GitHub Repository Guide Agent

Project Report

submitted as part of

Winter in Data Science (WiDS) 2025

Submitted by

Yindukuri Sai Durga Esha
24B2119

WiDS 2025

1 Introduction

Open-source software development has become a cornerstone of modern software engineering, with GitHub serving as the primary platform for hosting, collaborating on, and maintaining software projects. Millions of repositories exist on GitHub, ranging from small personal projects to large-scale industrial systems. While this abundance of resources is beneficial, it also introduces a significant challenge: understanding an unfamiliar codebase efficiently.

For a new developer joining a project, the initial onboarding process can be time-consuming and overwhelming. Developers are often required to read lengthy README files, explore deeply nested directory structures, and manually identify important configuration files, source code modules, and entry points. In many cases, documentation may be incomplete, outdated, or written with the assumption that the reader already has prior context. This creates a steep learning curve, especially for beginners and contributors new to open-source development.

Recent advancements in Artificial Intelligence, particularly Large Language Models (LLMs), have demonstrated strong capabilities in understanding and summarizing natural language text. These models can extract key information, identify structure, and generate human-readable explanations from large amounts of unstructured data. When combined with programmatic access to repository metadata through the GitHub API, LLMs present an opportunity to significantly improve the developer onboarding experience.

This project, titled **GitHub Repository Guide Agent**, was developed as part of the Winter in Data Science (WiDS) 2025 program. The goal of the project is to design and implement an AI-powered assistant that helps users explore GitHub repositories by automatically generating a structured developer guide. The system fetches repository documentation and file structure, analyzes them using an LLM, and presents a clear, guided explanation of the project's purpose, organization, and important components.

Over the course of four weeks, the project evolved from a simple GitHub README fetcher into a fully integrated web-based application with an interactive user interface. Each week focused on a different aspect of the system, including GitHub API integration, AI-driven summarization, repository structure analysis, and frontend development using Streamlit. This report documents the learnings, experiments, challenges, and outcomes from the complete WiDS journey.

2 Problem Statement

Understanding an unfamiliar GitHub repository is a non-trivial task, especially for new contributors and students. Although most repositories include a README file, the quality, length, and clarity of these documents vary significantly. In many cases, README files are either too brief to provide meaningful guidance or excessively long without a clear structure, making it difficult for readers to extract key information quickly.

Additionally, repository codebases often contain numerous files and directories, including configuration files, scripts, modules, and documentation. Identifying which files are critical for understanding the project, where to begin reading the code, and how different components interact typically requires manual exploration. This process is time-consuming and relies heavily on prior experience with similar projects or technologies.

Existing solutions for repository exploration primarily depend on manual inspection through the GitHub interface or third-party documentation tools that require additional effort from repository maintainers. There is a lack of automated tools that can intelligently analyze both the documentation and structure of a repository to generate a coherent onboarding guide for developers.

The problem addressed in this project is the absence of an automated, intelligent system that can:

- Understand the purpose of a GitHub repository from its documentation,
- Analyze the repository structure to identify important files,
- Explain how the codebase is organized, and
- Provide a logical onboarding path for new developers.

Without such a system, developers—particularly beginners—face a steep learning curve when approaching new repositories. This project aims to bridge this gap by leveraging GitHub APIs and Large Language Models to automatically generate a structured, human-readable developer guide that simplifies repository exploration and onboarding.

3 Objectives of the Project

The primary objective of this project is to design and develop an AI-powered system that assists developers in understanding GitHub repositories quickly and effectively. The system aims to automate the process of repository exploration by analyzing both documentation and repository structure to generate a guided onboarding experience.

The specific objectives of the project are as follows:

- To fetch and process repository data, including README files and file structure, using the GitHub API.
- To utilize a Large Language Model (LLM) for interpreting repository documentation and generating meaningful explanations.
- To identify important files and directories within a repository that are critical for understanding the project.

- To generate a structured developer guide explaining the purpose, organization, and workflow of the repository.
- To provide a logical onboarding path for new contributors, indicating where to begin and how to navigate the codebase.
- To design a user-friendly interface that allows users to explore repositories without manual code inspection.
- To ensure modularity and extensibility so that additional features or models can be integrated in the future.

Through these objectives, the project seeks to reduce the cognitive load on developers, improve onboarding efficiency, and promote better understanding of open-source codebases.

4 Week-wise Learnings and Experiments

This section presents a detailed account of the learnings, experiments, and outcomes from each week of the WiDS 2025 program. The project evolved incrementally over four weeks, with each phase building upon the previous one to finally develop an AI-powered GitHub Repository Guide Agent.

4.1 Week 1: GitHub API Integration and README Fetching

The first week focused on understanding how to interact programmatically with GitHub repositories using the GitHub REST API. The primary goal was to fetch and display the README file of any public GitHub repository.

Key learnings from this week include:

- Understanding how GitHub repositories are structured and how data is exposed through the GitHub API.
- Using the `PyGithub` library to authenticate and communicate with GitHub.
- Working with Personal Access Tokens (PATs) and understanding secure credential handling using environment variables.
- Fetching repository metadata such as repository name, owner, and file contents.
- Decoding Base64-encoded README files returned by the API.
- Implementing basic error handling for scenarios such as invalid repository names, missing README files, and authentication errors.

By the end of Week 1, a functional Python script was developed that allowed users to input a repository name in `owner/repo` format and retrieve the README content directly in the terminal. This week laid the foundation for all future stages of the project.

4.2 Week 2: Integrating the AI Brain using Large Language Models

Week 2 marked a major transition from simple data retrieval to intelligent understanding. The focus shifted to integrating a Large Language Model (LLM) to analyze and summarize repository documentation.

The key concepts and skills learned during this phase include:

- Understanding what Large Language Models are and how they process natural language text.
- Recognizing the strengths of LLMs in tasks such as summarization, explanation, and structured text generation.
- Learning the fundamentals of prompt engineering, including role specification and output structuring.
- Designing prompts that guide the model to behave as a technical assistant rather than a generic chatbot.
- Integrating an external LLM API (such as Gemini) into a Python application.
- Handling API keys securely using environment variables.
- Managing practical constraints such as token limits and large README files.

An AI processing pipeline was created where the fetched README content from Week 1 was passed to the LLM, which then generated a concise and structured explanation of the repository. This week introduced the idea of AI as a modular system component rather than a standalone feature.

4.3 Week 3: Repository Structure Analysis and Guided Developer Tour

In Week 3, the project expanded beyond README analysis to include a deeper understanding of the repository's internal structure. The objective was to analyze the file tree and identify important files relevant for developer onboarding.

Major learnings from this week include:

- Fetching the complete repository file tree using GitHub's tree API.
- Filtering out irrelevant files such as build artifacts, dependencies, and auto-generated files.
- Identifying important files based on naming conventions and file types (e.g., `main.py`, `app.py`, configuration files).
- Understanding how different files contribute to a project's workflow.
- Designing logic to extract only meaningful information for new developers.
- Combining README understanding with file structure analysis to produce a cohesive explanation.

This week introduced the concept of a **guided developer tour**, where the AI not only explains what the project does but also highlights where important logic resides. The system began producing structured developer guides instead of plain summaries.

4.4 Week 4: Interactive Web Interface using Streamlit

The final week focused on transforming the backend logic into a user-friendly web application. A Streamlit-based frontend was developed to allow interactive exploration of GitHub repositories.

Key learnings from this phase include:

- Building interactive web applications using Streamlit.
- Managing application state using `st.session_state`.
- Designing clean and minimal user interfaces using custom CSS.
- Handling user input validation and error feedback gracefully.
- Structuring the application into backend logic (data fetching and AI processing) and frontend presentation.
- Improving usability through layout design, expandable sections, and clear call-to-action elements.

By the end of Week 4, the project evolved into a complete AI-powered tool that allows users to input a GitHub repository name and instantly receive a structured developer guide through an interactive web interface.

4.5 Summary of Week-wise Progression

Across four weeks, the project progressed from a simple script to a full-fledged AI application. Each week contributed a critical component, ultimately resulting in a system capable of automating developer onboarding for GitHub repositories.

5 Project Overview and System Architecture

5.1 Project Overview

The **GitHub Repository Guide Agent** is an AI-powered assistant designed to help developers quickly understand unfamiliar GitHub repositories. Instead of manually navigating multiple files and documentation, the system automatically analyzes a repository and generates a structured, human-readable developer guide.

The primary goal of this project is to **reduce onboarding time** for new contributors by answering key questions such as:

- What does this project do?
- How is the codebase organized?
- Which files are important?

- Where should a new developer start?

The system takes a GitHub repository as input and processes its contents using a combination of GitHub's REST API for data retrieval, a Large Language Model (LLM) for reasoning and explanation, and prompt engineering to ensure structured and useful outputs.

This project evolved incrementally over four weeks, starting from simple data fetching and culminating in a fully interactive AI-assisted exploration tool.

5.2 High-Level System Workflow

The system follows a **pipeline-based architecture**, where each stage performs a clearly defined role.
High-level workflow:

1. User provides a GitHub repository link or `owner/repo` name
2. Repository files and documentation are fetched using the GitHub API
3. Relevant content is filtered and preprocessed
4. A carefully designed prompt is constructed
5. The prompt is sent to a Large Language Model
6. The LLM generates a structured developer guide
7. The output is presented to the user via CLI or Streamlit interface

This modular design ensures clarity, maintainability, and ease of extension.

5.3 System Architecture

The architecture of the project is divided into four main components.

5.3.1 User Interface Layer

The user interface layer is responsible for:

- Accepting user input (repository URL or name)
- Displaying the AI-generated developer guide

The project initially used a command-line interface and was later extended to a **Streamlit-based web application** for improved usability.

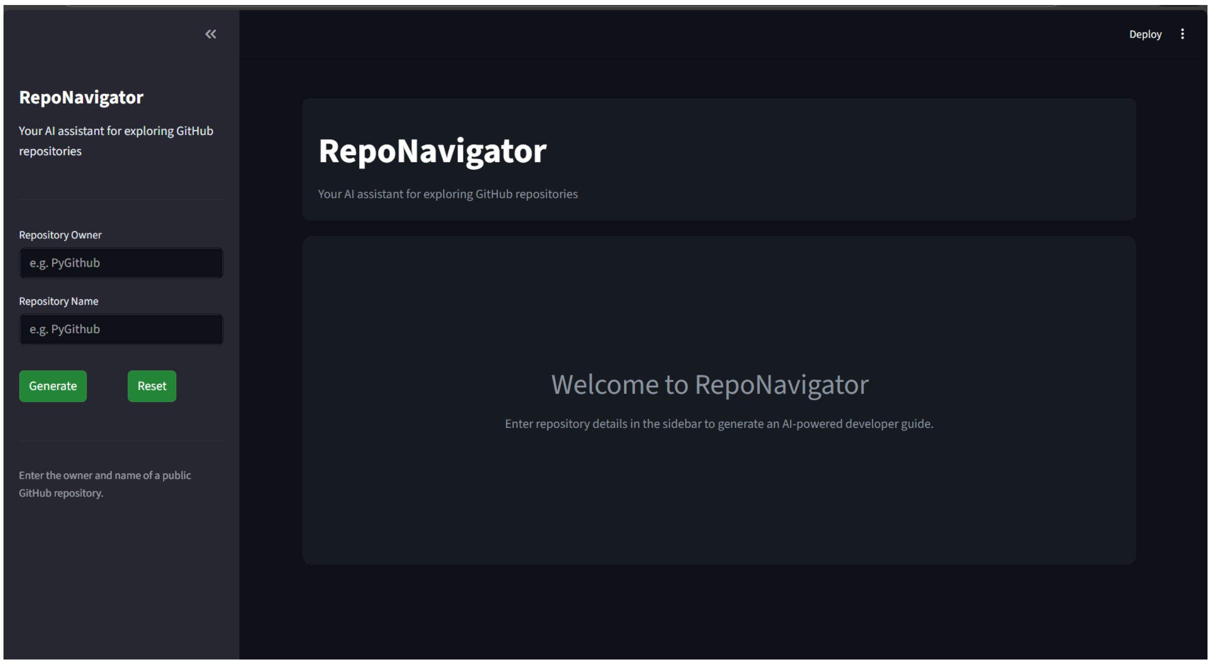

«
RepoNavigator
Your AI assistant for exploring GitHub repositories
Repository Owner
e.g. PyGithub
Repository Name
e.g. PyGithub
Generate
Reset
Enter the owner and name of a public GitHub repository.
Deploy
RepoNavigator
Your AI assistant for exploring GitHub repositories
Welcome to RepoNavigator
Enter repository details in the sidebar to generate an AI-powered developer guide.

Figure 1: Streamlit Web Interface of the GitHub Repository Guide Agent

5.3.2 GitHub Data Extraction Layer

This layer handles all interactions with GitHub using the GitHub REST API through the PyGithub library. It is responsible for fetching:

- README files
- Repository directory structure
- Key source code files

It also manages error handling for invalid repositories, missing files, authentication failures, and API rate limits. This layer is strictly responsible for data retrieval and remains independent of AI reasoning logic.

5.3.3 AI Reasoning Layer (LLM)

The AI reasoning layer uses a Large Language Model to:

- Read and interpret repository documentation
- Infer the project's purpose and structure
- Identify important files and directories
- Generate a logical onboarding path for developers

The model is instructed to behave as a *senior software engineer onboarding a new developer*, ensuring professional and practical explanations.

5.3.4 Prompt Engineering and Control Logic

Prompt engineering plays a critical role in output quality. This component:

- Defines the role and expectations of the LLM
- Enforces structured output formatting
- Reduces hallucinations by grounding responses in repository data
- Separates system instructions from user input

Careful prompt design ensures consistent, readable, and reliable developer guides.

5.4 Architectural Design Principles

The system was designed following key software engineering principles:

- **Modularity**: Each component has a single responsibility
- **Scalability**: Easy extension to support additional repository features
- **Security**: API keys managed using environment variables
- **Robustness**: Graceful handling of missing or incomplete data
- **Explainability**: Outputs are structured and easy to understand

5.5 Technologies Used

- **Python**: Core implementation language
- **PyGithub**: GitHub REST API integration
- **Large Language Model (Gemini-2.5-flash)**: Text understanding and generation
- **Streamlit**: Interactive web-based user interface
- **Environment Variables**: Secure storage of API keys

6 Implementation Details and Experiments

The GitHub Repository Guide Agent was developed incrementally over four weeks using Python, with a modular design separating data fetching, AI reasoning, and user interaction. This approach ensured clarity, scalability, and ease of debugging throughout the development process.

In Week 1, the system was implemented to interact with GitHub using the PyGithub library, authenticate via a Personal Access Token, and fetch README files from public repositories with proper error handling. Week 2 introduced a Large Language Model to summarize repository documentation using structured prompts and secure API integration. In Week 3, repository structure analysis was added by traversing directories and identifying important files and folders, enabling deeper architectural understanding. Finally, Week 4 focused on building a Streamlit-based web interface that allowed users to input repository and owner names and view clear, AI-generated explanations interactively.

6.1 Experiments and Observations

Multiple repositories of varying sizes and domains were tested to evaluate the system's effectiveness.

Observations:

- Repositories with well-written README files produced more accurate summaries
- Very large repositories required content filtering due to LLM context limits
- Prompt refinement significantly improved output structure and relevance
- The system was effective in identifying key files and entry points

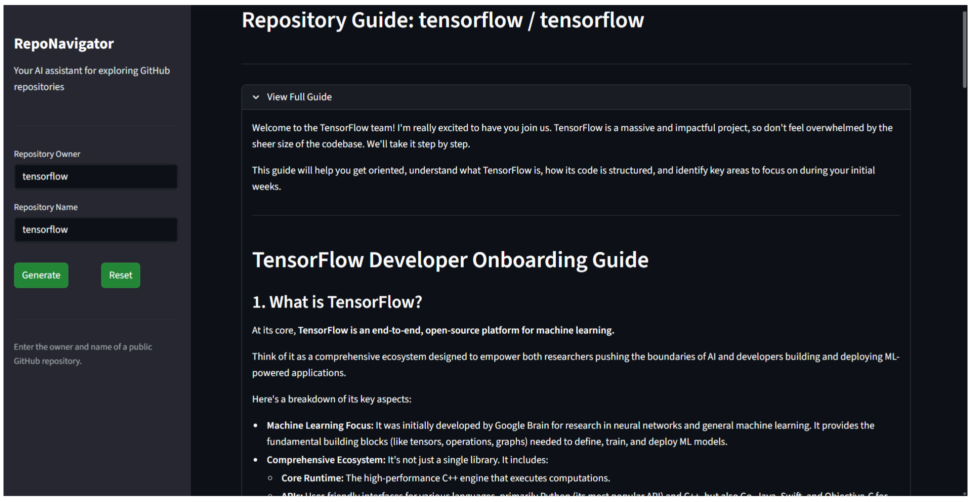
6.2 Limitations Identified

Despite its effectiveness, certain limitations were observed:

- AI-generated explanations may occasionally miss project-specific nuances
- Large repositories may require selective file sampling
- Outputs depend heavily on the quality of repository documentation

7 Final Result

The final outcome of this project is a functional GitHub Repository Guide Agent capable of automatically analyzing and explaining public GitHub repositories. The system integrates GitHub API-based data extraction, Large Language Model reasoning, and a Streamlit-based web interface to generate structured and developer-friendly repository guides.



Repository Owner: tensorflow

Repository Name: tensorflow

Enter the owner and name of a public GitHub repository.

Repository Guide: tensorflow / tensorflow

Welcome to the TensorFlow team! I'm really excited to have you join us. TensorFlow is a massive and impactful project, so don't feel overwhelmed by the sheer size of the codebase. We'll take it step by step.

This guide will help you get oriented, understand what TensorFlow is, how its code is structured, and identify key areas to focus on during your initial weeks.

TensorFlow Developer Onboarding Guide

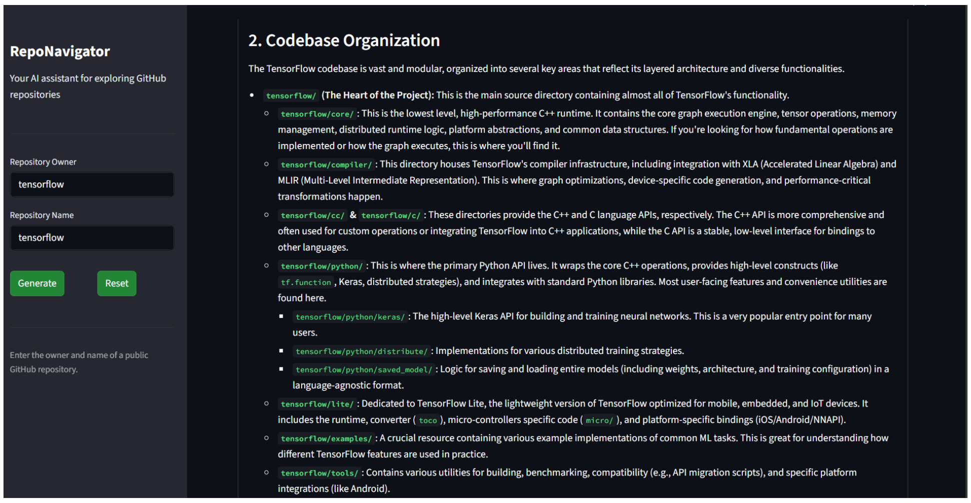
1. What is TensorFlow?

At its core, **TensorFlow is an end-to-end, open-source platform for machine learning.**

Think of it as a comprehensive ecosystem designed to empower both researchers pushing the boundaries of AI and developers building and deploying ML-powered applications.

Here's a breakdown of its key aspects:

- **Machine Learning Focus:** It was initially developed by Google Brain for research in neural networks and general machine learning. It provides the fundamental building blocks (like tensors, operations, graphs) needed to define, train, and deploy ML models.
- **Comprehensive Ecosystem:** It's not just a single library. It includes:
  - **Core Runtime:** The high-performance C++ engine that executes computations.



RepoNavigator

Your AI assistant for exploring GitHub repositories

Repository Owner: tensorflow

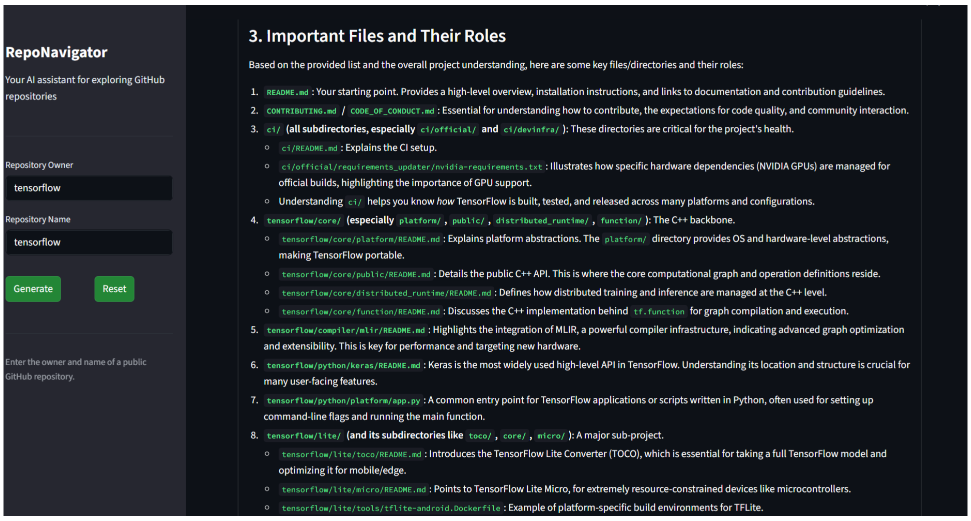
Repository Name: tensorflow

Enter the owner and name of a public GitHub repository.

2. Codebase Organization

The TensorFlow codebase is vast and modular, organized into several key areas that reflect its layered architecture and diverse functionalities.

- `tensorflow/` **(The Heart of the Project):** This is the main source directory containing almost all of TensorFlow's functionality.
  - `tensorflow/core/` : This is the lowest level, high-performance C++ runtime. It contains the core graph execution engine, tensor operations, memory management, distributed runtime logic, platform abstractions, and common data structures. If you're looking for how fundamental operations are implemented or how the graph executes, this is where you'll find it.
  - `tensorflow/compiler/` : This directory houses TensorFlow's compiler infrastructure, including integration with XLA (Accelerated Linear Algebra) and MLIR (Multi-Level Intermediate Representation). This is where graph optimizations, device-specific code generation, and performance-critical transformations happen.
  - `tensorflow/cc/` & `tensorflow/c/` : These directories provide the C++ and C language APIs, respectively. The C++ API is more comprehensive and often used for custom operations or integrating TensorFlow into C++ applications, while the C API is a stable, low-level interface for bindings to other languages.
  - `tensorflow/python/` : This is where the primary Python API lives. It wraps the core C++ operations, provides high-level constructs (like `tf.function` , Keras, distributed strategies), and integrates with standard Python libraries. Most user-facing features and convenience utilities are found here.
    - `tensorflow/python/keras/` : The high-level Keras API for building and training neural networks. This is a very popular entry point for many users.
    - `tensorflow/python/distribute/` : Implementations for various distributed training strategies.
    - `tensorflow/python/saved_model/` : Logic for saving and loading entire models (including weights, architecture, and training configuration) in a language-agnostic format.
  - `tensorflow/lite/` : Dedicated to TensorFlow Lite, the lightweight version of TensorFlow optimized for mobile, embedded, and IoT devices. It includes the runtime, converter (`toco`), micro-controllers specific code (`micro/`), and platform-specific bindings (iOS/Android/NNAPI).
  - `tensorflow/examples/` : A crucial resource containing various example implementations of common ML tasks. This is great for understanding how different TensorFlow features are used in practice.
  - `tensorflow/tools/` : Contains various utilities for building, benchmarking, compatibility (e.g., API migration scripts), and specific platform integrations (like Android).



RepoNavigator

Your AI assistant for exploring GitHub repositories

Repository Owner: tensorflow

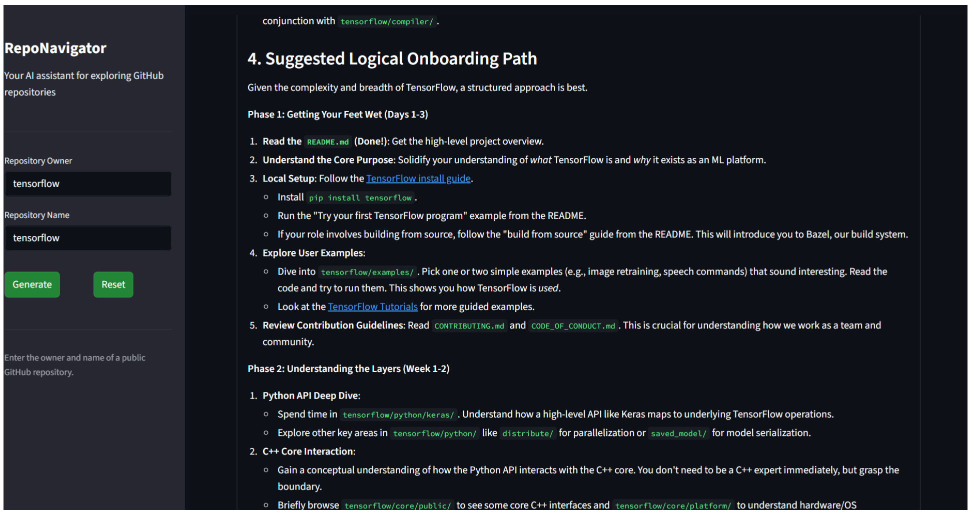
Repository Name: tensorflow

Enter the owner and name of a public GitHub repository.

3. Important Files and Their Roles

Based on the provided list and the overall project understanding, here are some key files/directories and their roles:

1. `README.md` : Your starting point. Provides a high-level overview, installation instructions, and links to documentation and contribution guidelines.
2. `CONTRIBUTING.md` / `CODE_OF_CONDUCT.md` : Essential for understanding how to contribute, the expectations for code quality, and community interaction.
3. `ci/` **(all subdirectories, especially** `ci/official/` **and** `ci/devinfra/`): These directories are critical for the project's health.
   - `ci/README.md` : Explains the CI setup.
   - `ci/official/requirements_updater/nvidia-requirements.txt` : Illustrates how specific hardware dependencies (NVIDIA GPUs) are managed for official builds, highlighting the importance of GPU support.
   - Understanding `ci/` helps you know *how* TensorFlow is built, tested, and released across many platforms and configurations.
4. `tensorflow/core/` **(especially** `platform/` , `public/` , `distributed_runtime/` , `function/`): The C++ backbone.
   - `tensorflow/core/platform/README.md` : Explains platform abstractions. The `platform/` directory provides OS and hardware-level abstractions, making TensorFlow portable.
   - `tensorflow/core/public/README.md` : Details the public C++ API. This is where the core computational graph and operation definitions reside.
   - `tensorflow/core/distributed_runtime/README.md` : Defines how distributed training and inference are managed at the C++ level.
   - `tensorflow/core/function/README.md` : Discusses the C++ implementation behind `tf.function` for graph compilation and execution.
5. `tensorflow/compiler/mlir/README.md` : Highlights the integration of MLIR, a powerful compiler infrastructure, indicating advanced graph optimization and extensibility. This is key for performance and targeting new hardware.
6. `tensorflow/python/keras/README.md` : Keras is the most widely used high-level API in TensorFlow. Understanding its location and structure is crucial for many user-facing features.
7. `tensorflow/python/platform/app.py` : A common entry point for TensorFlow applications or scripts written in Python, often used for setting up command-line flags and running the main function.
8. `tensorflow/lite/` **(and its subdirectories like** `toco/` , `core/` , `micro/`): A major sub-project.
   - `tensorflow/lite/toco/README.md` : Introduces the TensorFlow Lite Converter (TOCO), which is essential for taking a full TensorFlow model and optimizing it for mobile/edge.
   - `tensorflow/lite/micro/README.md` : Points to TensorFlow Lite Micro, for extremely resource-constrained devices like microcontrollers.
   - `tensorflow/lite/tools/tflite-android.Dockerfile` : Example of platform-specific build environments for TFLite.



RepoNavigator

Your AI assistant for exploring GitHub repositories

Repository Owner: tensorflow

Repository Name: tensorflow

Enter the owner and name of a public GitHub repository.

conjunction with `tensorflow/compiler/` .

4. Suggested Logical Onboarding Path

Given the complexity and breadth of TensorFlow, a structured approach is best.

Phase 1: Getting Your Feet Wet (Days 1-3)

1. **Read the** `README.md` **(Done!):** Get the high-level project overview.
2. **Understand the Core Purpose:** Solidify your understanding of *what* TensorFlow is and *why* it exists as an ML platform.
3. **Local Setup:** Follow the TensorFlow install guide.
   - Install `pip install tensorflow` .
   - Run the "Try your first TensorFlow program" example from the README.
   - If your role involves building from source, follow the "build from source" guide from the README. This will introduce you to Bazel, our build system.
4. **Explore User Examples:**
   - Dive into `tensorflow/examples/` . Pick one or two simple examples (e.g., image retraining, speech commands) that sound interesting. Read the code and try to run them. This shows you how TensorFlow is *used*.
   - Look at the TensorFlow Tutorials for more guided examples.
5. **Review Contribution Guidelines:** Read `CONTRIBUTING.md` and `CODE_OF_CONDUCT.md` . This is crucial for understanding how we work as a team and community.

Phase 2: Understanding the Layers (Week 1-2)

1. **Python API Deep Dive:**
   - Spend time in `tensorflow/python/keras/` . Understand how a high-level API like Keras maps to underlying TensorFlow operations.
   - Explore other key areas in `tensorflow/python/` like `distribute/` for parallelization or `saved_model/` for model serialization.
2. **C++ Core Interaction:**
   - Gain a conceptual understanding of how the Python API interacts with the C++ core. You don't need to be a C++ expert immediately, but grasp the boundary.
   - Briefly browse `tensorflow/core/public/` to see some core C++ interfaces and `tensorflow/core/platform/` to understand hardware/OS