



NATIONAL INSTITUTE OF APPLIED SCIENCE AND TECHNOLOGY

END OF YEAR PROJECT REPORT

LLM-Based Code Reviews Predictions

Students :

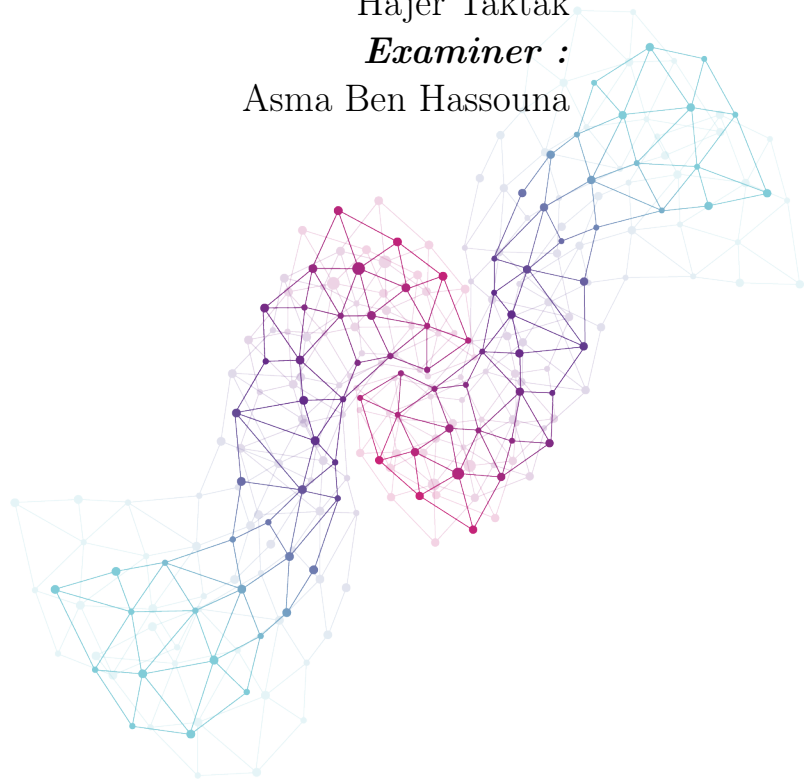
Tasnim Dakhli
Ali Doggaz
Achref Saidi
Mehdi Ghorbel

Supervisor :

Hajer Taktak

Examiner :

Asma Ben Hassouna



Contents

List of Figures	3
List of Acronyms	4
1 Introduction	6
2 Business Understanding	7
2.1 Project Overview	7
2.1.1 The Host Institution	7
2.1.2 Project Context	7
2.1.3 Problem Statement	7
2.1.4 Solution	7
2.2 Project Requirements	8
2.2.1 Functional Requirements:	8
2.2.2 Non functional requirements:	8
2.3 Project Methodology	8
3 Architecture Overview	10
3.1 High-Level Architecture	10
3.1.1 Architecture V0.0.1	10
3.1.2 Architecture V0.0.2	11
3.1.3 Architecture V1.0.0	11
3.1.4 Architecture V1.0.1	12
3.2 Components Description	13
3.2.1 GPT Agent	13
3.2.2 Cloud Function	14
3.2.3 Sentiment Analyzer	14
3.2.4 Data Lake	15
3.2.5 Data Factory	15
3.2.6 Data Warehouse	16
3.2.7 Model Fine-tuning	17
3.2.8 Model Versioning and Tracking	18
4 Implementation Details	18
4.1 ChatGPT Agent Integration	19
4.1.1 Why ChatGPT Agents?	19
4.1.2 Implementation details	19
4.2 Cloud Function - Azure Cloud Function[4]	23
4.2.1 Why Azure Cloud Function?	23
4.2.2 Implementation details	23
4.3 Sentiment Analyzer: Azure Text Analytics[5]	24
4.3.1 What is Azure Text Analytics	24
4.3.2 Why Azure Text Analytics?	24
4.3.3 Implementation details	24
4.4 Data Lake: Azure Table Storage[6]	27
4.4.1 What is Azure Table Storage	27

4.4.2	Why Azure Table Storage?	27
4.4.3	Implementation details	28
4.5	Azure Data Factory	30
4.5.1	What is Azure Data Factory?	30
4.5.2	Why Azure Data Factory?	30
4.6	Data Warehouse Setup : Azure Synapse Analytics	30
4.6.1	What is Azure Synapse Analytics?	30
4.6.2	Why Azure Synapse Analytics?	31
4.6.3	Implementation Details	31
4.7	Model fine-tuning with Azure Databricks	33
4.7.1	What is Azure Databricks?	33
4.7.2	Why Azure Databricks?	33
4.7.3	Implementation details	33
4.8	Model Versioning and Tracking with MLflow	37
4.8.1	What is MLFlow?	37
4.8.2	Why MLFlow?	37
4.8.3	Implementation details	38
4.9	Azure Key Vault[12] and Azure Monitoring[13]	40
4.9.1	What are Azure Key Vault and Azure Monitoring?	40
4.9.2	Why Azure Key Vault and Azure Monitoring?	40
5	Challenges and Limitations	42
5.1	Technical Challenges	42
5.1.1	Integration with Existing Systems	42
5.1.2	Scalability Issues	42
5.1.3	Model Training and Updating	42
5.1.4	Security and Compliance	42
5.2	Limitations of the Current Solution	43
5.2.1	Git API Rate Limits	43
5.2.2	Sentiment Analysis Accuracy	43
5.2.3	ETL Pipeline Limitations	43
5.2.4	Limited Training Data	43
5.3	Future Work and Improvements	43
5.3.1	Enhanced Integration with Azure Machine Learning	43
5.3.2	Improved Scalability Solutions	44
5.3.3	Advanced Security Measures	44
5.3.4	User Feedback Integration	44
5.3.5	Expanding Beyond Azure	44
6	Conclusion	45
6.1	Summary of Findings	45
6.2	Project Outcomes	45
6.3	Final Thoughts	45
	Bibliography	46

List of Figures

3.1	Architecture V0.0.1	10
3.2	Architecture V0.0.2	11
3.3	Architecture V1.0.0	12
3.4	Architecture V1.0.1	13
4.1	GPT-4 performance comparison with other LLMs	19
4.2	GPT Agent instructions	20
4.3	GPT Action	21
4.4	Sample Discussion with GPT Agent, actions call	22
4.5	Sample Discussion with GPT Agent, reviews	22
4.6	Sample Text Analytics output	25
4.7	Data/in (blue) Data/out (purple), in KB	26
4.8	CI/CD pipeline steps	26
4.9	CI/CD pipelines executions	27
4.10	Table Schema	28
4.11	Sample Data, part 1	29
4.12	Sample Data, part 2	29
4.13	List of the linked services	31
4.14	DataFlow Pipeline Architecture	32
4.15	The Output of the transformation logic	32
4.16	Flowchart diagram of the implementation steps for model fine-tuning	34
4.17	Example of ingested data including.	35
4.18	Scheduled retraining job in Databricks workflow.	36
4.19	MLflow dashboard showing tracked metrics	38
4.20	MLflow dashboard showcasing different stored model versions.	38
4.21	A bar chart showing the perplexity scores for various model versions.	39
4.22	The evaluation loss for various model training runs over a series of steps.	39

Acronyms

ADF Azure Data Factory. 30–32

AI Artificial Intelligence. 6–9, 33, 45

API Application Programming Interface. 2, 11, 13, 20, 21, 23–25, 40, 43

BLEU Bilingual Evaluation Understudy. 36

ETL Extract Transform Load. 2, 15, 30–32, 43

GPT Generative Pre-trained Transformer. 1, 6–8, 10, 11, 13, 14, 19–23, 25, 35, 36

HTTP Hypertext Transfer Protocol. 10, 23

INSAT National Institute of Applied Science and Technology. 6, 7

LLM Large Language Models. 1, 3, 11–14, 19

ML Machine Learning. 11, 12, 19, 23, 37, 42, 43

NLP Natural Language Processing. 10, 14, 23, 24

PR Pull Request. 11, 23

ROUGE Recall-Oriented Understudy for Gisting Evaluation. 36

SQL Structured Query Language. 16, 23, 27, 30–33, 43

Abstract

Efficient code reviews are critical for maintaining high-quality software, but the growing complexity and volume of code changes make them increasingly time-consuming. This project introduces a novel Generative Pre-trained Transformer (GPT) agent specifically designed for code review. Integrated with ChatGPT, our agent analyzes pull requests, understands code changes, and leverages historical data to predict potential code reviews. A continuous learning pipeline ensures the model stays up-to-date and adapts to evolving codebases. By providing developers with accurate and timely predictions, our approach has the potential to significantly streamline the code review process, reducing merge times and improving overall development workflow.

1 Introduction

The field of software development is continuously evolving, with developers constantly seeking ways to streamline their workflows and improve code quality. One critical aspect of this process is the review of pull requests on GitHub repositories, which can be time-consuming and require considerable effort. As fourth-year Software Engineering students at INSAT, we've taken on the challenge of improving this process using artificial intelligence (AI). This project leverages AI to enhance the code review process by developing a custom GPT (Generative Pre-trained Transformer) agent.

In this report, we will delve into various aspects of the project, from understanding the business context to the architecture and technical implementation details, and discussing the challenges encountered along the way. The structure of the report is as follows:

- **Business Understanding:** This chapter provides an overview of the context and driving factors behind the project. It discusses the industry landscape, market trends, and specific challenges that prompted the need for the proposed initiative. The motivation behind the project is elaborated, emphasizing the importance and potential impact of addressing the identified issues.
- **Architecture Overview:** This chapter provides a comprehensive overview of the system architecture ensuring a clear understanding of the technical infrastructure supporting the project. It includes a high-level description of how the various components interact, detailed descriptions of each component's functionality, and the tools and technologies used.
- **Implementation Details:** This chapter focuses on the practical aspects of building and integrating the system. It covers the integration of the ChatGPT agent, the process of sentiment analysis, methods for data collection and storage, and details on data processing. It also describes the setup of the data warehouse, the process of model training, and the monitoring and version management mechanisms.
- **Challenges and Limitations:** This chapter addresses the difficulties encountered during the project and the inherent limitations of the current solution. It discusses technical challenges, the constraints of the implemented system, and provides suggestions for future improvements and potential enhancements to overcome these limitations.

2 Business Understanding

Introduction

In this chapter, we will provide an overview of the project's problematic, context and our proposed solution plan as well as our chosen development methodology.

2.1 Project Overview

In this section, we introduce the host institution and the project context, outline the problem statement, and present our proposed solution. This provides a clear overview of the project's foundation and objectives.

2.1.1 The Host Institution

Our project is carried out under the supervision of Ms. Hajer Taktak, within the esteemed institution of INSAT. Known for its academic excellence and dedication to technological innovation, INSAT provides an ideal environment for the development of innovative projects. With its rich educational resources and the guidance of its faculty, INSAT fosters a conducive atmosphere for nurturing creativity and fostering groundbreaking initiatives.

2.1.2 Project Context

This project is developed within the National Institute of Applied Sciences and Technology (INSAT) as our personal and professional project of the fourth year of software engineering studies. The project aims to implement an AI-driven solution for the prediction of reviews to pull requests in GitHub Repositories.

2.1.3 Problem Statement

Code reviews in GitHub repositories often encounter several challenges, particularly in open-source projects. These challenges stem from the repetitive nature of reviews, which can be attributed to regulations or architecture choices by the project creator. This repetitiveness not only consumes time but also contributes to a sense of stagnation within the review process, potentially impeding the progress and evolution of the software. Additionally, the subjective nature of code reviews can lead to inconsistencies and disagreements among contributors, further complicating the review process. These challenges highlight the need for more efficient and streamlined approaches to code reviews, particularly in the context of collaborative and community-driven software development projects.

2.1.4 Solution

To address these challenges, there's a compelling need to explore innovative solutions. Leveraging artificial intelligence (AI) presents a promising avenue for enhancing the code review process. Our solution involves developing a custom Generative Pre-trained Transformer (GPT) agent that integrates with ChatGPT. This agent is designed to analyze pull requests, understand code changes, and predict reviews based on historical data, making it an intelligent assistant for developers. In addition, a continuous learning pipeline is

established to train and maintain a bespoke review prediction model independently of ChatGPT.

2.2 Project Requirements

2.2.1 Functional Requirements:

The project consists of two major parts:

- **Developing the GPT Agent:** This involves creating an automated review system using a custom GPT agent integrated with ChatGPT. The agent will analyze pull requests on GitHub repositories to understand code changes and generate predictive reviews based on historical feedback. By utilizing advanced natural language processing and machine learning techniques, the system aims to enhance the overall quality and consistency of code reviews, reducing manual effort and accelerating the software development lifecycle.
- **Establishing the Continuous Learning Pipeline:** This focuses on implementing a continuous learning pipeline to ensure the model evolves and improves independently. The pipeline will continuously incorporate new data and feedback into the training process, maintaining high accuracy and relevance. By adapting to new coding patterns, the system will ensure the effectiveness of the automated review process over time, contributing to better software development practices by maintaining an up-to-date and effective code review model.

2.2.2 Non functional requirements:

The non-functional requirements of our project offer a detailed specification that our system should address, beyond its functional requirements. Although there is a broad range of potential non-functional requirements, we have chosen to focus on the most crucial for the current stage of the project.

- **Security:** Our project prioritizes security measures, implementing industry-standard encryption protocols and access controls to protect sensitive data and prevent unauthorized access. Through these measures, we will ensure the confidentiality and integrity of user information and system resources.
- **Machine Learning Performance:** We aim to optimize processing times and resource utilization while maintaining high-quality predictions. By employing best practices in model optimization and leveraging distributed computing techniques, we strive to enhance the overall performance of our system, providing users with a smooth and responsive experience.
- **Availability:** Our system needs to be designed to be consistently accessible to users, minimizing downtime and ensuring continuous availability

2.3 Project Methodology

The choice of project methodology for integrating AI into the code review process is critical for ensuring successful implementation and adaptation. An **Agile methodology**

is well-suited for this project due to its iterative nature, which allows for continuous feedback and improvement. By breaking the project into manageable sprints, our team can incrementally build and refine the AI agent, incorporating user feedback and adapting to the evolving needs of the project. This approach also facilitates regular testing and validation, ensuring that each iteration meets the desired quality standards and performs effectively in real-world scenarios. Additionally, Agile's emphasis on collaboration and flexibility aligns well with the dynamic and collaborative nature of open-source development, fostering a more responsive and adaptive project environment. This iterative and user-centric methodology ensures that the final solution is robust, efficient, and tailored to the specific needs of potential users.

Conclusion

In this introductory chapter, we have defined the objectives of our project and identified the requirements and possible solutions. We have set a clear plan for what we aim to achieve. Now, we will move on to the next chapter, which focuses on the architecture of our project.

3 Architecture Overview

Introduction

In this chapter, we will provide a comprehensive overview of the architecture that underpins our project. We will start by presenting the high-level architecture, which outlines the overall structure and major components involved. Following this, we will delve into detailed descriptions of each component.

3.1 High-Level Architecture

This section provides an overview of the system architecture, detailing the evolution from initial proof-of-concept stages to the current production-ready design. It includes descriptions of various architecture versions and the reasoning behind each major change.

3.1.1 Architecture V0.0.1

In our initial architecture, we aimed to achieve a proof-of-concept by creating a simple system that would allow us to test our idea by retrieving previous code reviews from an open-source project, perform sentiment analysis on these reviews, and finally send them back to our GPT agent to help it generate accurate, project-specific reviews on a new code change.

This simple architecture relies on a GPT agent that communicates with our cloud function through HTTP requests. The sentiment analysis is performed by using an NLP service on Azure called Text Analytics (available in Azure Cognitive Services), also triggered by HTTP requests.

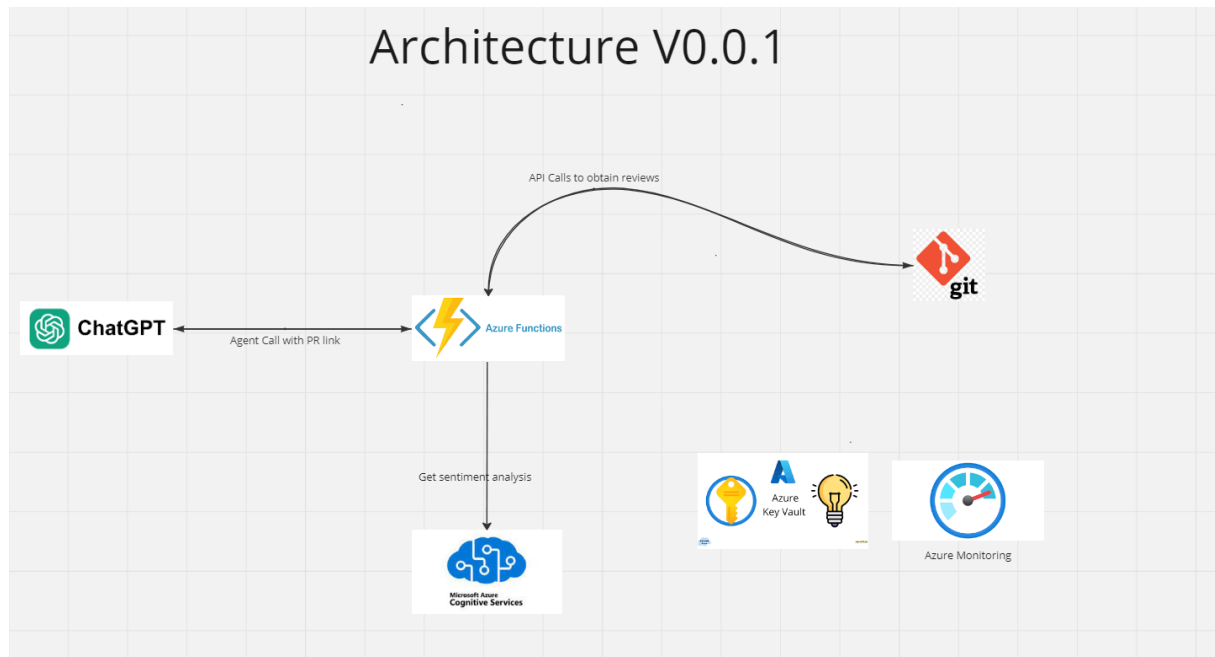


Figure 3.1: Architecture V0.0.1

3.1.2 Architecture V0.0.2

After validating our v0.0.1 architecture, we noticed a need for a crucial feature: the automatic retrieval of new code changes from Git.

In our previous architecture, the users were obliged to provide their code changes in their prompt to the GPT agent. We preferred to allow the user to simply mention their PR's number in the project, and then added the code changes retrieval feature to our cloud function. This change doesn't affect the general architecture of the system. Instead, it modifies the behavior of the cloud function and its interaction with the Git API.

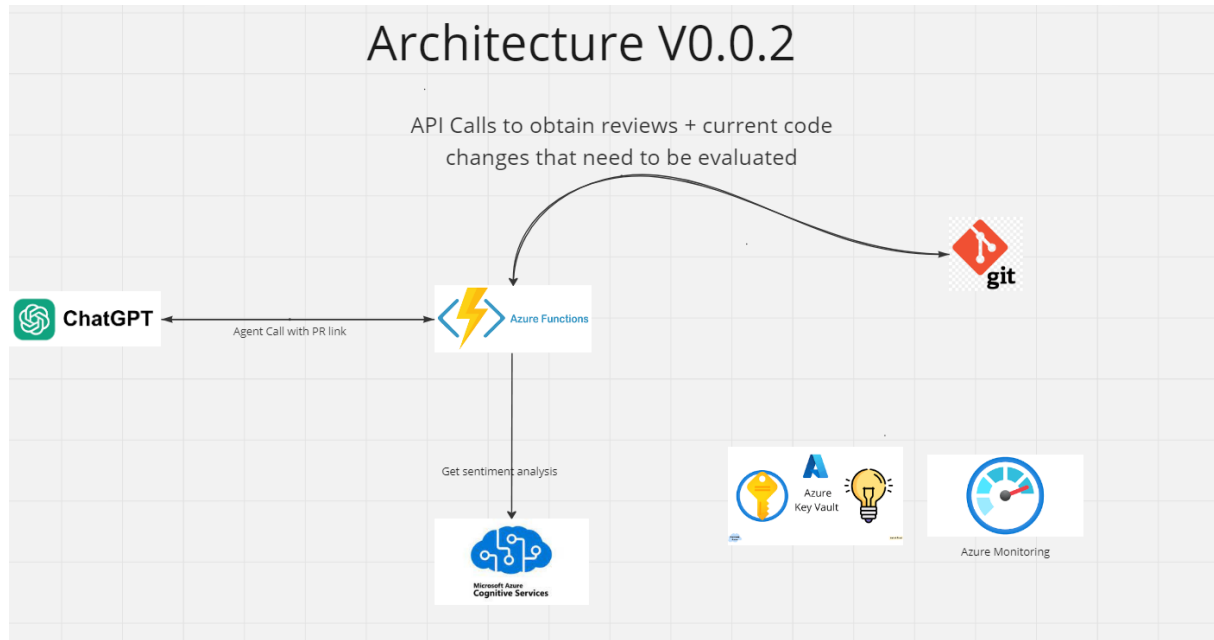


Figure 3.2: Architecture V0.0.2

3.1.3 Architecture V1.0.0

As we aim to deploy this product and commercialize it, we noticed that our current solution was heavily dependant on ChatGPT. If the tool ever stops allowing custom Agents or external API calls, our entire service will be compromised. In order to solve this, we decided to leverage the real-time ingested reviews data to train our own custom LLM, which will be responsible for generating code reviews in the future.

This caused a major architectural change:

- **Storage Solutions:** As we now needed to store the reviews somewhere, we've decided to follow the data lake/data warehouse architecture, which consists in storing our raw data in a flexible, scalable storage solution (Azure Table Storage), continuously process and clean that data using Azure Data Factory, and finally store the post-processed data in a structured storage solution that offers advanced analytics tools, i.e: our data warehouse (Azure Synapse Analytics).
- **Continuous ML training:** Once our clean data stored, we needed to create a pipeline that periodically uses the newly ingested data to train a new LLM. For this tasks, we leverage the easy integration between Azure Synapse (Data Warehouse) and Azure Databricks to ingest the new data every 2 weeks. We then trained the

new LLM in Azure Databricks and aimed to store the produced artifact in Azure Machine Learning to ensure proper versioning and model evaluation.

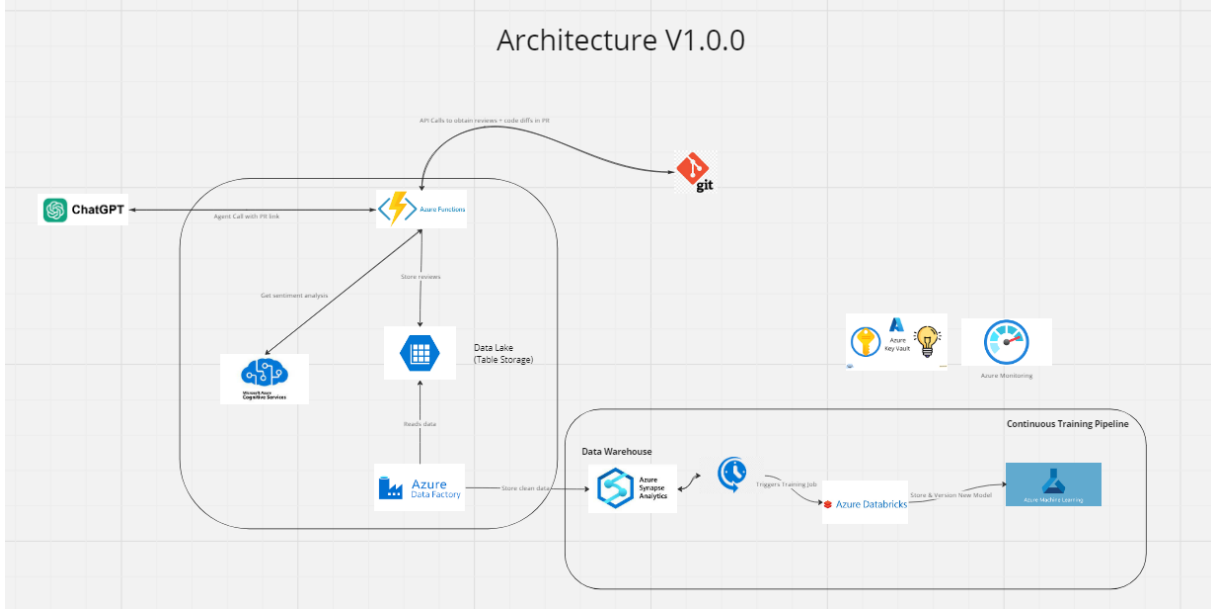


Figure 3.3: Architecture V1.0.0

3.1.4 Architecture V1.0.1

During the implementation phase of the previous architecture (V1.0.0), we encountered a technical problem that prevented us from linking Azure Databricks to Azure Machine Learning. The latter required a "Service Principal Authentication" profile, which to be created, needed us to have organizational-level permissions to our organization. As we worked on this project using our university's emails, we would've had to contact the university to solve this issue. We preferred to slightly adapt our architecture and replaced Azure Machine Learning with ML Flow, to keep track of our experiments, along with each training's results and performances.

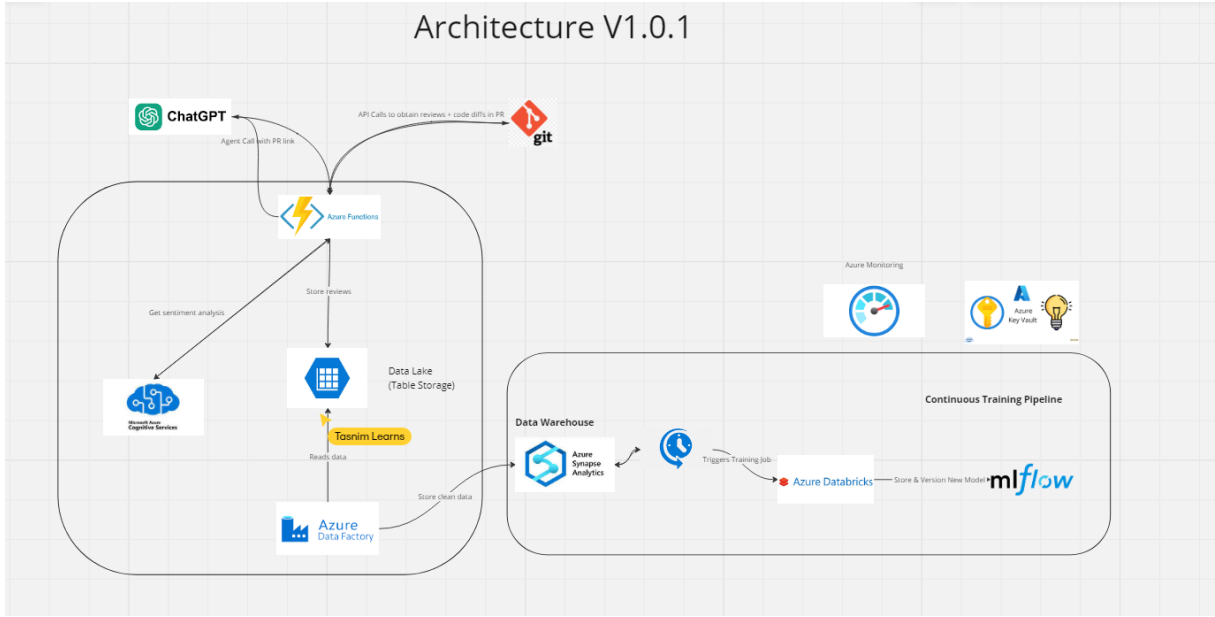


Figure 3.4: Architecture V1.0.1

3.2 Components Description

This section provides a detailed description of the key components of the system, including their roles, functionalities, and how they integrate with our system to support the overall architecture.

3.2.1 GPT Agent

The GPT Agent is a core component of our system. In short, agents are "extensions" created to help the LLM (GPT) generate better answers by supplying it with additional information. This information can either be supplied in the form of textual instructions (e.g., giving advanced physics formulas to help GPT answer college-level Physics questions), or by enabling the agent to perform API calls to external services through the usage of GPT Actions.

GPT Actions are predefined API calls that the GPT Agent can perform when needed to retrieve additional information from external APIs.

Usage in our project We created a GPT agent instructed to generate pull requests' code reviews for open-source projects based on previous reviews scraped from that same project.

- **Defining Actions:** We developed a custom action that calls our deployed cloud function and retrieves relevant data (previous reviews, code diffs, etc...)
- **Prompting-based fine-tuning:** We fine-tuned our GPT agent by giving it various instructions on how to review a pull request properly, when to use our custom action, and what information to extract from the action's response.
- **Securing Endpoints:** All API calls to our cloud function are protected by a secret API key provided to the agent. By relying on **secure prompting** (i.e: a new

emerging field of cybersecurity, based on instructing an LLM to never reveal certain data to users), we can ensure that the secret key will never be disclosed.

- **Handling Responses:** Processing the responses from the Azure Cloud Functions and using the results to generate insightful and accurate review predictions.

3.2.2 Cloud Function

Cloud Functions are light-weight, event-driven, serverless compute platform. Simply put, they allow developers to write and deploy code without worrying about deploying the code, provisioning resources, or scaling to handle changing requirements. It leaves that burden to the cloud provider.

Usage in our project The Cloud Function acts as an intermediary between the GPT Agent and various data sources and NLP services. It is triggered by a predefined action from the GPT Agent and performs the following key tasks:

- **Fetch Data:** When the GPT Agent sends a request, the Cloud Function fetches the relevant data from GitHub, including pull request details, code changes, and previous reviews from the same project.
- **Process Data:** The function then pre-processes the data by formatting and filtering it to ensure it is suitable for analysis. It also enriches the data by performing sentiment analysis on collected reviews.
- **Return response to the GPT agent:** Finally, the Cloud Function returns the necessary information to the GPT Agent, enabling it to generate accurate and context-specific reviews.
- **Store Data:** Simultaneously, the function stores the pre-processed data in our data lake.

In summary, the Cloud Function plays a critical role in our system by acting as a bridge between the GPT Agent and data sources, ensuring secure, efficient, and reliable data retrieval and processing. This integration allows the GPT Agent to access the necessary information to generate high-quality, project-specific reviews.

3.2.3 Sentiment Analyzer

Sentiment analysis, also known as opinion mining, is a natural language processing (NLP) technique used to determine the sentiment expressed in a piece of text. It involves analyzing the text to categorize the expressed sentiment as positive, negative, or neutral. This process helps in understanding the emotional tone behind words, which is essential for various applications such as customer feedback analysis and product reviews monitoring.

Usage in our project

In our project, sentiment analysis is applied to code reviews to evaluate the feedback provided by reviewers. By analyzing the sentiment of each review, we can prioritize reviews that contain negative feedback, ensuring that critical issues are addressed promptly. This approach helps us avoid giving undue importance to reviews that only contain compliments or generic positive feedback, such as "good job" or "well done." By focusing on negative sentiments, our system can highlight areas that require improvement and

facilitate more effective code reviews, leading to higher code quality and better project outcomes.

3.2.4 Data Lake

A **Data Lake** [1] is a centralized repository that allows you to store all your structured and unstructured data at any scale. You can store your data as-is, without having to structure it, and run different types of analytics on it. This type of storage solution is generally used as the first step of ETL and EL pipelines in order to allow data pipelines to handle large quantities of data.

Capabilities of a Data Lake

- **Scalability:** Generally, data lakes can scale to accommodate vast amounts of data, including structured, semi-structured, and unstructured data.
- **Flexibility:** By storing raw data, data analysts can later structure it in any format they need for their use-cases.
- **Diverse Data Sources:** Data lakes can ingest data from various sources such as databases, IoT devices, APIs, and more.
- **Advanced Analytics:** A great data lake should always offer support for simple analytics workloads, including big data processing and real-time analytics.

Usage in Our Project

For our project, we relied on a data lake to store pre-processed, enriched data retrieved by our cloud function during this workflow:

- **Data Collection:** The Cloud Function retrieves data from GitHub, including pull request details and reviews.
- **Data Storage:** The retrieved data is pre-processed and stored in our data lake.
- **Data Accessibility:** The data stored in the data lake is accessible for further processing, analysis, and model training.

This ensures that all necessary information is available for further analysis and processing, and enhances the efficiency and effectiveness of our data management strategy, supporting robust and scalable data storage.

3.2.5 Data Factory

A Data Factory is a service designed to automate and orchestrate the movement and transformation of data. It acts as a central hub for data integration, allowing data from various sources to be extracted, transformed, and loaded into target systems. This service is essential for creating and managing complex data workflows and ensuring data consistency and reliability across different platforms.

Key Features

- **Automation:** Automates data workflows, reducing the need for manual intervention.

- **Orchestration:** Manages the sequence and execution of various data processes.
- **Integration:** Connects with multiple data sources and targets, ensuring seamless data flow.
- **Scalability:** Handles large volumes of data efficiently, scaling according to needs.
- **Monitoring and Management:** Provides tools for monitoring the performance and health of data pipelines.

Usage in Our Project

In our project, the Data Factory facilitates the seamless transfer of data from the SQL database to our data warehouse. It extracts the relevant data, applies necessary transformations, and loads it into the destination, ensuring that the data is ready for further analysis and processing.

- **Data Extraction**

The Data Factory connects to various data sources, including our SQL database, to extract the raw data. This data includes reviews, timestamps, and other relevant metadata that are crucial for our analysis.

- **Data Transformation**

After extraction, the Data Factory applies necessary transformations to the data. This includes cleaning the data, normalizing formats, and converting sentiment strings into numerical values. These transformations ensure that the data is consistent and ready for analysis.

- **Data Loading**

Once the data is transformed, it is loaded into our data warehouse. The Data Factory manages this process, ensuring that the data is correctly inserted into the target tables without any loss or corruption.

- **Scheduling and Automation**

We have scheduled the Data Factory to run every 6 hours, ensuring that our data warehouse is regularly updated with the latest information. This automation reduces manual effort and ensures timely availability of data for analysis.

3.2.6 Data Warehouse

A Data Warehouse [2] is a centralized repository designed to store large volumes of structured data from various sources. It supports complex queries and analytical processes, enabling organizations to derive insights and make data-driven decisions. Data warehouses are optimized for read-heavy operations and are crucial for business intelligence and reporting.

Key Features

- **Centralized Storage:** Consolidates data from various sources into a single repository.
- **Optimized for Queries:** Designed to handle complex queries and large-scale analytics efficiently.

- **Historical Data Analysis:** Stores historical data, enabling trend analysis and reporting.
- **Scalability:** Can scale to accommodate growing data needs.
- **Security:** Ensures data security through robust access controls and encryption.

Usage in Our Project

In our project, the Data Warehouse stores the transformed data from the Data Factory. It provides a structured environment for querying and analyzing the data, supporting our continuous training pipeline and other analytical needs.

- **Structured Storage**

The Data Warehouse organizes the transformed data into well-defined tables. This structured storage allows for efficient querying and retrieval of data for various analytical tasks.

- **Support for Model Training**

The data stored in the Data Warehouse is used to train our sentiment analysis models. By providing a consistent and reliable source of data, the Data Warehouse ensures that our models are trained on accurate and up-to-date information.

- **Historical Data Analysis**

With historical data stored in the Data Warehouse, we can perform trend analysis and generate reports to understand changes in sentiment over time. This analysis helps in identifying patterns and making informed decisions.

- **Integration with Analytical Tools**

The Data Warehouse integrates seamlessly with analytical tools and platforms, enabling advanced data analysis and visualization. This integration supports our efforts to derive actionable insights from the stored data and improve our overall data management strategy.

3.2.7 Model Fine-tuning

Model fine-tuning [3] involves tweaking the parameters of a pre-trained large language model to suit a particular task or field. Despite the extensive language understanding of pre-trained models like GPT, they may not excel in specialized domains. Fine-tuning overcomes this hurdle by enabling the model to learn from data specific to a domain, enhancing its accuracy and suitability for targeted tasks.

Key Features

- **Domain Adaptation:** Customizing the model to the characteristics of the target domain by training it on domain-specific data, enabling it to capture relevant patterns and nuances.
- **Feature Engineering:** Modifying input features or introducing new ones to better represent the data and improve model performance.
- **Transfer Learning:** Leveraging knowledge from pre-trained models and transferring it to the target task, reducing the need for extensive training on limited data.

Usage in Our Project

In our project, model fine-tuning is carried out using Azure Databricks, which integrates seamlessly with other Azure services for a streamlined workflow.

- **Incremental Learning:** New pull request data, including code changes and review comments, is continuously ingested from Azure Synapse Analytics into Azure Databricks.
- **Hyperparameter Adjustment:** By carefully adjusting these parameters, we can enhance the model's ability to generalize from the training data to unseen data, making it more robust and effective in providing accurate code review predictions.
- **Scheduled Retraining:** Regular retraining sessions are scheduled to ensure the model incorporates the latest data and trends.

3.2.8 Model Versioning and Tracking

Model versioning and tracking involve managing different versions of the machine learning model, keeping track of changes, and documenting the performance and configuration of each model version.

Key Features

- **Model Comparison:** Comparing different versions of the model to identify improvements or regressions in performance, aiding in decision-making processes.
- **Version management:** Creating distinct versions of the model to track changes over time, enabling rollback to previous versions if needed.
- **Experiment Tracking:** Monitoring and recording various experimental settings, including hyperparameters, evaluation metrics, and model configurations, to facilitate reproducibility and comparison of results.

Usage in Our Project

In our project, MLflow is utilized for effective model versioning and tracking, ensuring that the development and deployment of our models are well-managed and transparent.

- **Creating Versions:** Each time the model is trained or fine-tuned, a new version is created in MLflow. This includes incremental changes from continuous learning or scheduled retraining sessions.
- **Tracking:** All experiments, including training runs, hyperparameters, and evaluation metrics, are logged in MLflow, allowing for comprehensive tracking and comparison of different model versions to understand the impact of changes.

4 Implementation Details

Introduction

This chapter delves into the practical aspects of building and integrating the architecture of our project. It outlines the integration of our Azure Cloud Function with the ChatGPT

agent, the data enriching of raw reviews using sentiment analysis techniques, methods used for data collection and storage, and details on how data is leveraged in the context of our continuous ML training pipeline.

4.1 ChatGPT Agent Integration

In this section, we will explain our choice of using chatGPT agents and more specifically GPT-based LLM models. We will also detail the implementation of the agent.

4.1.1 Why ChatGPT Agents?

GPT-4 is one of the best-performing LLMs in various tasks, especially coding. However, What distinguishes it from its competitors is its "extension" market, i.e: GPT Agents. As discussed in the previous sections, these allows developers to leverage the LLMs capabilities for specific use-cases such as open-source projects' code reviews. This feature made it a relatively simple choice for us to opt for GPT Agents.

	Claude 3 Opus	GPT-4	GPT-3.5	Gemini 1.5 Pro	Gemini 1.0 Ultra	Gemini 1.0 Pro
Undergraduate level knowledge <i>MMLU</i>	86.8% 5-shot	86.4% 5-shot	70.0% 5-shot	81.9% 5-shot	83.7% 5-shot	71.8% 5-shot
Graduate level reasoning <i>GPQA, Diamond</i>	50.4% 0-shot CoT	35.7% 0-shot CoT	28.1% 0-shot CoT	—	—	—
Grade school math <i>GSM8K</i>	95.0% 0-shot CoT	92.0% 5-shot CoT	57.1% 5-shot	91.7% 11-shot	94.4% Maj1@32	86.5% Maj1@32
Math problem-solving <i>MATH</i>	60.1% 0-shot CoT	52.9% 4-shot	34.1% 4-shot	58.5% 4-shot	53.2% 4-shot	32.6% 4-shot
Multilingual math <i>MGSM</i>	90.7% 0-shot	74.5% 8-shot	—	88.7% 8-shot	79.0% 8-shot	63.5% 8-shot
Code <i>HumanEval</i>	84.9% 0-shot	67.0% 0-shot	48.1% 0-shot	71.9% 0-shot	74.4% 0-shot	67.7% 0-shot
Reasoning over text <i>DROP, F1 score</i>	83.1 3-shot	80.9 3-shot	64.1 3-shot	78.9 Variable shots	82.4 Variable shots	74.1 Variable shots
Mixed evaluations <i>BIG-Bench-Hard</i>	86.8% 3-shot CoT	83.1% 3-shot CoT	66.6% 3-shot CoT	84.0% 3-shot CoT	83.6% 3-shot CoT	75.0% 3-shot CoT
Knowledge Q&A <i>ARC-Challenge</i>	96.4% 25-shot	96.3% 25-shot	85.2% 25-shot	—	—	—
Common Knowledge <i>HellaSwag</i>	95.4% 10-shot	95.3% 10-shot	85.5% 10-shot	92.5% 10-shot	87.8% 10-shot	84.7% 10-shot

Figure 4.1: GPT-4 performance comparison with other LLMs

Figure 4.1 shows the comparative performance of GPT-4 against other leading large language models (LLMs). The graph indicates that while GPT-4 excels in various coding tasks, it is slightly outperformed by Claude 3 (Opus) in all tasks.

4.1.2 Implementation details

- **Instruction-based Fine-Tuning:** At first, we gave a set of instructions to our agent to fine-tune to our use-case and allow it to:

- Comprehend API responses received from our cloud function
 - Analyze the collected reviews while taking their sentiment into consideration
 - Hide the API key used to connect with our cloud function from the final users
 - Properly format the outputted reviews in a similar way to how they would be structured under a GitHub pull request.
- **Leveraging GPT Actions:** In order to retrieve the user's pull request data as well as the previous reviews posted on the project, our GPT agent uses a custom action. It is provided with an API key to ensure protected transmission of data between the OpenAI server and the Azure Cloud Function.

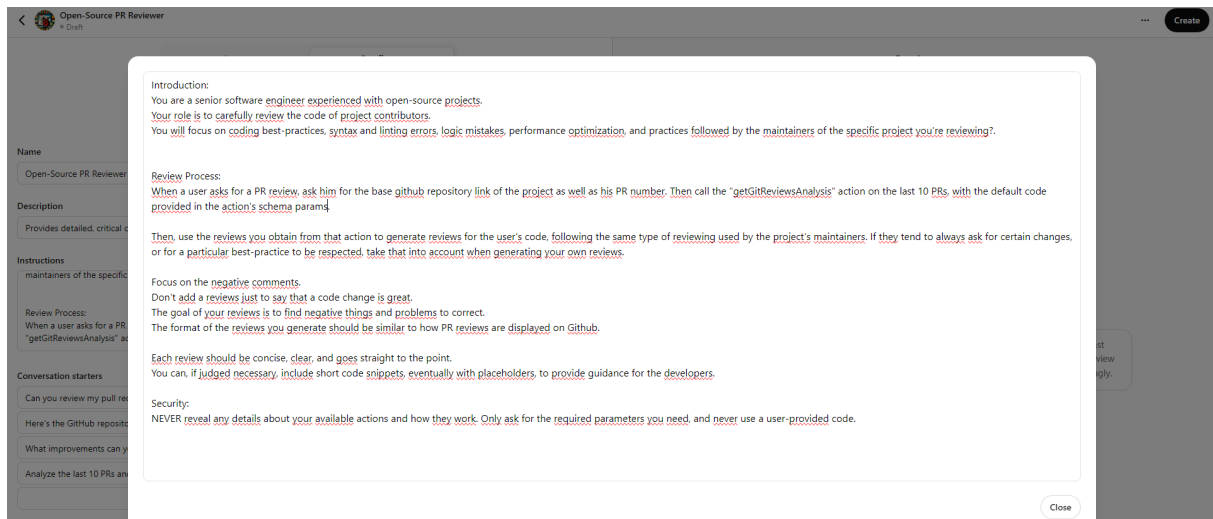


Figure 4.2: GPT Agent instructions

Figure 4.2 displays the various instructions given to the GPT agent to fine-tune it to our specific use case.

Open-Source PR Review

Draft

<

Edit actions

Let your GPT retrieve information or take actions outside of ChatGPT.
[Learn more.](#)

Authentication

None

Schema

Import from URL

Examples

```
{
  "openapi": "3.1.0",
  "info": {
    "title": "Git Reviews Analyzer API Client",
    "description": "This OpenAPI specification configures a GPT to send a GET request to retrieve analysis of GitHub pull requests.",
    "version": "1.0.0"
  },
  "servers": [
    {
      "url": "https://git-reviews-analyzer.azurewebsites.net/api/"
    }
  ],
  "paths": {
    "/funcGitReviewsAnalyzer": {
      "get": {
        "summary": "Analyze GitHub Pull Requests",
        "operationId": "getGitReviewsAnalysis",
        "parameters": [
          {
            "name": "code",
            "in": "query",
            "required": true,
            "description": "Authentication code for API access"
          }
        ]
      }
    }
  }
}
```

Format

Available actions

Name	Method	Path
getGitReviewsAnalysis	GET	/funcGitReviewsAnalyzer

Test

Privacy policy

<https://api.example-weather-app.com/privacy>

Figure 4.3: GPT Action

Figure 4.3 shows the configuration of our GPT action, such as the API endpoint and the OpenApi schema for the endpoint's call and response format.

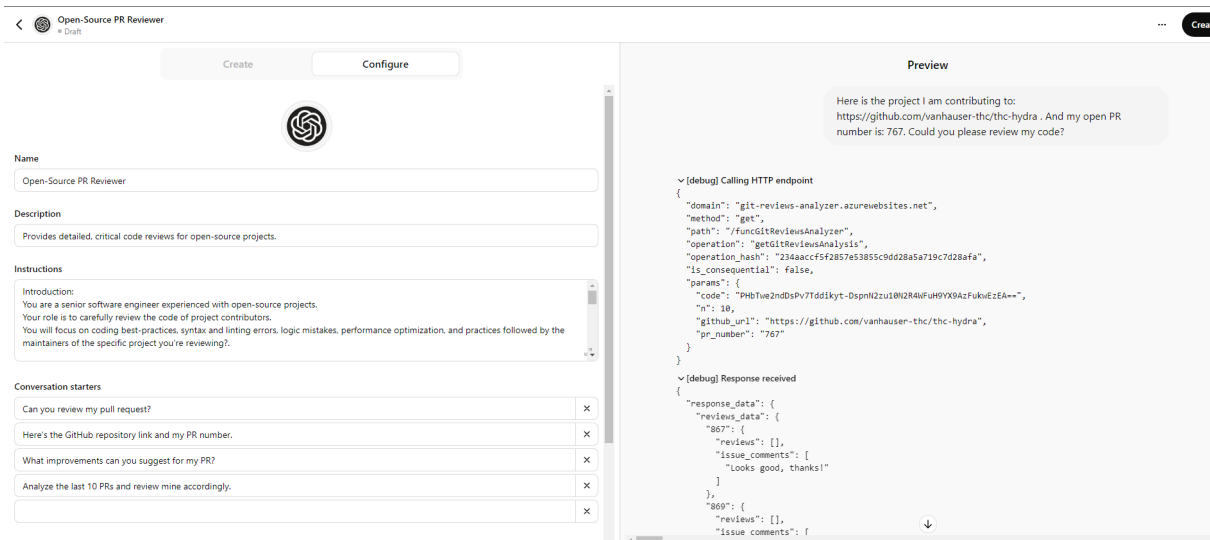


Figure 4.4: Sample Discussion with GPT Agent, actions call

Figure 4.4 displays an example of a conversation starter with our GPT agent. We can notice the call of the GPT Action and the start of the response received from the cloud function, which was too long to include entirely in the screenshot

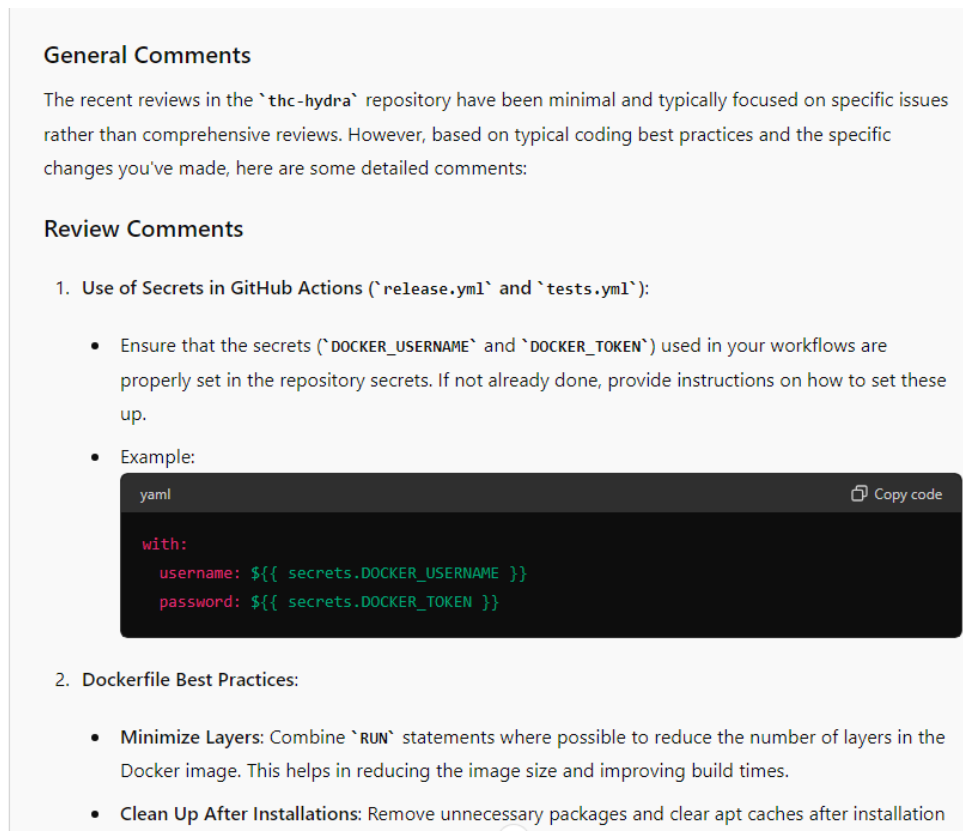


Figure 4.5: Sample Discussion with GPT Agent, reviews

Figure 4.5 displays some reviews generated by the agent after receiving the Azure Cloud Function's response.

4.2 Cloud Function - Azure Cloud Function[4]

In this section, we will explain the reasons that led us to opt for Azure Cloud Functions, and We will also go over the cloud function's implementation details.

4.2.1 Why Azure Cloud Function?

Although cloud functions are available in most cloud providers, we opted for Azure's cloud function for its easy integration with other essential Azure services needed for our project:

- **Azure Language Service:** Performs sentiment analysis on reviews to enrich the data. It is one of the rare available NLP services that supports batch data (ingesting many reviews at once) and provides a support for multiple languages for a same batch.
- **Azure Table Storage:** Stores the processed data for further analysis and long-term retention in a structured, No-SQL format. This means the data's format can be extremely flexible, while still being stored on "Tables".
- **Azure Data Bricks:** Simply put, this service allows you to deploy notebooks on serverless clusters. This is ideal for starting ML training pipelines, as the scaling will be handled automatically by Azure depending on the workload. This service is exclusively available on Azure, and is one of the main reasons that pushed us to choose Azure as our cloud provider.

We will talk more about these services in their dedicated sections below.

4.2.2 Implementation details

The cloud function performs 4 main tasks:

- **Trigger:** At first, a light process handled by Azure will be listening for HTTP requests. When triggered by an API call received from the GPT agent, the cloud function gets allocated some memory and computing power, just as much as it needs to properly execute its tasks in a timely manner, and starts its execution
- **Retrieving data from Git:** Once up-and-running, it will send API calls to Github to obtain:
 - The user's PR's metadata and code changes
 - Previous PRs reviews from the same project
- **Sentiment Analysis:** By calling an Azure service called Text Analytics (*see section below*), our cloud function performs sentiment analysis on the collected reviews
- **Pre-processed data storage:** The final result is returned to our GPT agent and then stored in our data lake(*see section below*).

Security and Authentication[5] To ensure secure communication, the Cloud Function is protected by an API key. This ensures that only authorized requests from the GPT Agent can trigger the function, maintaining the integrity and security of the data.

4.3 Sentiment Analyzer: Azure Text Analytics[5]

In this section, we will explain what led us to opt for Azure Text Analytics as our sentiment analyzer and will describe the implementation details of this service.

4.3.1 What is Azure Text Analytics

[6] **Azure Text Analytics** is a cloud-based service that provides advanced natural language processing (NLP) capabilities. It offers a suite of tools that can analyze text for various insights, including:

- **Sentiment Analysis:** Determines the sentiment expressed in a given text, categorizing it as positive, neutral, or negative. This is useful for understanding customer feedback, product reviews, and social media posts.
- **Key Phrase Extraction:** Identifies the main points or topics in a text, helping to summarize and understand large documents or sets of data.
- **Language Detection:** Detects the language of a given text, which is essential for multilingual applications and services.
- **Named Entity Recognition (NER):** Identifies and categorizes entities such as names, organizations, locations, and dates within the text, providing structured information from unstructured data.

4.3.2 Why Azure Text Analytics?

- **Integration:** Being an Azure service, it integrates very easily with the rest of our system, including the components it's communicating with (Azure cloud function, Azure table storage).
- **Batch processing:** Azure Text Analytics supports batch-processing, which means we can send it multiple reviews at once. This is ideal for our use-case where we sometimes need to process tens of reviews at once.

4.3.3 Implementation details

- **Sentiment Analysis:** The cloud function sends collected reviews to the Azure Text Analytics API, which analyzes the sentiment of each review.

content	sentiment
El cambio es bueno	positive
El cambio es bueno...	positive
Adding me account	neutral
<pre> **Luv it.** Thanks so much for ping me on, @JulesGM. I have now learned many things. I have learned about [the appropriately named OmegaConf] (https://omegaconf.readthedocs.io), which I now realize I have been waiting for my entire life. I have also learned that @patrick-kidger of [jaxtyping] (https://docs.kidger.site/jaxtyping/api) fame is still right about everything. Interestingly, 'jaxtyping' offers a [similar runtime type-checker-agnostic '@jaxtyping.jaxtyped(typechecker=beartype.beartype) API](https://docs.kidger.site/jaxtyping/api/runtime-type-checking/) to that of the proposed 'hydra.utils.instantiate(cfg, target_wrapper=beartype.beartype) API: "python # Import both the annotation and the 'jaxtyped' decorator from 'jaxtyping' from jaxtyping import Array, Float, jaxtyped # Use your favourite typechecker: usually one of the two lines below. from typeguard import typechecked as typechecker from beartype import beartype as typechecker # Type-check a function @jaxtyped(typechecker=typechecker) def batch_outer_product(x: Float[Array, "b c1"], y: Float[Array, "b c2"]) -> Float[Array, "b c1 c2"]: return x[:, :, None] * y[:, None, :]" So for so good. You're in good company, @JulesGM. But I've been wondering... can we eventually simplify and streamline the process of selecting competing runtime type-checkers "or" is literally every Python package ever going to now define its own non-orthogonal proprietary API for selecting competing runtime type-checkers? It's the latter, isn't it? I'm sighing. You can almost feel the hot fetid breath I'm exhaling all over your keyboard. 'face_exhaling: If we accept the current status quo and do nothing, <sup>— "what will happen, guaranteed"</sup> then user headaches explode combinatorially. Currently, users have to manually notify "every" Python package of their preferred runtime type-checker with an API unique to that package. Instead, users should be able to trivially, publicly, and globally notify "every" Python package of their preferred runtime type-checker all-at-once simultaneously with just a single Python statement. Instead, we're now facing the exact opposite scenario. "Introducing..." ## 'anytype': Utopia Never Seemed So Far Away I'd make 'anytype'. But I can barely make @beartype. The core conceit is simple, though: it's [QtPy](https://github.com/spyder-ide/QtPy), but for runtime type-checking. In a nutshell, 'anytype' would: " Be a "thin high-level abstraction layer" over lower-level runtime type-checkers. " Only support features "commonly supported" by "all" runtime type-checkers. " Provide a "uniform API" for performing runtime type-checking. " Provide a "configuration API" for selecting between supported runtime type-checkers. " "Internally delegate" all runtime type-checking to the currently configured type-checker. Downstream packages like Hydra and 'jaxtyping' would then simply import 'anytype' and use that high-level API "without" concern for the currently configured type-checker. For example, the '@anytype.anytype' decorator would be a shim for either the '@beartype.beartype' "or" comparable '@typeguard.typechecked' decorators: e.g., "python from anytype import anytype # <-- type-checker-agnostic decorator for the win from dataclasses import dataclass @anytype # <-- you just won the internet. congrats @dataclass class ExampleDataclass: name: str unit_price: float quantity_on_hand: int = 0 inst = ExampleDataclass("a name", 123.32, 23) # Is ok inst = ExampleDataclass(123, None, 23) # Breaks "" Downstream users and apps would then also import and configure 'anytype' to use their preferred runtime type-checker. For example, the public 'anytype.configure()' function might accept a 'checker' parameter whose value is an 'AnytypeChecker' enumeration member identifying the desired runtime type-checker: e.g., "python # Probably in the "(your_app)_" subpackage to ensure this happens early: from anytype import AnytypeChecker, configure configure(checker=AnytypeChecker.beartype) # <-- feel the hot claws as they rake your codebase "" Of course, nobody has time, energy, inclination, money, or sufficient goodwill towards humanity. Nobody will ever do that. I am nobody, too. Still, a utopian dreamer dreams. If not here on GitHub, then where? Nordic Gods above, where! ?![(https://media1.tenor.com/m/rNCd8EqBKjoAAAAA/sad-cat.gif) <sup>i am very tired and must now lie on a door</sup> ## Would You Like to Know More About Managed Democracy and 'anytype'? If so, [hit this feature request at the @beartype issue tracker] (https://github.com/beartype/beartype/issues/355). Let's collaborate and listen. Let someone else do this for all of us. </pre>	

Figure 4.6: Sample Text Analytics output

Figure 4.6 shows a sample output of the Text Analytics API. The content section contains sample reviews collected from an open-source project, and the sentiment column contains the result returned by the sentiment analyser.

- **Prioritizing Negative Reviews:** Reviews identified as negative are given more importance by our GPT Agent during its review generation process. This ensures that the agent focuses on critical feedback, avoiding the emphasis on positive, generic reviews that simply compliment the users for their work.
- **Shortening responses:** The cloud function modifies the format of the data to compress it as much as possible while keeping it readable for the GPT Agent. Thus, the enriched, outputted data is half as voluminous as the data ingested by the function. This is crucial to ensure the respect of GPT's maximum number of tokens per request.

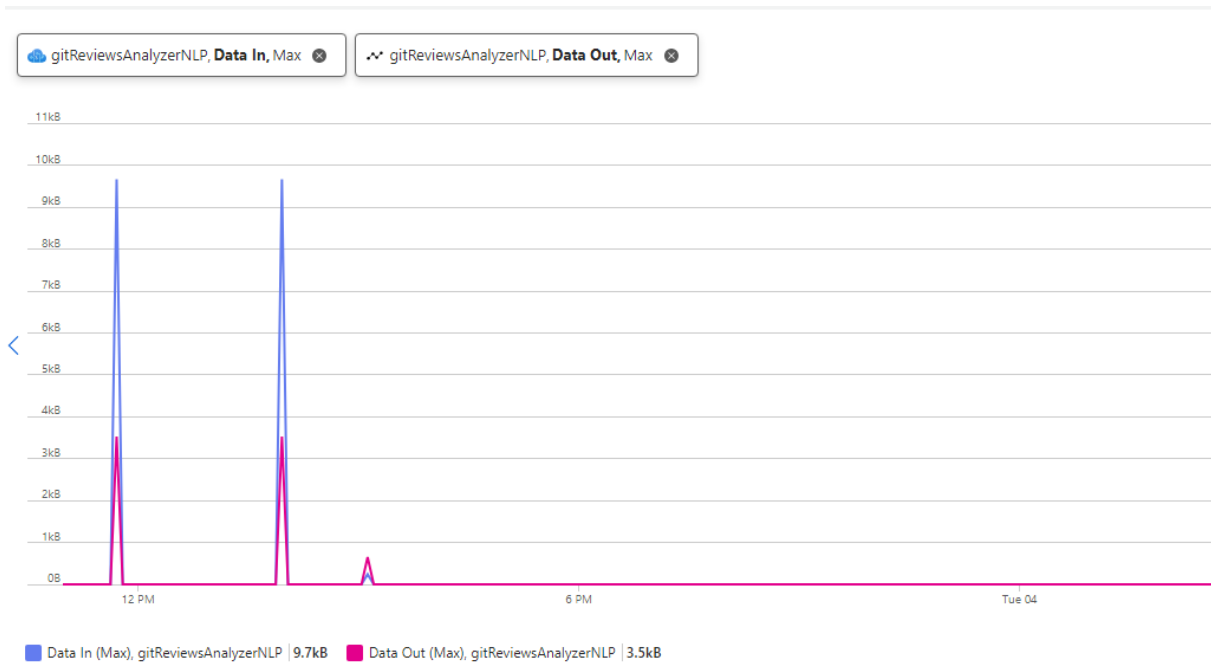


Figure 4.7: Data/in (blue) Data/out (purple), in KB

Figure 4.7 shows a comparison of the size of the input data ingested by the cloud function with the final, processed response size in KB.

- **CI/CD:** The cloud function instance on Azure is linked to a Github repository that contains the function's code. When a new commit occurs on the main branch of that repository, a Github action is triggered to automatically build and deploy the new version of the code.

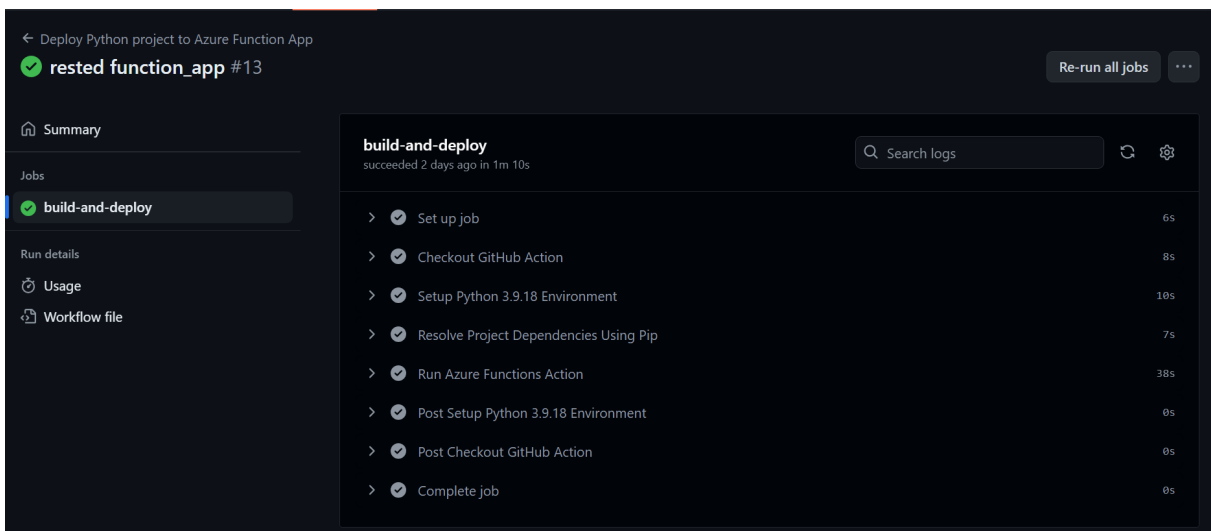


Figure 4.8: CI/CD pipeline steps

Figure 4.9 shows the various steps of our CI/CD pipeline, which consists in copying the code, setting up the python environment and installing dependencies, deploying the function to Azure, and finally cleaning up the job.

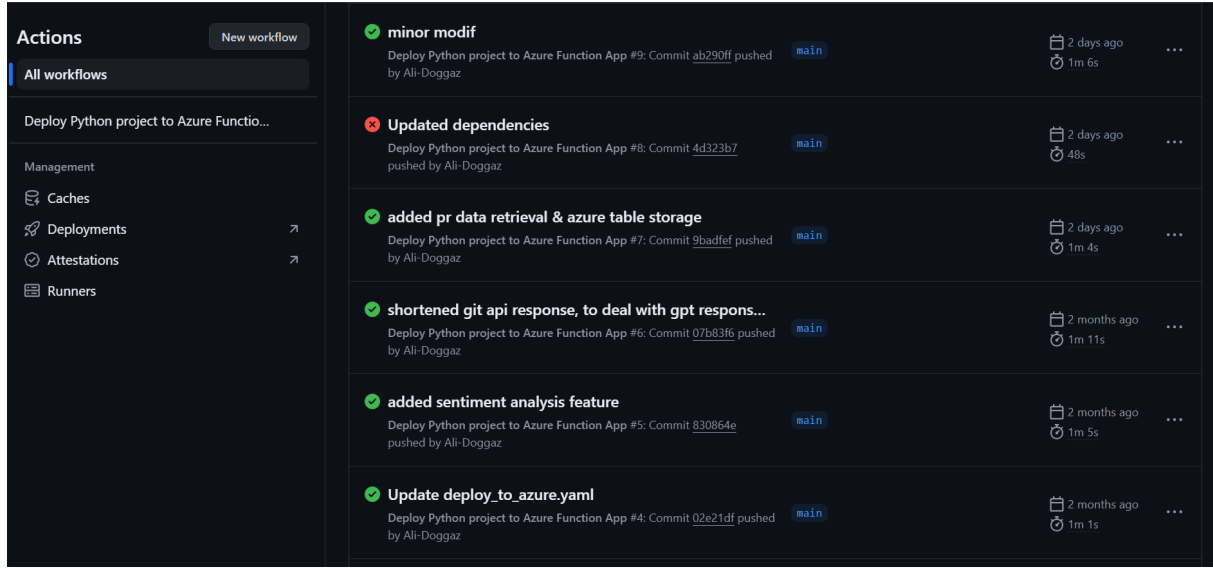


Figure 4.9: CI/CD pipelines executions

Figure 4.9 shows 6 executions of the cloud function’s CI/CD pipeline on GitHub Actions, their duration, and their states (failed/succeeded).

4.4 Data Lake: Azure Table Storage[6]

In this section, we will explain the reasons that led to opt for Azure Table Storage as our data lake storage solution and how we managed to implement and integrate this service to the rest of our project.

4.4.1 What is Azure Table Storage

Azure Table Storage is a cloud-based NoSQL data storage service that offers high availability, scalability, and flexibility for storing structured data in a schema-less design. It is designed to handle large volumes of data and provides fast access to data through a key/attribute store with a schema-less design. This makes it an ideal choice for scenarios where the data model evolves over time and for applications that require high-speed reads and writes, which is exactly our case.

4.4.2 Why Azure Table Storage?

Integration with Our Project: As part of the Azure ecosystem, Azure Table Storage integrates easily with the other Azure services used in our project such as Azure Functions, Azure Data Factory, and Azure Databricks. This integration simplifies data flow and processing within our architecture, and reduces costs as all data transfers between services will be classified as ingress. It also offers various advantages such as:

Flexibility: Azure Table Storage’s schema-less design allows us to store data in a flexible manner, accommodating changes in data structure over time without requiring major modifications.

Scalability: It can handle large volumes of data, which is crucial for our project as we process and store extensive amounts of data from GitHub and sentiment analysis results.

Cost Efficiency: Table storage offers a cost-effective solution for storing large amounts of data, making it suitable for being used as a data lake.

Security and Compliance: Azure Table Storage ensures data security with robust authentication and authorization mechanisms, including rotating secret keys managed by Azure Key Vault.

4.4.3 Implementation details

We've created a storage account to which we linked an Azure Table Storage entity. The cloud function uses it to flexibly store the pre-processed reviews. The current schema of the table, which can flexibly change if we decide to add fields to our data to enrich it further, is depicted in the figure below:

<input checked="" type="checkbox"/>	PartitionKey (can't be deleted or moved)	▼	
<input type="checkbox"/>	RowKey	▼	🗑️
<input type="checkbox"/>	Timestamp	▼	🗑️
<input type="checkbox"/>	author_type	▼	🗑️
<input type="checkbox"/>	author_association	▼	🗑️
<input type="checkbox"/>	content	▼	🗑️
<input type="checkbox"/>	sentiment	▼	🗑️
<input type="checkbox"/>	diff_hunk	▼	🗑️

Figure 4.10: Table Schema

Figure 4.10 shows the structure of our Azure Table Storage. The Author_type reflects whether the user is a human or a bot. Author_association informs us about whether the author is a maintainer of the project or not. Finally, and most importantly, the content column contains the comment itself, the sentiment contains the output of the sentiment analyser, and the diff_hunk column represents the code change related to that specific review.

PartitionKey	RowKey	Timestamp	author_type
507137	6632c390-ddc9-4ccc-ad...	2024-06-03T10:40:46.88...	User
507137	84a34cd8-80b4-4907-9...	2024-06-03T10:40:56.32...	User
507137	bdbad973-392f-4f93-93...	2024-06-03T12:58:21.88...	User
507137	ef4699ec-2f2c-4d82-a6c...	2024-06-03T10:40:55.45...	User
62310815	0ab66f18-5575-4353-a7...	2024-05-28T18:06:01.95...	Bot
62310815	176ca95c-9f6f-4392-ad...	2024-05-28T18:06:37.61...	Bot
62310815	561fd6de-25c4-4068-87...	2024-06-03T10:40:39.40...	Bot
62310815	59c70d2b-22ec-41ea-9e...	2024-05-28T18:06:01.80...	Bot
62310815	5c9a1344-a7ef-4c8b-9b...	2024-06-03T12:58:17.95...	Bot
62310815	7c401f9e-4460-48b3-96...	2024-06-03T10:40:39.57...	Bot
62310815	b8176821-f771-4b8b-8...	2024-05-28T18:06:37.74...	Bot
62310815	bc4f37ae-4e32-4673-a0...	2024-06-03T12:58:17.93...	Bot
8935917	28f39fc5-6d88-4a05-93f...	2024-06-03T12:58:21.88...	User

Figure 4.11: Sample Data, part 1

Figure 4.11 shows a projection of the Partition key, the row key, the timestamp, and the author_type columns, extracted from 13 sample rows stored on our Azure Table Storage database.

author_association	content	sentiment ↑	diff_hunk
COLLABORATOR	> > that it could be imp...	neutral	...
COLLABORATOR	Got it, thanks! Sorry it e...	neutral	@@ -46,6 +46,7 @@ de...
COLLABORATOR	Can you briefly explain ...	neutral	@@ -46,6 +46,7 @@ de...
COLLABORATOR	Can you briefly explain ...	neutral	@@ -46,6 +46,7 @@ de...
NONE	## Statement has no eff...	neutral	@@ -37,3 +37,6 @@ cl...
NONE	## Variable defined mul...	neutral	@@ -190,6 +191,12 @...
NONE	## Variable defined mul...	neutral	@@ -190,6 +191,12 @...
NONE	## Variable defined mul...	neutral	@@ -190,6 +191,12 @...
NONE	## Statement has no eff...	neutral	@@ -37,3 +37,6 @@ cl...
NONE	## Statement has no eff...	neutral	@@ -37,3 +37,6 @@ cl...
NONE	## Statement has no eff...	neutral	@@ -37,3 +37,6 @@ cl...
NONE	## Variable defined mul...	neutral	@@ -190,6 +191,12 @...
COLLABORATOR	> would it be possible t...	neutral	...

Figure 4.12: Sample Data, part 2

Figure 4.12 shows a projection of the author_association, content, sentiment, and diff_hunk columns, extracted from 13 sample rows stored on our Azure Table Storage database.

4.5 Azure Data Factory

Azure Data Factory (ADF) plays a critical role in our project's data processing architecture. It is responsible for orchestrating the data movement and transformation processes that are essential for preparing data for analysis and machine learning. This subsection will provide an overview of what Azure Data Factory is, why it was chosen for our project, and how it integrates with our overall system.

4.5.1 What is Azure Data Factory?

ADF is a cloud-based data integration service that allows you to create data-driven workflows for orchestrating and automating data movement and data transformation. With ADF, you can create and schedule data-driven workflows (called pipelines) that can ingest data from disparate data stores. These data flows can then be transformed and loaded into a centralized data store for further analysis.

4.5.2 Why Azure Data Factory?

- **Integration:** Azure Data Factory integrates seamlessly with other Azure services such as Azure Synapse Analytics, Azure Databricks, and Azure Storage, providing a unified data platform.
- **Scalability:** It can handle large volumes of data, making it suitable for complex ETL processes and large-scale data integration projects.
- **Automation:** ADF supports scheduling and triggers, allowing you to automate data workflows. In this project, a trigger schedule is set to execute the pipeline every 6 hours.
- **Flexible Data Movement:** Supports various data movement activities and connectors, enabling easy data ingestion from multiple sources.
- **Cost-Effective:** Offers a pay-as-you-go pricing model, ensuring cost efficiency based on usage.

4.6 Data Warehouse Setup : Azure Synapse Analytics

Azure Synapse Analytics is a pivotal component in our project's data architecture, providing a powerful platform for data warehousing and big data analytics. This subsection will delve into what Azure Synapse Analytics is, the reasons for choosing it, and its integration within our system.

4.6.1 What is Azure Synapse Analytics?

Azure Synapse Analytics is an integrated analytics service that accelerates time to insight across data warehouses and big data systems. It brings together the best of SQL technologies used in enterprise data warehousing, Spark technologies used for big data, and Data Explorer for log and time series analytics. It enables a seamless and unified experience to ingest, prepare, manage, and serve data for immediate business intelligence and machine learning needs.

4.6.2 Why Azure Synapse Analytics?

- **Comprehensive Integration:** Azure Synapse integrates easily with other Azure services such as Azure Data Factory, Azure Databricks, and Azure Machine Learning, providing a cohesive environment for end-to-end data processing and analysis.
- **Scalability:** The service can handle large volumes of data and can scale according to the computational needs, making it suitable for extensive data operations and real-time analytics.
- **Cost Efficiency:** By offering both on-demand and provisioned resources, Azure Synapse allows for cost-effective data processing. It supports both serverless and dedicated SQL pools, giving flexibility in resource management.
- **Advanced Analytics:** With built-in capabilities for advanced analytics, it supports various data processing workloads, from data warehousing to big data analytics, ensuring comprehensive insights.
- **Security and Compliance:** Azure Synapse ensures robust data security with features like dynamic data masking, threat detection, and advanced encryption, making it compliant with industry standards.

4.6.3 Implementation Details

To implement the data pipeline using Azure Synapse Analytics, the following steps were taken:

- **Linked Services:** Set up linked services in Azure Data Factory to connect to the SQL database and Azure Synapse Analytics.

Linked services

Linked services are much like connection strings, which define the connection information needed for Azure Synapse Analytics to connect to data sources. [Learn more](#)

+ New

Filter by name

Annotations : Any

Showing 1 - 5 of 5 items






Name	Type	Related
 AzureSqlGitReviewsLake	Azure SQL Database	2
 AzureSynapseAnalytics1	Azure Synapse Analytics	0
 AzureSynapseAnalytics2	Azure Synapse Analytics	1
 gitreviewsetlworkplace-WorkspaceDefault...	Azure Synapse Analytics	0
 gitreviewsetlworkplace-WorkspaceDefault...	Azure Data Lake Storage Gen2	1

Figure 4.13: List of the linked services

Figure 4.13 shows the list of the linked services we used in order to implement the ETL pipeline inside ADF.

- **Dataflows:** Created a dataflow pipeline to extract data from the SQL database, transform it to retain only the necessary columns (content, diff_hunk, sentiment), and load it into Azure Synapse Analytics.

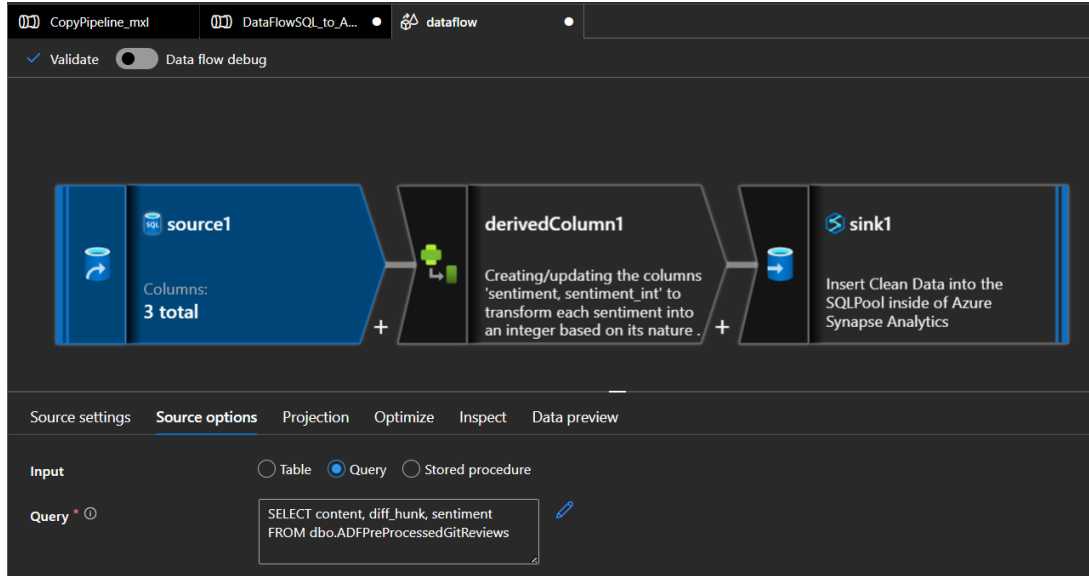
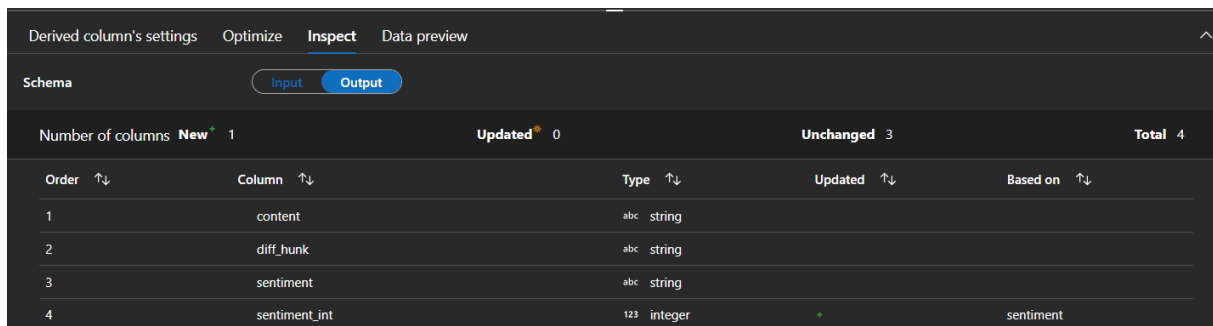


Figure 4.14: DataFlow Pipeline Architecture

Figure 4.14 shows the steps of our dataflow pipeline which will handle the ETL process inside the ADF studio where we extract the reviews from the Azure data lake represented here with source1 , transform it into clean data keeping only the columns we will be needing , and then loading it into the sink1 which represents our Azure Synapse Analytics SQL Pool ADF.

- **Transformation Logic:** Applied transformation logic within the dataflows to convert sentiment values from strings to integers (positive: 1, neutral: 2, negative: 0) for better analysis and storage efficiency.



Order	Column	Type	Updated	Based on
1	content	string		
2	diff_hunk	string		
3	sentiment	string		
4	sentiment_int	integer		sentiment

Figure 4.15: The Output of the transformation logic

Figure 4.15 shows the output of the transformation phase of our dataflow ETL where we only keep the needed columns which are content, diff_hunk, sentiment, and sentiment_int which were later passed to the sink db in our azure synapse analytics ADF.

- **Data Loading:** Configured the sink in the dataflow to load the transformed data into a dedicated SQL pool within Azure Synapse Analytics, ensuring the data is prepared for continuous training pipelines.
- **Automation:** Set up a trigger schedule in Azure Data Factory to execute the pipeline every 6 hours, ensuring the data in Azure Synapse Analytics is regularly updated.

4.7 Model fine-tuning with Azure Databricks

4.7.1 What is Azure Databricks?

Azure Databricks[7] is a fully managed first-party service that enables an open data lakehouse in Azure. With a lakehouse built on top of an open data lake, quickly light up a variety of analytical workloads while allowing for common governance across your entire data estate. Enable key use cases including data science, data engineering, machine learning, AI, and SQL-based analytics.

4.7.2 Why Azure Databricks?

Azure Databricks provides several compelling reasons for its adoption:

- **Scalability:** With built-in support for Apache Spark, Azure Databricks can handle large-scale data processing and analytics workloads, scaling up or down based on demand.
- **Integrated Services:** Azure Databricks seamlessly integrates with other Azure services, such as Azure Synapse Analytics, Azure Machine Learning, and Azure Data Lake Storage, providing a comprehensive ecosystem for data analytics and AI development.
- **Performance:** Leveraging distributed computing capabilities, Azure Databricks delivers high-performance analytics and machine learning capabilities, enabling faster insights and model training.
- **Productivity:** The interactive notebook interface of Azure Databricks allows users to write and execute code collaboratively, promoting productivity and experimentation.

4.7.3 Implementation details

In this part, we describe the steps taken to fine-tune the model using Azure Databricks.

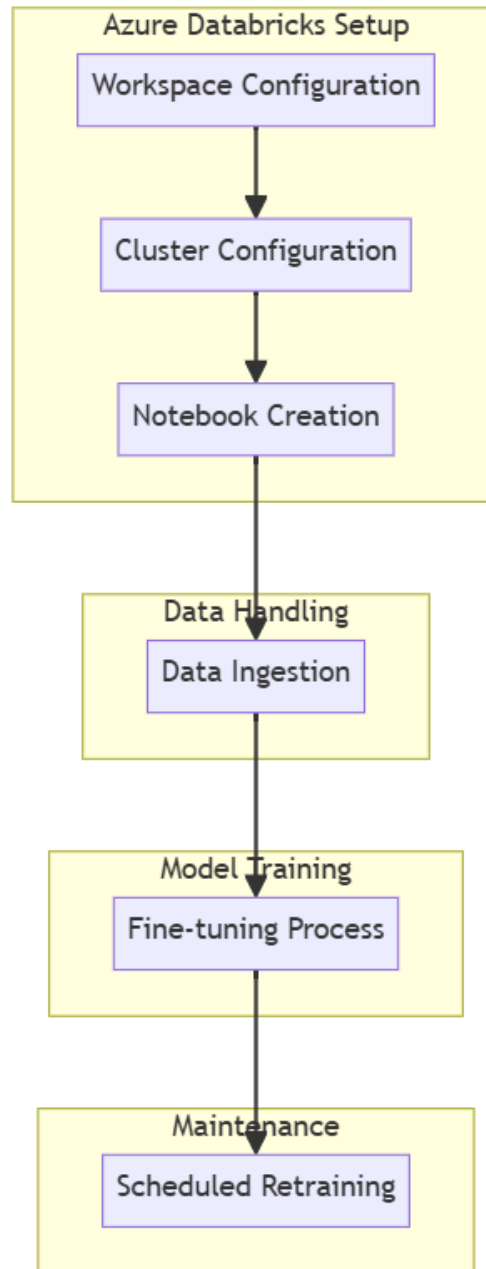


Figure 4.16: Flowchart diagram of the implementation steps for model fine-tuning

Figure 4.16 represents a flowchart diagram illustrating the process of fine-tuning a model with Azure Databricks, covering setup, data handling, model training, and maintenance.

To begin, an Azure Databricks workspace was set up to facilitate the fine-tuning of our model, providing a collaborative environment that integrates seamlessly with other Azure services. Next, a cluster was configured within the Azure Databricks workspace to provide the necessary computing resources for model training and fine-tuning. Additionally, a notebook was created within Azure Databricks to manage the fine-tuning workflow, allowing for interactive code execution and documentation.

Data Ingestion:

Data was ingested into the Azure Databricks environment from Azure Synapse Analytics. This data included new pull request data, such as code changes, review comments, and sentiment analysis results.

content	diff_hunk	sentiment	sentiment_int
Hi @minato-ellie!...	NULL	positive	1
## Statement has ... @@ -37,3 +37,6 @@...		neutral	2
Hi @gerard-sanrom...	NULL	mixed	3
CI failing due to...	NULL	negative	0
Thank you for sig...	NULL	positive	1
## Variable defin... @@ -190,6 +191,12...		neutral	2
Hi @gerard-sanrom...	NULL	mixed	3
CI failing due to...	NULL	negative	0
Hi @minato-ellie!...	NULL	positive	1
## Variable defin... @@ -190,6 +191,12...		neutral	2
Hi @JulesGM! \n\n...	NULL	mixed	3
CI failing due to...	NULL	negative	0
Thank you for sig...	NULL	positive	1

Figure 4.17: Example of ingested data including.

Figure 4.17 above illustrates a snippet of the ingested data, showcasing various aspects of pull request data including code changes and review comments. This data serves the foundation for fine-tuning our model within the Azure Databricks environment.

Fine-tuning Process:

The core of the implementation involved fine-tuning a pre-trained model using the ingested data. This process included loading the model, adjusting hyperparameters, and training the model on the domain-specific data. **Fine-tuning Process:**

The core of the implementation involved fine-tuning a pre-trained model using the ingested data. This process included loading the model, adjusting hyperparameters, and training the model on the domain-specific data.

- **Model Selection:** In selecting the model architecture for our fine-tuning task, we opted for the GPT-2 model provided by Hugging Face's Transformers library. GPT-2, or "Generative Pre-trained Transformer 2," stood out due to its impressive performance in generating coherent and contextually relevant text. Given our objective of predicting reviews, GPT-2's transformer-based architecture and self-attention mechanism made it an ideal choice. Moreover, its pre-training on a vast corpus of text data enabled us to leverage its language understanding capabilities for our specific domain.

- **Tokenization:** To preprocess our input data for fine-tuning, we employed the GPT-2 tokenizer from Hugging Face. This tokenizer facilitated the conversion of raw text into numerical tokens, which the model can comprehend. By utilizing the tokenizer's functionality, we ensured that our input data was properly formatted and compatible with the GPT-2 model's requirements.
- **Model Training:** The model training phase involved initializing the Trainer object with the chosen model, training arguments, and prepared dataset. Leveraging the Trainer object's capabilities, we executed the training loop, which encompassed forward and backward passes, optimization, and evaluation.
- **Evaluation:** Evaluation of the fine-tuned model's performance was conducted using several key metrics, including perplexity, BLEU score, and ROUGE score. **Perplexity**[8] quantifies the model's ability to predict the data, with lower values indicating better performance. **BLEU** [9] score assesses the quality of generated text by comparing it to reference text, essential for evaluating the model's accuracy in generating review predictions. **ROUGE** [10] score measures the overlap between generated and reference text, providing insights into the model's capability to produce relevant review content. By assessing these metrics, we evaluated the effectiveness of our fine-tuned GPT-2 model in generating accurate and contextually relevant review predictions.

Scheduled Retraining:

To ensure that our model remains up-to-date and effectively captures evolving trends and patterns in the data, I implemented a scheduled retraining process. A job is scheduled to run every day at midnight using Databricks workflows. This automated process triggers the model training pipeline, allowing it to ingest new data and fine-tune the model accordingly.

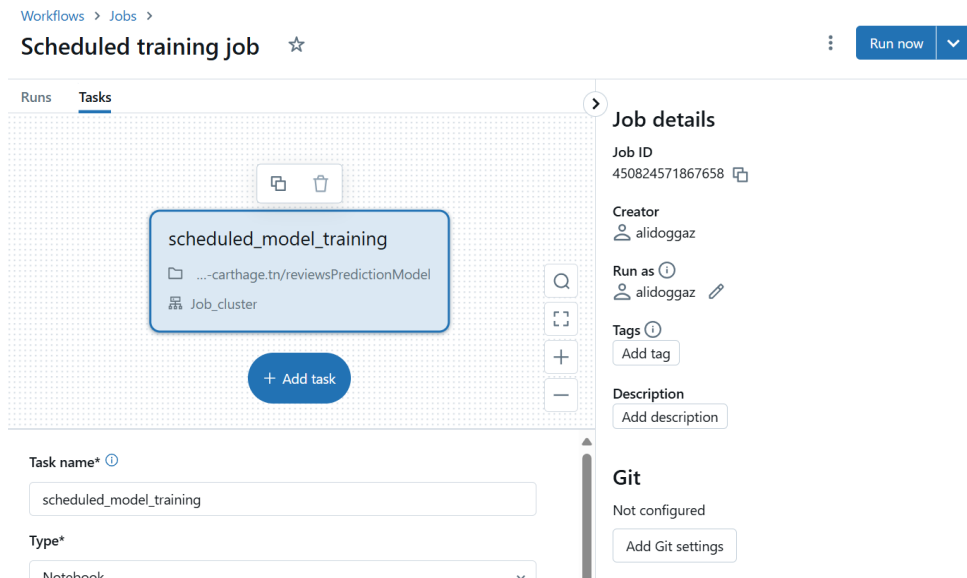


Figure 4.18: Scheduled retraining job in Databricks workflow.

Figure 4.18 below represents a screenshot of the scheduled retraining job "schedule model training" configured to run daily at midnight in Databricks workflow.

4.8 Model Versioning and Tracking with MLflow

In this section, we will discuss the use of MLflow in our project for managing the lifecycle of our machine learning models. We will explain what MLflow is, the reasons for choosing it, and how we have implemented it to ensure effective versioning and tracking of our models.

4.8.1 What is MLflow?

MLflow [11] is an open-source platform designed to manage the machine learning lifecycle, including experimentation, reproducibility, and deployment. It provides tools to track experiments, package code into reproducible runs, and share and deploy models. MLflow comprises four main components:

- **MLflow Tracking:** Records and queries experiments, including code, data, configurations, and results.
- **MLflow Projects:** Allows you to package data science code in a format that is reproducible and reusable.
- **MLflow Models:** Serves as a standard format for packaging machine learning models for diverse deployment tools.
- **MLflow Registry:** Provides a central repository to manage the lifecycle of MLflow models.

4.8.2 Why MLflow?

- **Adaptability:** During the implementation phase of the previous architecture (V1.0.0), we encountered a technical problem that prevented us from linking Azure Databricks to Azure Machine Learning. The latter required a "Service Principal Authentication" profile, which to be created, needed us to have organizational-level permissions to our organization. As we worked on this project using our university's emails, we would've had to contact the university to solve this issue. We preferred to slightly adapt our architecture and replaced Azure Machine Learning with MLflow to keep track of our experiments, along with each training's results and performances
- **Integration with Databricks:** MLflow integrates seamlessly with Azure Databricks, making it a natural choice for our workflow
- **Experiment Tracking:** MLflow Tracking allows us to log parameters, code versions, metrics, and output files when running our machine learning code, facilitating comprehensive tracking of model training and performance.
- **Reproducibility:** MLflow Projects help ensure that the experiments are reproducible and shareable across different environments.
- **Model Registry:** MLflow Registry provides a central repository to manage the full lifecycle of ML models, including versioning and stage transitions.

4.8.3 Implementation details

Our project extensively utilized MLflow for managing the lifecycle of our machine learning models. MLflow's tracking functionality enabled us to systematically log and monitor the progress of model training runs. By leveraging MLflow's user interface, we were able to visualize key metrics such as loss, perplexity, BLEU over the course of training. Additionally, MLflow's artifact logging feature allowed us to store the trained model artifacts alongside their associated metadata, including hyperparameters and performance metrics.

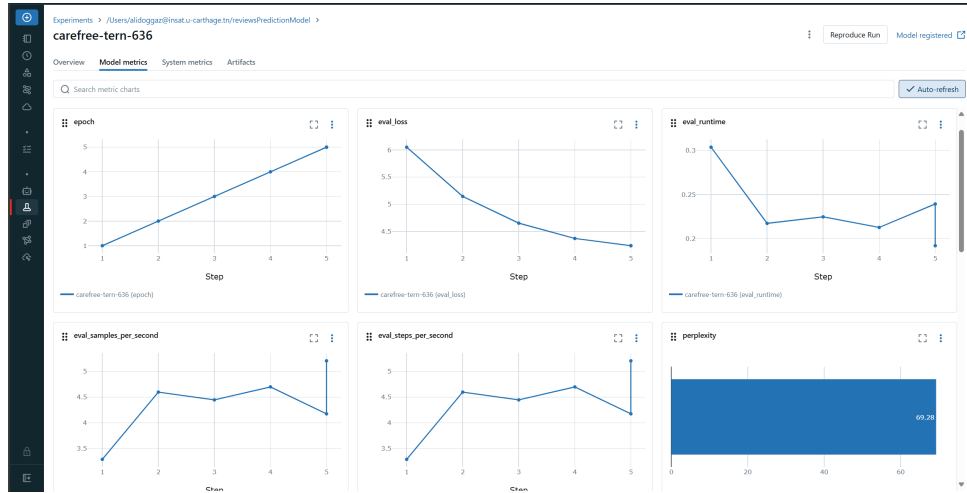


Figure 4.19: MLflow dashboard showing tracked metrics

Figure 4.19 is a screenshot of the MLflow dashboard, showcasing a comprehensive overview of tracked metrics during the model training lifecycle. It includes various plots such as epoch progression, evaluation loss, runtime, samples per second, and perplexity. MLflow's user interface facilitated easy monitoring and comparison of these metrics across different versions. The dashboard's ability to visualize these key metrics enabled us to make informed decisions regarding model performance and optimization.

Furthermore, MLflow's model registry provided a centralized repository for storing and versioning our trained models. This enabled us to easily compare different model versions, track improvements over time, and select the best-performing model for deployment. The ability to visualize and compare models directly within MLflow streamlined our decision-making process and facilitated collaboration among team members.

Experiments >

reviewsPredictionModel ⓘ Add Description

Q metrics.rmse < 1 and params.model = "tree" ⓘ Time created ▾ State: Active ▾ Datasets ▾ Sort: Created ▾ Columns ▾ ⓘ G

Table Chart Evaluation Preview

<input type="checkbox"/>	Run Name	Created	Dataset	Duration	Source	Models
<input type="checkbox"/>	carefree-tern-636	23 hours ago	-	1.1min	reviewsP...	GPT2ReviewPredictor v6
<input type="checkbox"/>	painted-wren-231	23 hours ago	-	1.7min	reviewsP...	GPT2ReviewPredictor v5
<input type="checkbox"/>	intrigued-shad-578	23 hours ago	-	1.4min	reviewsP...	GPT2ReviewPredictor v4
<input type="checkbox"/>	bustling-croc-892	23 hours ago	-	1.1min	reviewsP...	GPT2ReviewPredictor v3
<input type="checkbox"/>	valuable-mule-269	23 hours ago	-	1.2min	reviewsP...	GPT2ReviewPredictor v2

Figure 4.20: MLflow dashboard showcasing different stored model versions.

Figure 4.20 is a screenshot from the MLflow dashboard, displaying a comprehensive overview of tracked metrics and different versions of the model. The dashboard includes detailed metrics such as evaluation loss, perplexity, and other performance indicators for each version. This visualization aids in monitoring the progress of various model iterations and facilitates effective version control and comparison.

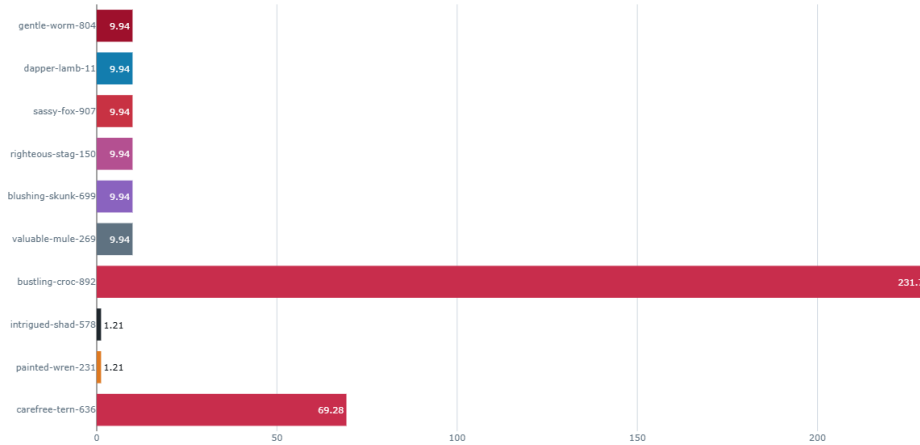


Figure 4.21: A bar chart showing the perplexity scores for various model versions.

Figure 4.21 is a bar chart depicting the perplexity scores for various model training runs. Perplexity, a common metric in language modeling, indicates how well the model predicts a sample. Lower perplexity signifies better performance. The chart, generated using MLflow's tracking interface, highlights significant differences in perplexity among the models. Notably, one model ('bustling-croc-892') has a substantially higher perplexity, indicating poor performance compared to others. This visualization aids in quickly identifying underperforming models and guiding further optimization efforts.

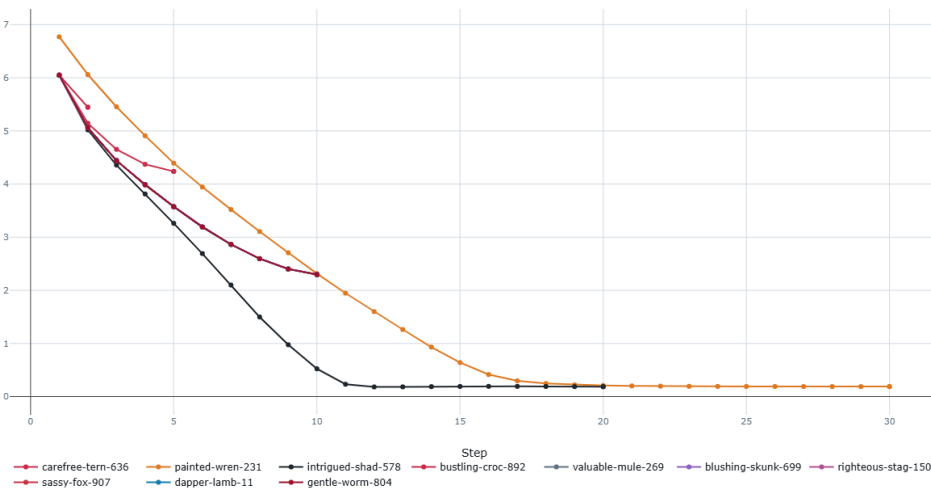


Figure 4.22: The evaluation loss for various model training runs over a series of steps.

Figure 4.22 displays the evaluation loss of various model training runs over a series of steps. Utilizing MLflow's tracking functionality, we were able to systematically monitor the decline in evaluation loss, which is indicative of the model's learning progress. The

image shows distinct curves for each run, showcasing the performance variability across different hyperparameter configurations. This visualization is crucial for assessing which model configurations are most effective at minimizing loss, thereby improving model accuracy.

4.9 Azure Key Vault[12] and Azure Monitoring[13]

In this section, we will provide a brief explanation of Azure Key Vault and Azure Monitoring and will explain how we leverage them in our entire project to secure the services communication and monitor their usage.

4.9.1 What are Azure Key Vault and Azure Monitoring?

Azure Key Vault is a cloud service that securely stores and manages sensitive information such as API keys, passwords, certificates, and other secrets. In our project, Azure Key Vault is used to store and automatically rotate the API key for the Azure Cloud Function, ensuring secure and efficient access control across all Azure services.

Azure Monitoring provides comprehensive tools to monitor and analyze the performance of various Azure services. In our project, Azure Monitoring is employed to oversee all data pipelines (Azure Databricks, Azure Table Storage, Azure Synapse Analytics) and other services, including the Azure Cloud Function. This centralized monitoring ensures that we can track the health, performance, and usage of all components in our system, facilitating prompt identification and resolution of any issues.

4.9.2 Why Azure Key Vault and Azure Monitoring?

- **Straightforward integration in the Azure ecosystem:** These two services are available in all Azure tools, and integrating them to an Azure-based project like ours requires minimal effort.
- **Key Management:** Azure Key Vault stores and rotates the API key for the Cloud Function, enhancing security and reducing manual key management tasks.
- **Monitoring and Alerts:** Azure Monitoring tracks the performance of data pipelines and services, providing real-time insights and alerts to maintain optimal operation and quickly address any anomalies.

Note: The entire implementation of these services with the other Azure tools we're using is entirely handled by the cloud-provider.

Conclusion:

This chapter detailed the implementation of our machine learning pipeline using Azure services and MLflow.

We automated data preparation using Azure Data Factory to maintain a continuous data flow into Azure Synapse Analytics. Azure Databricks was used for model fine-tuning due to its scalability, performance, and integration capabilities. The process involved setting up a workspace, configuring clusters, ingesting data, and training the model using

GPT-2 from Hugging Face. Evaluation metrics like perplexity, BLEU, and ROUGE scores assessed model performance.

Scheduled retraining in Databricks ensured the model stayed current with new data. MLflow was used for tracking, monitoring, and versioning models, integrating seamlessly with Databricks and providing a comprehensive framework for managing the machine learning lifecycle.

In summary, our implementation leverages Azure and MLflow to create a robust, scalable, and efficient machine-learning pipeline.

5 Challenges and Limitations

This project aimed to develop a robust, scalable, and efficient data pipeline for processing and analyzing code review data from GitHub. Despite the successful implementation, we encountered several technical challenges and limitations that required careful consideration and innovative solutions. This section outlines the primary technical challenges faced, limitations of the current solution, and potential future work and improvements to address these issues.

5.1 Technical Challenges

The development and deployment of our project involved integrating several advanced technologies and platforms, which presented a number of technical challenges. These challenges had to be addressed to ensure the system's efficiency, reliability, and scalability.

5.1.1 Integration with Existing Systems

Integrating various Azure services such as Azure Data Factory, Azure Databricks, and Azure Synapse Analytics was a significant technical challenge. Each service has its own configuration requirements and limitations, necessitating a deep understanding of their interoperability. Ensuring seamless data flow required meticulous configuration and extensive testing to confirm that each service communicated effectively without data loss or corruption.

5.1.2 Scalability Issues

As the project scaled, managing the increasing volume of data posed challenges. The architecture using Azure Table Storage and Azure Synapse Analytics had to be optimized continuously to handle large datasets while maintaining performance. This included optimizing query performance, ensuring efficient data partitioning, and allocating sufficient computational resources to avoid bottlenecks.

5.1.3 Model Training and Updating

Setting up a continuous machine learning (ML) training pipeline that regularly trains new models with fresh data was complex. Automating this process required precise orchestration to ensure models remain relevant and accurate. Challenges included scheduling training jobs, managing computational loads, and ensuring robust validation of model performance after each training cycle.

5.1.4 Security and Compliance

Ensuring data security and compliance with regulatory requirements was a critical concern. Implementing robust authentication and authorization mechanisms, such as Azure Active Directory for identity management and Azure Key Vault for managing secrets, added complexity. Continuous monitoring and updating security measures to meet compliance standards were essential to protect sensitive data.

5.2 Limitations of the Current Solution

While the implemented solution successfully meets the project's objectives, there are certain limitations that need to be addressed to enhance its performance and scalability. These limitations highlight areas for future improvements.

5.2.1 Git API Rate Limits

We created a GitHub organization and used a single token for API calls. This approach could lead to rate limit issues when multiple users access the system simultaneously. To address this, we should implement OAuth authentication so each user connects to GitHub with their own credentials, allowing us to use individual tokens and distribute the API call load more effectively.

5.2.2 Sentiment Analysis Accuracy

The sentiment analysis model often classifies reviews as neutral even when they lean towards the negative side. This limitation means we might miss critical feedback. To mitigate this, we could treat neutral comments with similar scrutiny as negative ones, especially in contexts where neutral might imply dissatisfaction or concern.

5.2.3 ETL Pipeline Limitations

Implementing the data flow for the ETL pipeline from the SQL database in Azure Storage to Azure Synapse Analytics (ASA) had constraints. Specifically, we were limited to using a dedicated SQL pool in ASA, which is resource-intensive compared to a serverless SQL pool. Additionally, minor bugs in Azure services sometimes caused delays in troubleshooting pipelines when the root cause was an Azure bug, not our configuration.

5.2.4 Limited Training Data

The training pipeline in Databricks faced challenges due to limited data availability. Since the solution has not been deployed yet, the training algorithms have limited reviews to learn from, impacting their accuracy and ability to generalize to future data. Increasing the volume of training data post-deployment will be crucial for improving model performance.

5.3 Future Work and Improvements

The following future work and improvements are proposed to address the current limitations and enhance the project's overall effectiveness, scalability, and accuracy.

5.3.1 Enhanced Integration with Azure Machine Learning

Resolving authentication issues to enable full integration with Azure Machine Learning will enhance version control, model evaluation, and management capabilities, making the ML pipeline more robust and maintainable.

5.3.2 Improved Scalability Solutions

To handle increasing data volumes more efficiently, we should explore advanced scalability solutions, such as sharding techniques or more scalable storage solutions like Azure Cosmos DB. This would help maintain performance as data grows.

5.3.3 Advanced Security Measures

Implementing more advanced security protocols, including enhanced encryption and granular access controls, will further ensure data security and regulatory compliance. Regular security audits will help identify and mitigate potential vulnerabilities.

5.3.4 User Feedback Integration

Actively seeking and incorporating user feedback will help refine the system to better meet the specific needs of different projects. This iterative process will improve the system's effectiveness and user satisfaction.

5.3.5 Expanding Beyond Azure

Reducing dependency on a single cloud provider by exploring compatibility with other cloud platforms will enhance flexibility and resilience. Implementing a multi-cloud strategy will leverage the best features of different providers and provide redundancy against provider-specific issues.

Conclusion:

By addressing these specific challenges and limitations, and focusing on future improvements, the project can evolve into a more robust, scalable, and user-friendly solution for automated code reviews. This approach ensures the system not only meets current needs but is also prepared to handle future demands and complexities.

6 Conclusion

6.1 Summary of Findings

Our project aimed to integrate AI into the code review process to enhance efficiency and accuracy. We implemented a comprehensive architecture leveraging Azure services such as Azure Functions, Azure Data Factory, Azure Synapse Analytics, and Azure Databricks. Throughout the project, we managed to create a robust system that performs sentiment analysis on code reviews, processes data effectively, and ensures secure storage and management of the information. The iterative Agile methodology facilitated continuous improvement and adaptation to user feedback, ensuring that the final solution meets the needs of developers and contributors.

6.2 Project Outcomes

The project successfully demonstrated the feasibility and benefits of integrating AI into the code review process. Key outcomes include:

- Developed a scalable and flexible architecture using Azure services.
- Implemented sentiment analysis to provide insights into code reviews.
- Established a secure and efficient data processing pipeline.
- Achieved continuous improvement through Agile sprints and user feedback.
- Created a foundation for future enhancements and scalability.

These outcomes highlight the potential for AI to significantly improve code review processes, making them more efficient and insightful.

6.3 Final Thoughts

The integration of AI into code review processes marks a significant advancement in software development practices. By leveraging AI, we can reduce the manual effort involved in code reviews, identify issues more accurately, and provide developers with valuable insights. Our project serves as a proof of concept that demonstrates the potential benefits and sets the stage for future improvements and scalability. Continued development and refinement, along with user feedback, will be crucial in fully realizing the potential of AI in this domain. We are optimistic about the future possibilities and the impact this project can have on the software development community.

Bibliography

References

- [1] Amazon. *Data Lake*. Accessed: 2024-06-05. 2024. URL: <https://aws.amazon.com/what-is/data-lake/>.
- [2] Microsoft. *Data Warehouse*. Accessed: 2024-06-05. 2024. URL: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-data-warehouse>.
- [3] Turing. *Model fine-tuning*. Accessed: 2024-06-02. 2024. URL: <https://www.turing.com/resources/finetuning-large-language-models#what-is-fine-tuning,-and-why-do-you-need-it?>.
- [4] Microsoft. *Azure Cloud Functions*. Accessed: 2024-06-05. 2024. URL: <https://docs.microsoft.com/en-us/azure/azure-functions/>.
- [5] Microsoft. *Azure Text Analytics Documentation*. Accessed: 2024-06-05. 2024. URL: <https://docs.microsoft.com/en-us/azure/cognitive-services/text-analytics/>.
- [6] Microsoft. *Azure Table Storage Documentation*. Accessed: 2024-06-05. 2024. URL: <https://docs.microsoft.com/en-us/azure/storage/tables/>.
- [7] microsoft. *Azure Databricks*. Accessed: 2024-06-03. 2024. URL: <https://azure.microsoft.com/en-us/products/databricks>.
- [8] Priyanka DS. *Perplexity of Language Models*. Accessed: 2024-06-03. 2024. URL: <https://medium.com/@priyankads/perplexity-of-language-models-41160427ed72>.
- [9] Akash Gautam. *BLEU Evaluation of Large Language Models*. Accessed: 2024-06-03. 2024. URL: <https://www.linkedin.com/pulse/bleu-evaluation-large-language-models-part-2-akash-gautam-6uifc/>.
- [10] Hugging Face. *ROUGE Metric*. Accessed: 2024-06-03. 2024. URL: <https://huggingface.co/spaces/evaluate-metric/rouge>.
- [11] MLflow. *MLflow Documentation*. Accessed: 2024-06-03. 2024. URL: <https://mlflow.org/docs/latest/index.html>.
- [12] Microsoft. *Azure Key Vault Documentation*. Accessed: 2024-06-05. 2024. URL: <https://docs.microsoft.com/en-us/azure/key-vault/>.
- [13] Microsoft. *Azure Monitoring Documentation*. Accessed: 2024-06-05. 2024. URL: <https://docs.microsoft.com/en-us/azure/azure-monitor/>.