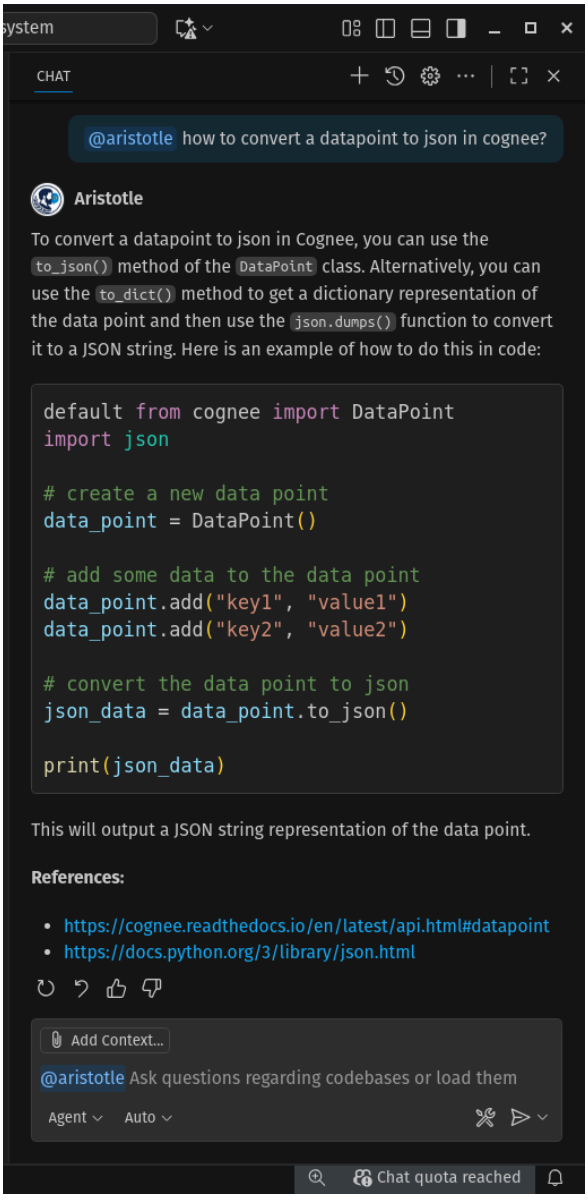


Image description:

- On the left: example combined search results obtained by the Aristotle agent according to the user's question which incorporates a hybrid of knowledge graph search and vector search result.
- On the right: example of the UI (in the form of VSCode extension) of Aristotle in action, where the user asks Aristotle a technical question about a codebase. Aristotle responded with accurate answer backed information and references, while also providing an example code snippet.

Image of zoom in on the UI Chat component:

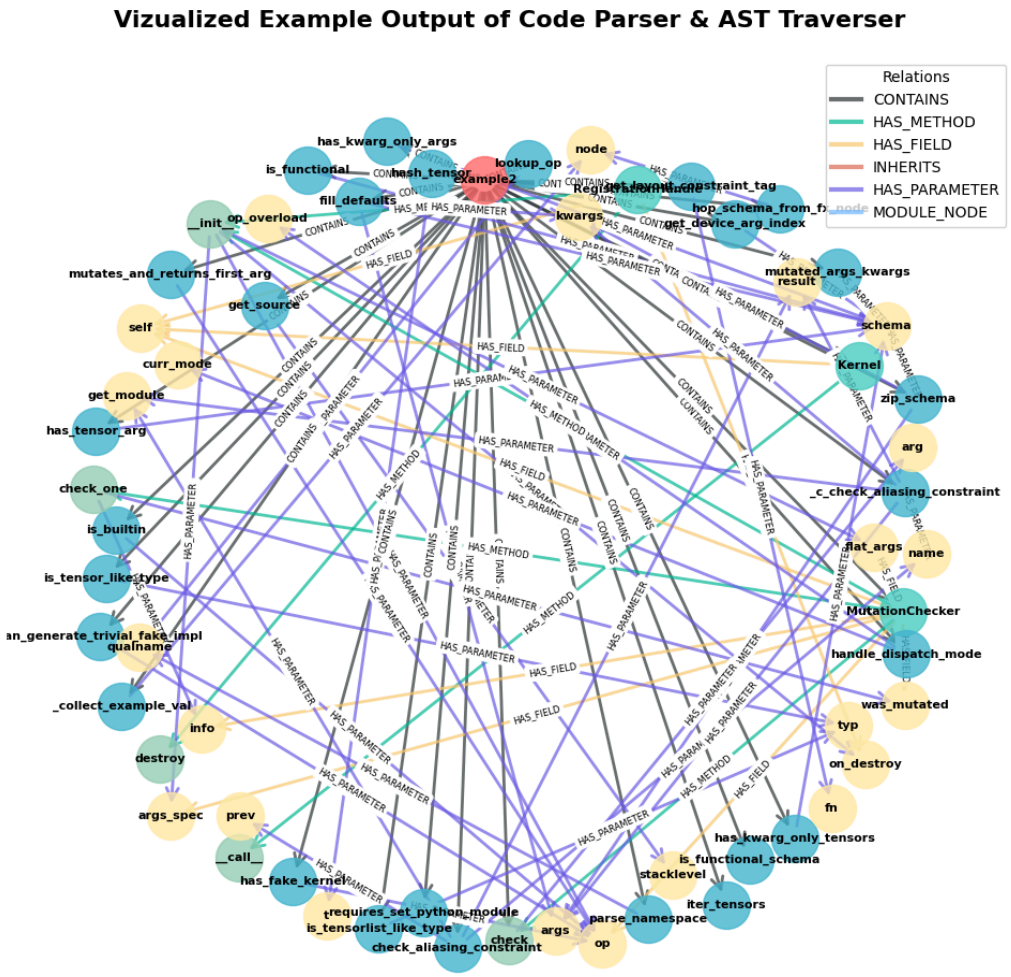


8. Timeline Plan that was followed

8.1 Proof of Concepts

Before the proposal, we had created some small proof of concept programs to see which libraries we can use and the possibility of integrating an LLM agentic system with RAG+KG. We had also successfully built a proof of concept that we are able to build the code parser and AST traverser for Python files. This has enabled us to instantly and accurately ingest any Python source code into KG triplets faster than any tool that uses an LLM to do the same.

The following image is a visualized example output from our proof of concept for the code parser and AST traverser that was randomly given a source code file from the PyTorch library codebase:



8.2 Timetable

	Tasks	Starting Date	End Date
1 - Development			
1.1	Implement codebase loading system (Git & local project ingestion)	Sep 15	Sep 17
1.2	Implement LLM agent for RAG that are able to take questions and use tools	Sep 18	Sep 19
1.3	Implement Pipeline 1	Sep 20	Sep 24

1.4	Implement Pipeline 2	Sep 25	Sep 28
1.5	Implement chat-based UI as VS Code extension	Sep 29	Oct 1
1.6	Implement Knowledge-Based System for query adjustments	Oct 2	Oct 3
1.7	Experimenting with Graphiti	Oct 4	Oct 5
1.8	Integrate LLM code generator for code snippets	Oct 6	Oct 7
1.9	Implement feature to auto-update Neo4j when codebase updates occur	Oct 8	Oct 9
2 - Testing & Documentation			
2.1	Test the overall system and fix severe bugs	Oct 10	Oct 11
2.2	Test the system for edge cases and fix them	Oct 12	Oct 13
2.3	Write the system documentation (installation and user guide)	Oct 14	Oct 15
3 - Final Deliverables			
3.1	Create demo video to promote the product	Oct 16	Oct 17
3.2	Create video to describe the system on a technical level	Oct 18	Oct 19
3.3	Write the full project report and appendices	Oct 20	Oct 23
3.4	Write individual project reports	Oct 24	Oct 25

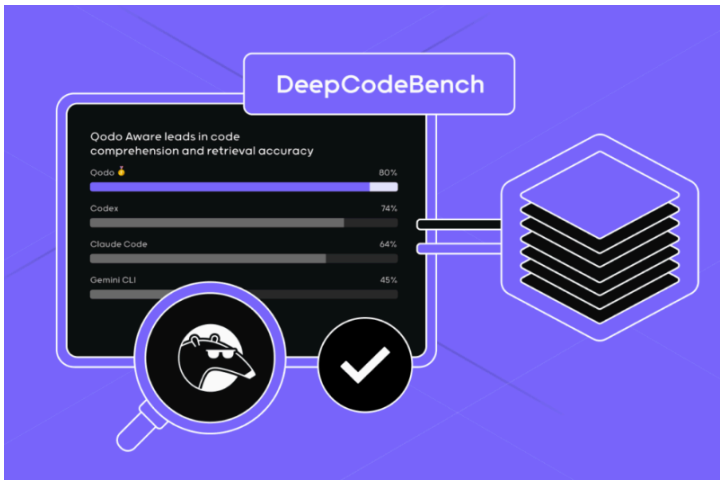
As of October 23rd, as per the timeline plan, we have properly followed and implemented the complete framework. We are happy to conclude that the complete VScode extension is functional effectively after deployment.

9. Results, Performance and Validation

9.1 Evaluation process:

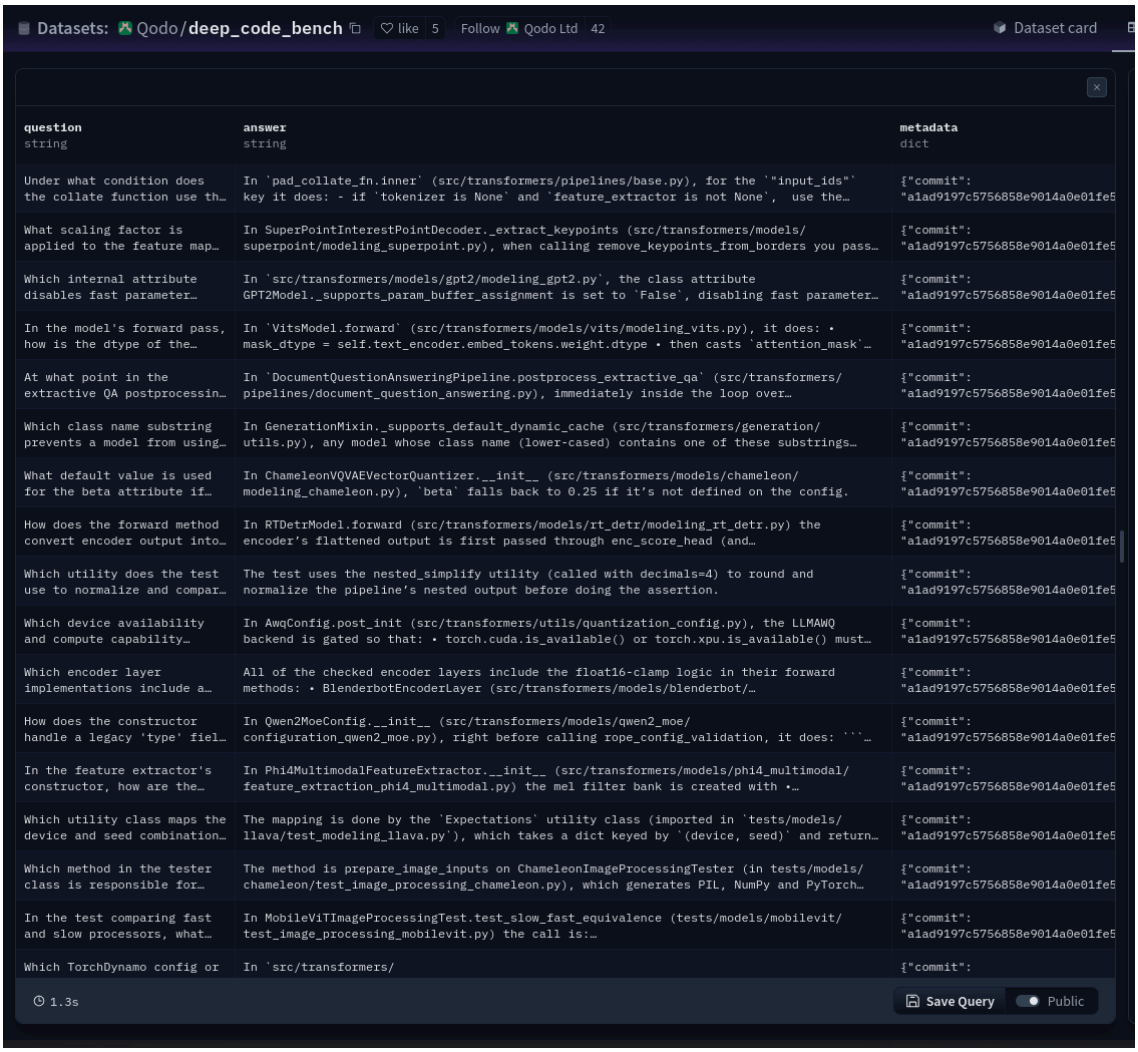
9.1.1 Standard Bench Mark:

We have to make sure the dataset used for evaluation is standardized and widely used for evaluating the deep code understanding of the models. The Qodo's Deep_code_bench dataset suited this checklist as it contains the question and answer pair mainly with the code context using the columns "facts"(List of factual statements supporting the answer) and "meta data"(Additional information about difficulty and scope). Therefore, instead of generating synthetic question answer pairs of code related questions, we have used the Deep_code_bench dataset for its robust standard nature.

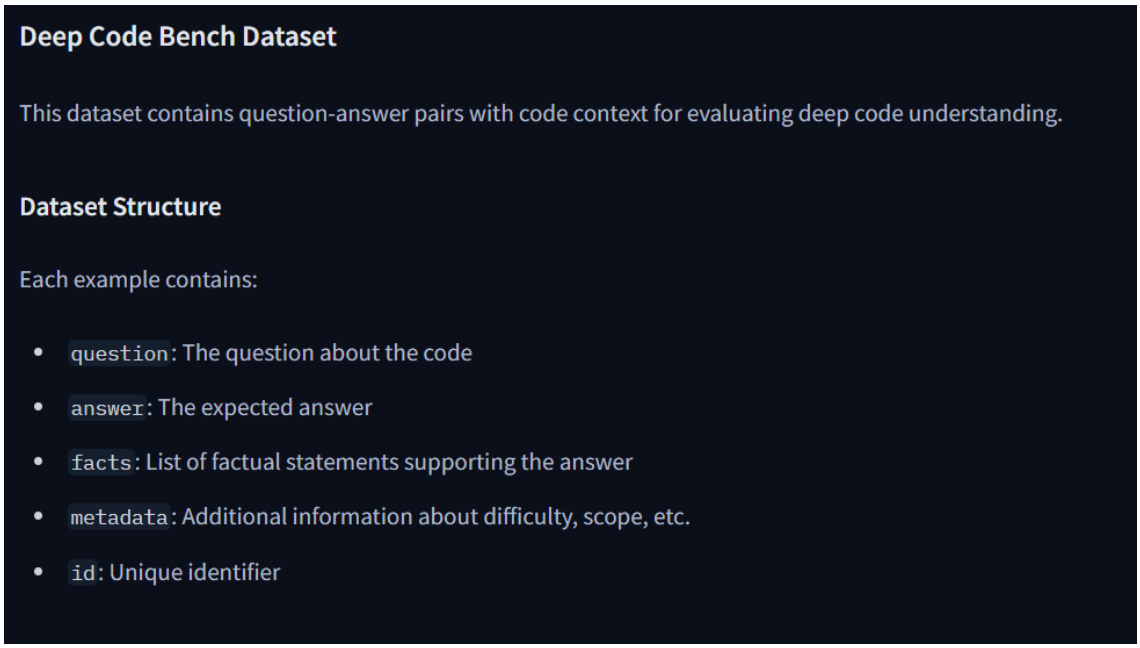


Reference Link:
https://www.qodo.ai/blog/deepcodebench-real-world-codebase-understanding-by-qa-benchmarking/

As our agentic framework is specifically for purely python based libraries and codebases, we had to filter out the bench mark dataset from the list of libraries or github repositories that are contained in its metadata. We had constrained ourselves with the graphiti, Keras and Microsoft's Qlib. Thus, the dataset when filtered contained 380 Question-Answer pairs along with extra information contexts, along with metadata describing the repository and question.

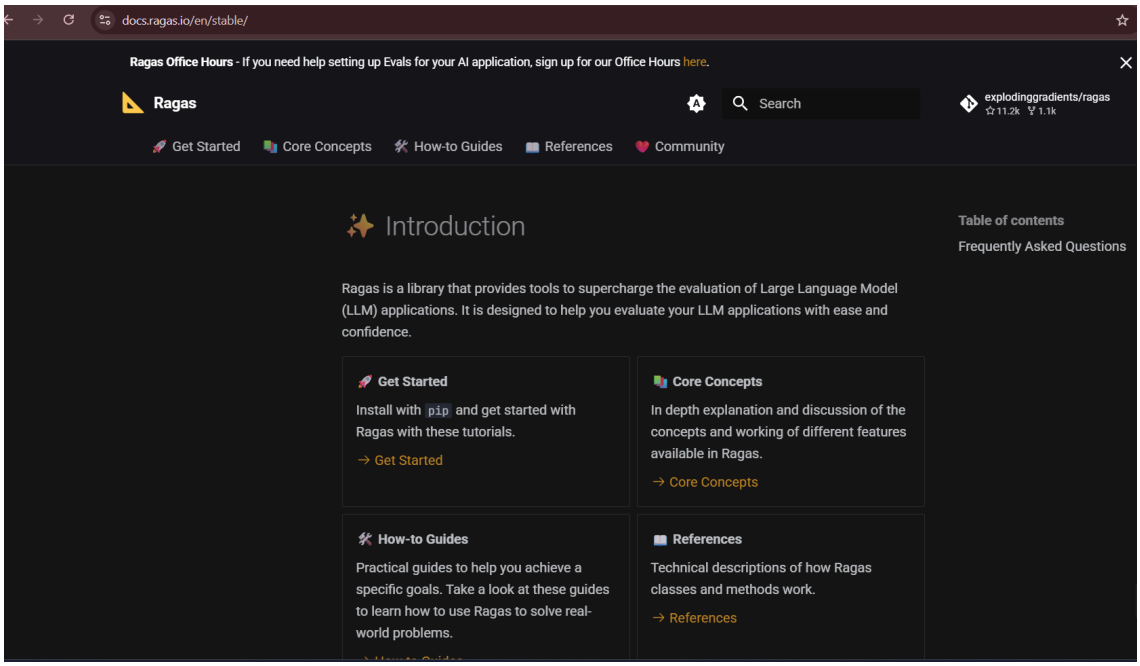


Dataset link: https://huggingface.co/datasets/Qodo/deep_code_bench

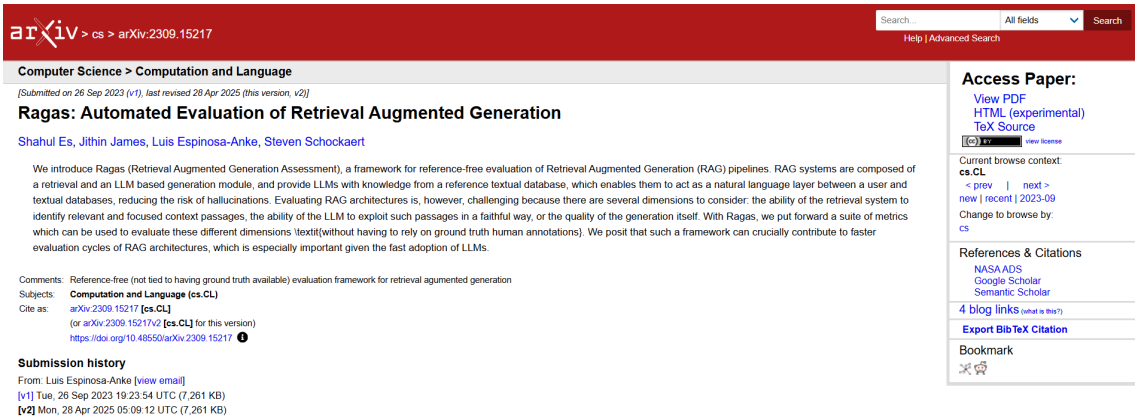


9.1.2 Framework and Technique:

We have used the RAGAS (Retrieval Augmented Generation Assessment), a python based library widely used for evaluation of production ready LLM applications. It provides an automated framework to evaluate the LLM's response both with and without RAG framework, purely based on metrics such as "Context Recall" ,"Faithfulness", "Factual Correctness(with modes such as accuracy, precision and F1 score)", and "Semantic Similarity". The overall score mean of these metrics is the RAGAS score, which can be averaged manually from the core metrics.



Reference Link: https://docs.ragas.io/en/stable/



Reference link:https://arxiv.org/abs/2309.15217

We have used ‘LLM as a judge' technique and embedding models under this RAGAS framework while also using to arrive at the correct score for each metric. While deciding to choose a LLM for judging, we must ensure that the judge model is very much advanced in reasoning. So, we have used GPT-5 as the judge to evaluate Aristotle agentic framework’s responses and the direct foundational generic base model Llama3.1:8B’s responses, separately. This is mainly done to compare the performance and evaluate the effectiveness of Aristotle’s agentic framework.

To make the evaluation more comprehensive and concrete in the production environment, we have recorded the responses of Aristotle in two scenarios. First scenario, when the users explicitly mention

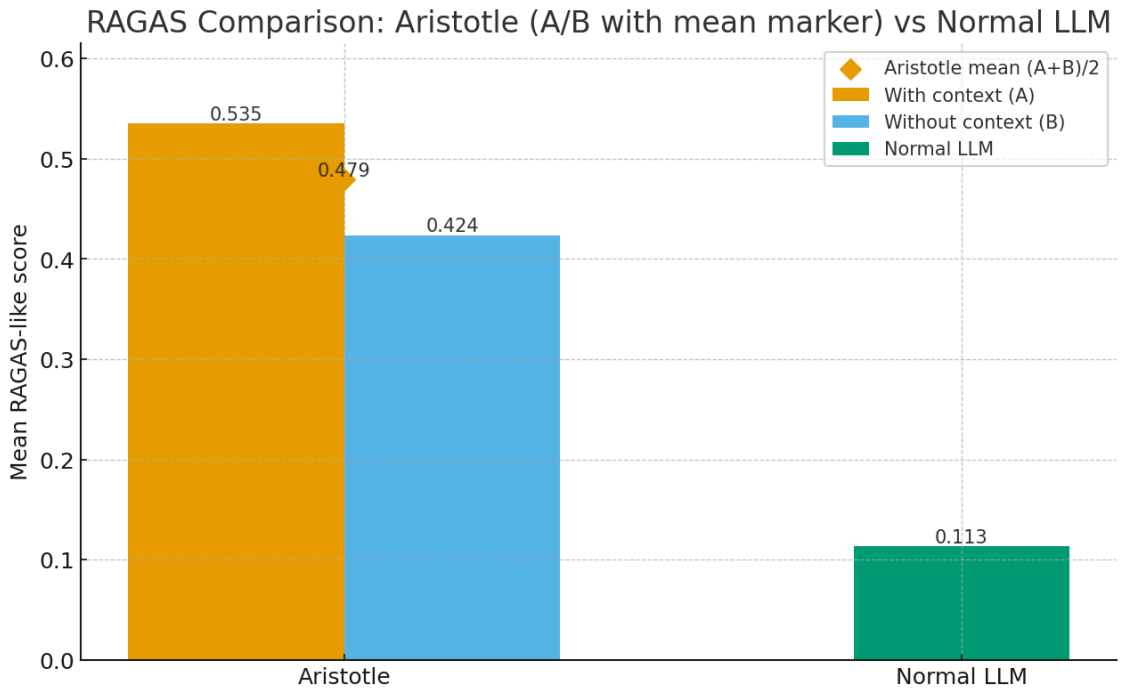
the context of the query and the second scenario when the users do not explicitly mention any context about their query.

9.2 Visualizing the evaluation metrics

1. **Aristotle Agentic Framework (With Llama3.1:8B) VS Normal LLM Without Aristotle Framework (Just Llama3.1:8B)**

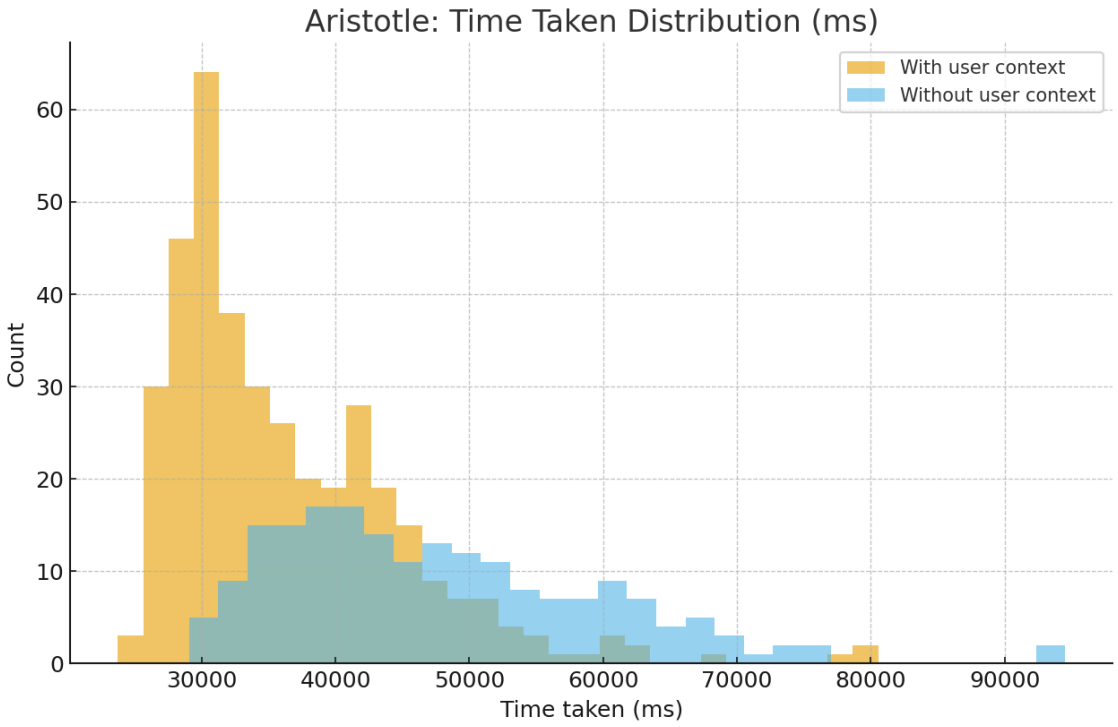
Framework/Model	mean_ragas
Aristotle (with context)	0.535162126
Aristotle (without context)	0.423769972
Normal LLM	0.113441643

Visualizing the RAGAS score



Here, both Normal LLM and Aristotle use the exact same foundational generic non-code generating model "Llama3.1:8B" that is not pre-trained for code generation. With the metrics, we can clearly conclude that the Aristotle Agentic framework's response is ~4.24 times better than the locally deployed generic model's direct response.

2. **Response Time taken distribution:**



Clearly, most of the questions with the "user context", Aristotle Agentic framework's response time is ~30secs (30000ms = 30secs), while in rare occasions, it reaches up to 80 secs ~ 1 min 20s, but answers accurately when compared to the same generic foundational model's response "without" Aristotle's framework like what we have seen with the RAGAS score.

10. Conclusion and Future Work

The VSCode extension of Aristotle Agentic framework being more accurate in its response with a mean RAGAS score of 0.479 compared to the same locally deployed model(score: 0.113), using smaller models with parameter size of just 8B, and comparatively better latency, it will help reduce significant amount of time wasted by tech professionals around the globe, improving their software development productivity rate, reducing the time taken from ideation to product/solution deployment, indirectly contributing to a good work-life balance, employee satisfaction, reduction in wastage of company resources like time, and significant overall positive growth in efficiency.

Researchers, freshers and non-technical coding aspirants can also use the Aristotle extension in VScode for learning to code and understand different libraries in a beginner friendly manner.

We have the GitHub link for reference: https://github.com/NUS-IRS-Project/Aristotle

From a technical point of view, this project significantly bridges the gap between the recent research works on LLMs, Knowledge Graphs, etc and the real life problem solving applications, while also being unique by adding an additional features of knowledge-based systems to address different problems such as LLM hallucinations and requiring the LLM to think too many times.

The future scope of this project is very wide, as this architectural design can be applied to any company's private code base or libraries, as an internal productivity tool within an organization or a project team while ensuring privacy features. The additional feature that we would like to add in the far future, is the automated coding capability as an interesting extrapolation of this project.

10.1 Findings & Recommendations

Findings : Vector+KG (+KBS) yields more grounded, doc-accurate answers and better structural reasoning than a standalone LLM, with acceptable interactive latency. Real-time workspace updates improve freshness compared to periodic re-indexing.

Recommendations:

1) Adopt Aristotle as an on-device editor companion for Python teams needing doc-grounded answers.

2) For enterprise roll-outs, add privacy controls (offline deployment, network rules) and observability (query traces, retrieval audits).

3) Extend KG schema/ingestion to other languages; evaluate multi-repo scaling and background pre-indexers.

4) Expand KBS rule coverage and automated query repair; explore GA-based model tuning.

11. Supplementary Materials, Acknowledgements and Appendices

11.1 Appendices

1. Feng, Y., Chen, X., Lin, B. Y., Wang, P., Yan, J., & Ren, X. (2020). Scalable Multi-Hop Relational Reasoning for Knowledge-Aware Question Answering (MHGRN). EMNLP. Link: https://arxiv.org/abs/2005.00646

2. Yasunaga, M., Ren, H., Bosselut, A., Liang, P., & Leskovec, J. (2021). QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering. NAACL. Link: https://aclanthology.org/2021.naacl-main.45/

3. Zhang, X., Yasunaga, M., et al. (2022). GreaseLM: Graph Reasoning Enhanced Language Models for Question Answering. ICLR. Link: https://arxiv.org/abs/2201.08860

4. Edge, D., Trinh, H., et al. (2024). A GraphRAG Approach to Query-Focused Summarization & Microsoft Research GraphRAG project/blog. Link: https://www.microsoft.com/en-us/research/publication/from-local-to-global-a-graph-rag-approach-to-query-focused-summarization

5. He, Y., Zheng, Z., et al. (2024). Explaining Graph Neural Networks with Large Language Models: A Counterfactual Perspective (LLM-GCE). arXiv. Link: https://arxiv.org/abs/2410.15165

6. Giorgi, F., Campagnano, C., Silvestri, F., & Tolomei, G. (2025). Natural Language Counterfactual Explanations for Graphs Using Large Language Models. AISTATS. Link: https://raw.githubusercontent.com/mlresearch/v258/main/assets/giorgi25a/giorgi25a.pdf

7. Pan, B., Yuan, J., et al. (2025). GraphNarrator: Generating Textual Explanations for Graph Neural Networks. ACL.Link: https://aclanthology.org/2025.acl-long.2

8. Avila, C. V. S., et al. (2024). Auto-KGQA: An Autonomous LLM-based Framework for Text-to-SPARQL. ESWC. Link: https://2024.eswc-conferences.org/wp-content/uploads/2024/05/77770162.pdf

9. Özsoy, M. G., & Tai, W. (2024–2025). Text2Cypher: Bridging Natural Language and Graph Querying (+ dataset and multilingual evaluation). Link: https://www.ijcai.org/proceedings/2024/0898.pdf

10. Tiwari, A., et al. (2025). Improving LLMs on Cypher Generation via LLM-Supervised Data Generation. NAACL Findings. Link: https://aclanthology.org/2025.naacl-short.53.pdf

11. Li, Y., Li, Z., et al. (2024). A Survey of Graph Meets Large Language Model: Progress and Future Directions. IJCAI. Link: https://www.ijcai.org/proceedings/2024/0898.pdf

12. 2024 Stack Overflow Survey on AI. Link: https://survey.stackoverflow.co/2024/ai

13. Özsoy, M. G., & Tai, W. (2024–2025). *Text2Cypher: Bridging Natural Language and Graph Querying (+ dataset and multilingual evaluation).* Link: https://www.ijcai.org/proceedings/2024/0898.pdf

14. EY. (2024). *EY survey reveals artificial intelligence is creating new hiring needs.* Link: https://www.ey.com/en_us/newsroom/2024/04/ey-survey-ai-creating-new-hiring-needs

15. Google. (2025). *Gemini Code Assist: Code customization overview.* Link: https://developers.google.com/gemini-code-assist/docs/code-customization-overview

16. EY. (2025). *AI Barometer 2025: Impact on work and workforce.* Link: https://www.ey.com/en_nl/services/ai/ai-barometer-2025-impact-op-werk-en-workforce

11.2 Acknowledgements

We would like to give acknowledgements to the professors who taught us at NUS-ISS:

- Dr. Gary Leung for this valuable advice on working as a team and for defining the project's requirements clearly.
- Dr. Xavier Xie for his in-depth teaching on knowledge graphs, which includes knowledge graphs embedding techniques, and knowledge graphs querying. He also taught us on how to implement a simple retrieval-augmented generation system that performs direct queries to the knowledge graph.
- Dr. Zhu Fangming for his feedback and recommendations on our project idea. He also taught us invaluable knowledge about evolutionary learning techniques, including genetic algorithms.
- Dr. Barry Adrian Shepherd for teaching us on how to implement recommendation systems, which included content-based similarity recommendation techniques.

11.3 Challenges and Road blocks that were resolved during the development duration:

We had identified several challenges and came up with a plan on how to address them:

- Identifying the appropriate LLMs to use in terms of size, deployment and free tier availability is a challenge in our case.

  We overcame this challenge with extensive research on different LLMs specifically focusing on their size and performance for an optimal user experience.

- Latency and Throughput of the system as there are several time-taking moving parts in the design along with the UI.

  We balanced the number of moving parts with their corresponding latency and performance with a conscious designing principle focused on optimum functionality.

- Time constraints of 1 month for the deliverable were managed carefully with time and product management principles.

11.3 Appendix of report: Mapped System Functionalities vs MRS, RS and CGS:

System Functionality	What it Covers	Machine Reasoning	Reasoning Systems	Cognitive Systems
Codebase Loading	Acquire files; Normalize structure	✅		
AST Parsing & Triplet Extraction	Build code entities/relations	✅		✅
Knowledge Graph in Neo4j	Code structure + temporal metadata	✅		✅
Graphiti	Link reasoning, neighborhood aggregation, Breadth First Search and Traversal in graph	✅	✅	✅
Vector Store (FAISS)	Semantic retrieval over docs			✅
KBS Relationship Validator	Reject/repair invalid queries	✅		
Agent (Llama3.1:8B)	Foundational Model integration		✅	
VS Code Extension UI	Chat: User experience, actions		✅	
Evaluation	DeepCode Bench + RAGAS, Accuracy & grounding assessment using Qwen3	✅	✅	