COBRA: Coding Biographies Rendered with AI

Prayash Joshi
Virginia Tech

December 15, 2024

Abstract

This report presents COBRA (Coding Biographies Rendered with AI), a novel system that transforms developers' GitHub data into personalized, collectible profile cards. By combining Hidden Markov Models (HMMs) and Large Language Models (LLMs), COBRA provides a unique approach to representing developer profiles and analyzing coding patterns. The system implements a dual-analysis pipeline: quantitative assessment using HMMs for temporal pattern recognition and qualitative evaluation using LLM-based agents for code quality assessment. Results demonstrate the effectiveness of this hybrid approach in capturing both the evolution of development practices and the nuanced aspects of code quality. The system achieved meaningful state identification with stability scores ranging from 0.57 to 0.83 across test repositories, while maintaining interpretable developer profile representations.

1 Introduction

The representation of developer expertise and coding practices is a significant challenge in software engineering. Traditional approaches often fail to capture the dynamic nature of development work and the evolution of coding practices over time. COBRA (Coding Biographies Rendered with AI) addresses this gap by introducing a novel approach that combines statistical modeling with AI-driven analysis to create comprehensive developer profiles.

COBRA aims to tackle the limitations of current methods for showcasing developer skills and contributions, which often rely on simple metrics like contribution graphs or repository statistics. These methods fall short in capturing the temporal evolution of coding practices, qualitative aspects of code quality, the context of development decisions, and the relative importance of different contributions.

The primary objectives of COBRA include developing a robust framework for analyzing GitHub repositories using both statistical and AI-driven methods, creating interpretable representations of developer profiles, establishing meaningful metrics for code quality and development patterns, and providing an engaging way to showcase developer skills through collectible cards.

2 Technical Approach

2.1 System Architecture

COBRA implements a dual-analysis pipeline that combines statistical modeling with AI-driven code analysis. The system architecture consists of three main components:

1. **Data Management Layer**: This layer handles repository cloning, commit history analysis, and data preprocessing. The `RepoDataManager` class in `data_manager.py` efficiently handles

repository caching, commit processing, and file classification. It uses optimized strategies like direct string manipulation for path handling, set-based file classification, and caching mechanisms to improve performance.[4]

2. **Analysis Engine**: The analysis engine comprises the HMM analyzer and LLM-based agent. The `CobraHMM` class in `cobra_hmm.py` implements the Hidden Markov Model (HMM) component, which models development patterns using a Gaussian HMM with three hidden states.[6] The LLM-based agent, implemented in the `agent.py` file, performs qualitative code analysis using the Ollama language model and LangChain for document processing and retrieval.

3. **Profile Generation System**: This component transforms the analysis results into collectible cards. (Implementation details to be added)

2.2 Hidden Markov Model Implementation

The HMM component uses a Gaussian HMM with three hidden states to model development patterns. The key features of the HMM implementation include biweekly aggregation of commit data for stability, a feature set consisting of commit frequency, code changes, test coverage, and maintainability, improved convergence through multiple random restarts, and state interpretation based on feature patterns.[5] The `CobraHMM` class encapsulates the core HMM implementation, providing methods for model training, prediction, and repository analysis.

2.3 LLM-Based Code Analysis

The qualitative analysis component utilizes a LangChain-based architecture with document processing using a FAISS vector store [2], code quality assessment using the Ollama (qwen2.5-coder:7b) language model[1], a custom prompting system for consistent evaluation, and structured output parsing for quantitative scoring.

3 Methodology

3.1 Data Collection and Preprocessing

Repository data is collected through a process that involves repository cloning and metadata extraction, commit history analysis and feature extraction, file classification and content analysis, and biweekly data aggregation for stability. The `RepoDataManager` class handles the data collection and preprocessing steps efficiently.

3.2 Analysis Pipeline

The analysis pipeline consists of two parallel streams:

HMM Analysis This stream involves feature scaling and normalization, model training with multiple restarts, state sequence prediction, and state interpretation and scoring.

LLM Analysis The LLM analysis stream includes document chunking and embedding, vector store creation, code quality assessment, and score aggregation and interpretation.

4 Results and Discussion

4.1 Quantitative Results

Analysis of test repositories yielded quantitative metrics, as shown in Table 1.

Repository	Stability	Maturity	Entropy	Dominant State
codekids	0.74	0.56	1.22	State 1 (56.41%)
llama2	0.83	0.54	0.82	State 1 (83.33%)
filmgenie	0.57	0.51	1.30	State 0 (62.50%)
vector-pytorch	0.79	1.13	0.84	State 0 (82.20%)

Table 1: Repository Analysis Results

The HMM analysis provided quantitative metrics for each repository, including stability scores, maturity scores, state entropy, and dominant states. These metrics offer insights into the development patterns and evolution of the repositories over time.

4.2 Qualitative Analysis

The LLM-based analysis provided qualitative insights into various aspects of the codebase, such as code quality patterns, documentation completeness, adherence to best practices, and development patterns and practices.

File	Score	Justification Summary
configurator.py	3.0	Lacks structure, documentation, error handling, uses `exec`.
sample.py	6.0	Lacks documentation, modularity, and proper error handling.
test_all.py	7.0	Functional, uses `pytest`, but lacks documentation and best practices.
tinystories.py	4.0	Modular but needs better structure and best practices.
tokenizer.py	8.0	Well-structured, with type hints, but lacks robust error handling.
train.py	8.0	Comprehensive training loop implementation with DDP and logging.
README.md	8.0	Well-documented, but some TODOs remain unsorted.
doc/stories260K.md	5.0	Informative markdown, lacks error handling or executable code.
doc/train_llama_tokenizer.md	7.0	Clear instructions, missing error handling and best practice details.

Table 2: Single-Agent Analysis Results

The single-agent analysis assigned scores to individual files based on factors such as code quality, documentation, and adherence to best practices. Files like `tokenizer.py` and `train.py` received high scores for their well-structured implementation and comprehensive functionality. On the

other hand, files like `configurator.py` and `tinystories.py` received lower scores due to lack of documentation, error handling, and best practices.

4.3 Challenges and Limitations

Several challenges were encountered during the implementation of COBRA. Training HMMs can be computationally expensive, especially with larger datasets, and selecting the optimal number of hidden states is also a challenge that may require experimentation and domain knowledge. Ensuring consistent and reliable code quality assessments using LLMs can be difficult due to their inherent unpredictability and dependence on prompt engineering. LLMs have limited context windows, which can hinder their ability to understand and analyze large codebases effectively. Striking the right balance between automated analysis and interpretability of the results is crucial to provide meaningful insights to users.

One significant limitation is the scalability of the analysis pipeline for large repositories with extensive commit histories. The current implementation may face performance bottlenecks when processing vast amounts of data. Additionally, the accuracy of the HMM and LLM-based analyses heavily relies on the quality and representativeness of the training data. Ensuring a diverse and comprehensive dataset is essential for generating reliable insights.

5 Future Work

Future developments for COBRA include the implementation of a multi-agent debating architecture that incorporates a system where agents engage in structured debates to provide more comprehensive and balanced code analysis.

Another area of future work is the integration of generative AI models to create unique and visually appealing artwork for the collectible profile cards. The development of social features and gamification elements can be introduced to enhance user engagement and foster a sense of community among developers. Expanding the validation of COBRA to cover a wider range of project types, sizes, and domains is necessary to assess its generalizability and robustness. Improving the interpretability of HMM state transitions is also an important future direction to provide more meaningful insights into the development lifecycle and patterns.

One potential approach to implement the multi-agent debating architecture is to assign different roles to each agent, such as a proponent agent that highlights the strengths of the codebase and an opponent agent that identifies areas for improvement.[3] These agents can engage in structured arguments, presenting evidence and counterarguments based on the analysis results. A moderator agent can facilitate the debate and summarize the key points for the user.

To integrate generative AI for card artwork, techniques like Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs) can be explored. These models can be trained on a dataset of existing developer profile designs and artwork to learn the underlying patterns and generate novel card designs that align with the user's preferences and the analyzed repository's characteristics.

6 Conclusion

COBRA demonstrates the potential of combining statistical modeling with AI-driven analysis to create meaningful representations of developer profiles. The dual-analysis approach, leveraging

HMMs and LLMs, provides both quantitative metrics and qualitative insights, offering a comprehensive view of development practices. The system's ability to generate interpretable results while maintaining engagement through collectible cards presents a promising direction for developer profile representation.

The project successfully achieved its objectives of developing a robust framework for analyzing GitHub repositories, creating interpretable developer profiles, establishing meaningful metrics for code quality and development patterns, and providing an engaging way to showcase developer skills. However, there are challenges and limitations to be addressed, such as the computational intensity of HMMs, consistency issues with LLMs, context window limitations, and balancing automated analysis with interpretability.

Future work on COBRA includes the implementation of a multi-agent debating architecture, integration of generative AI for card artwork, development of social features and gamification, validation across diverse project types, and enhanced interpretability of state transitions. Overall, COBRA presents a novel approach to developer profile representation that leverages the strengths of statistical modeling and AI-driven analysis, opening up new possibilities for showcasing and appreciating developer skills and contributions.

7 Figures

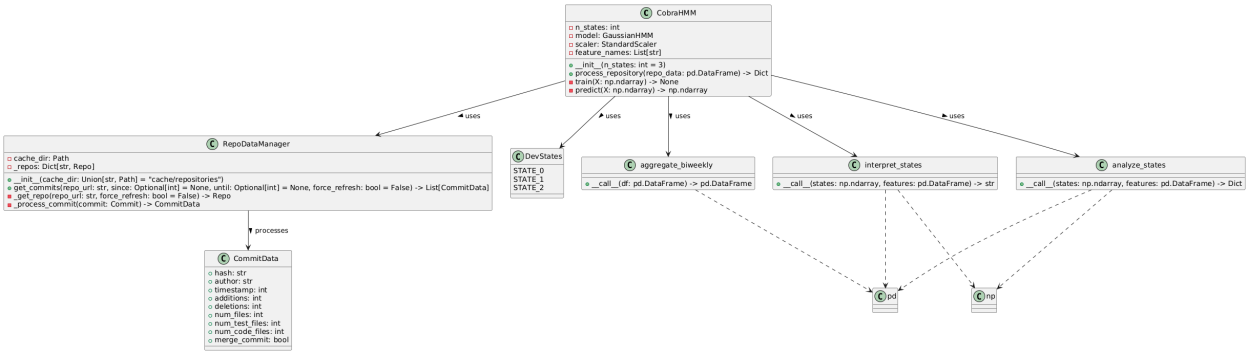

Figure 1: Cobra-HMM Architecture

References

[1] LangChain Contributors. Chatollama - langchain api reference, 2024. Accessed: 2024-12-15.

[2] LangChain Contributors. Faiss - langchain community api reference, 2024. Accessed: 2024-12-15.

[3] LangGraph Contributors. Langgraph reference, 2024. Accessed: 2024-12-15.

[4] PyGithub Contributors. Pygithub documentation, 2024. Accessed: 2024-12-15.

[5] hmmlearn Contributors. hmmlearn documentation, 2024. Accessed: 2024-12-15.

[6] N. Sviridov, M. Evtikhiev, and V. Kovalenko. Tnm: A tool for mining of socio-technical data from git repositories. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 295–299, 2021.

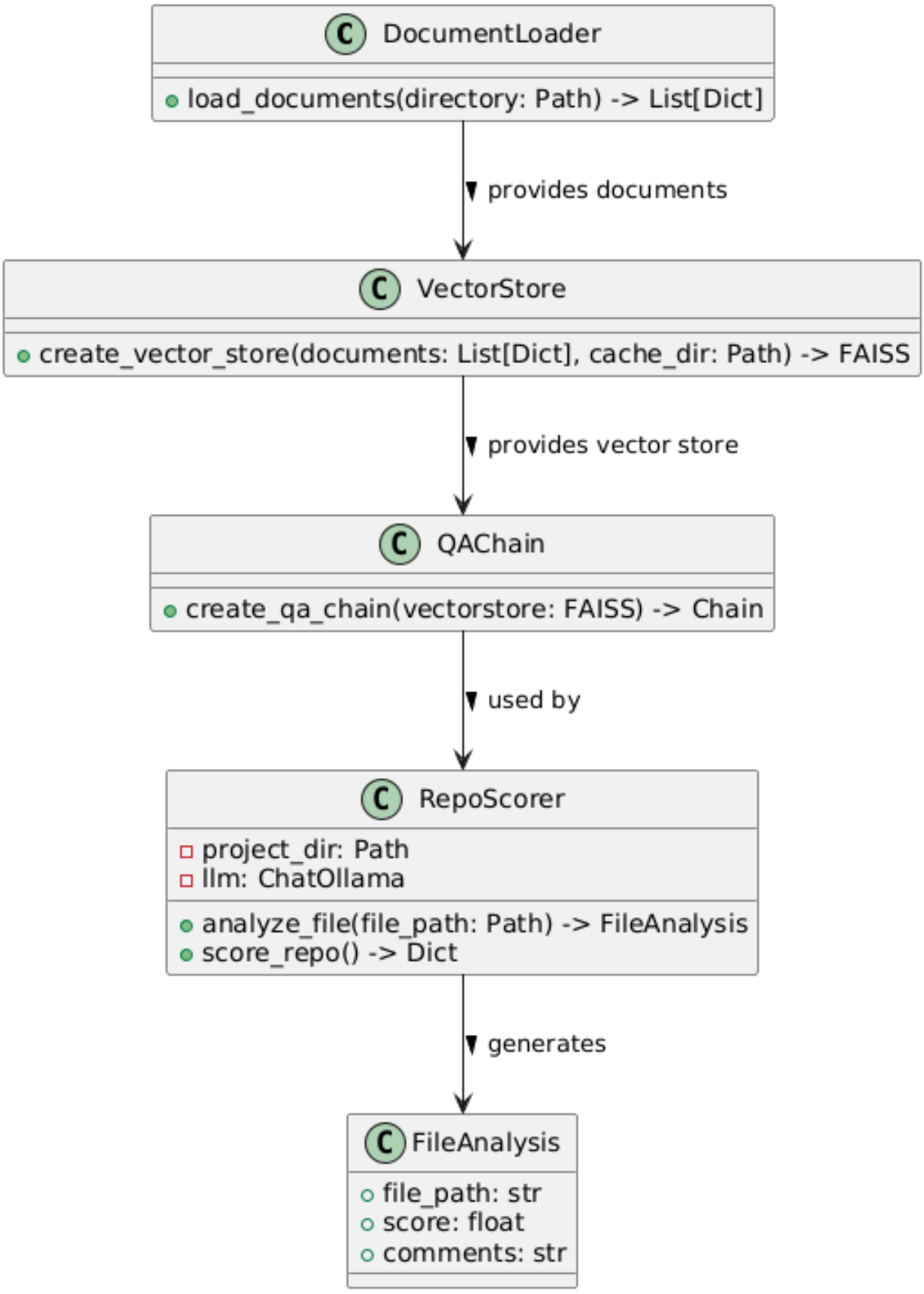

Figure 2: Single-Agent Architecture

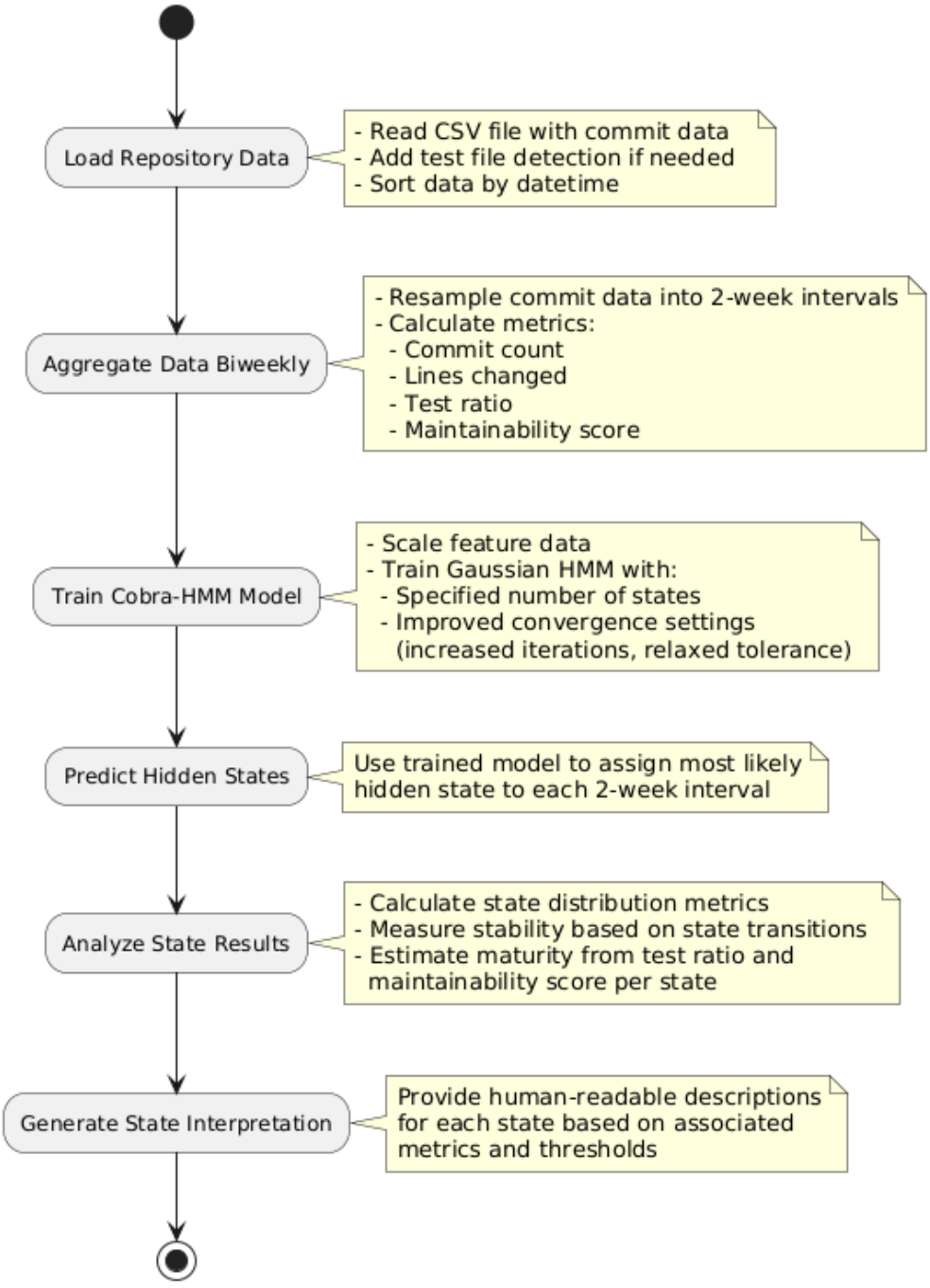

Figure 3: Cobra-HMM Pipeline

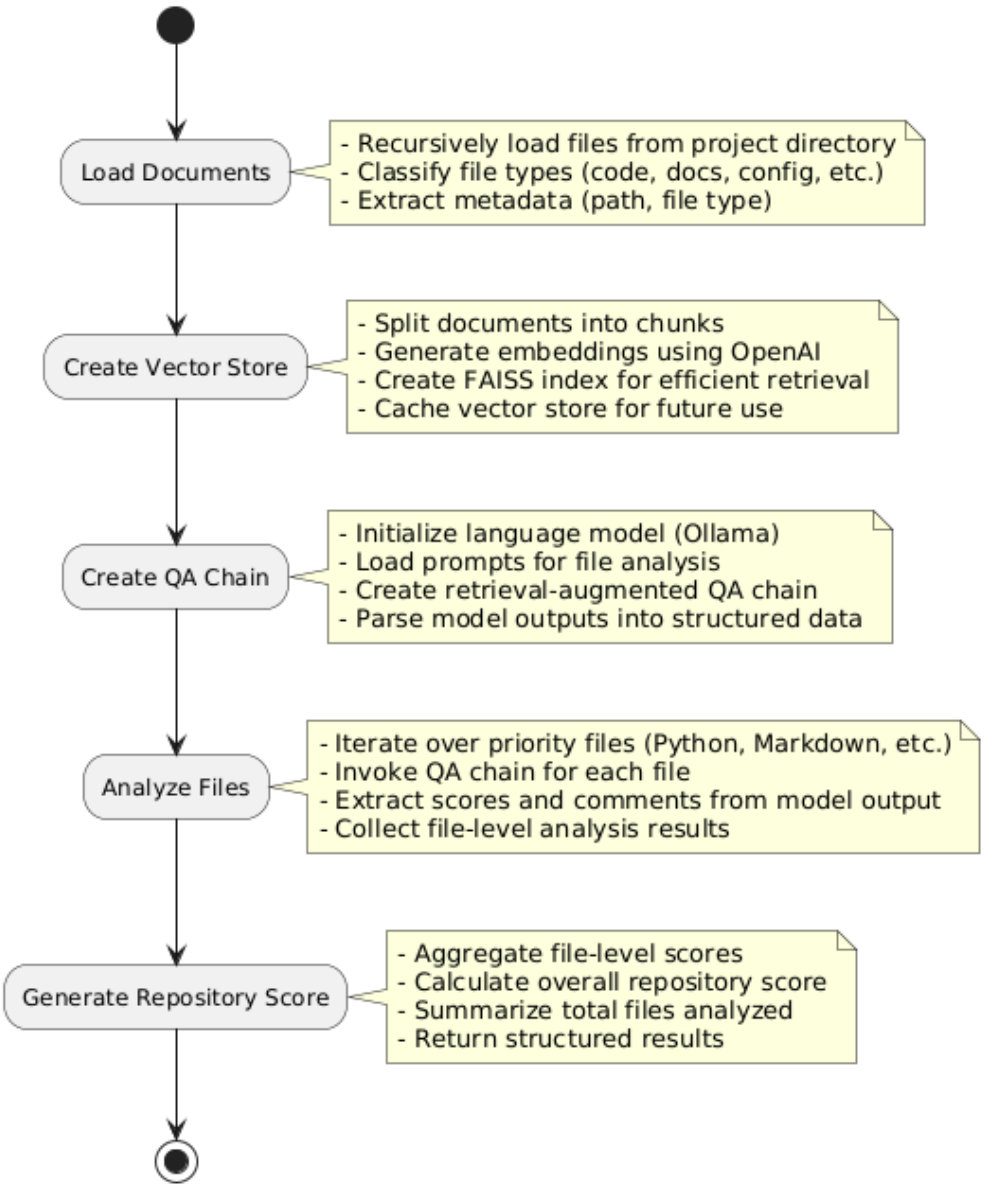

Figure 4: Single-Agent Pipeline

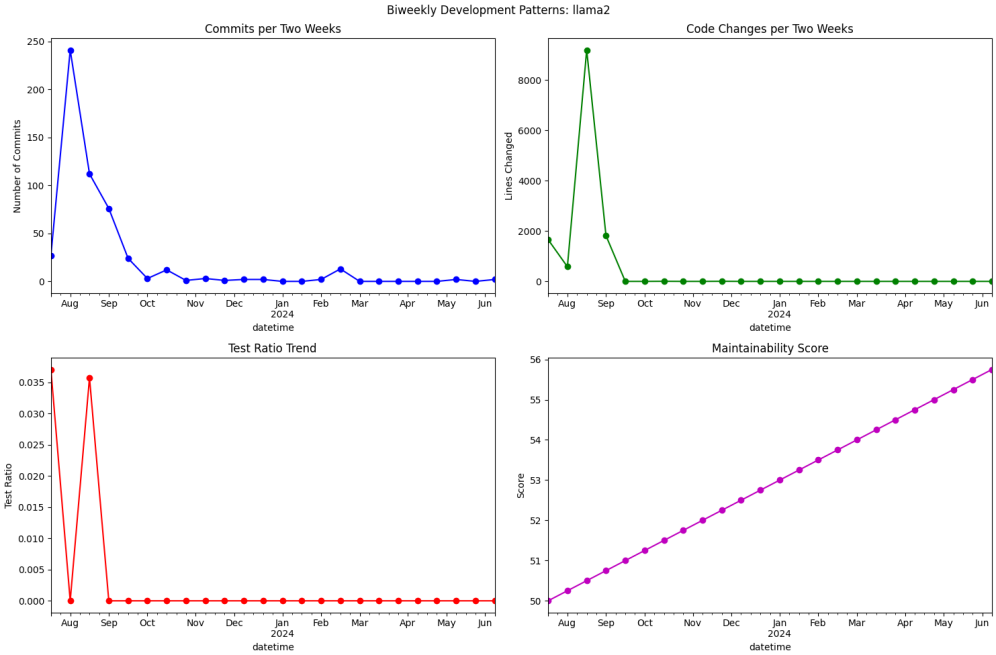

Figure 5: Stability and Maturity Scores

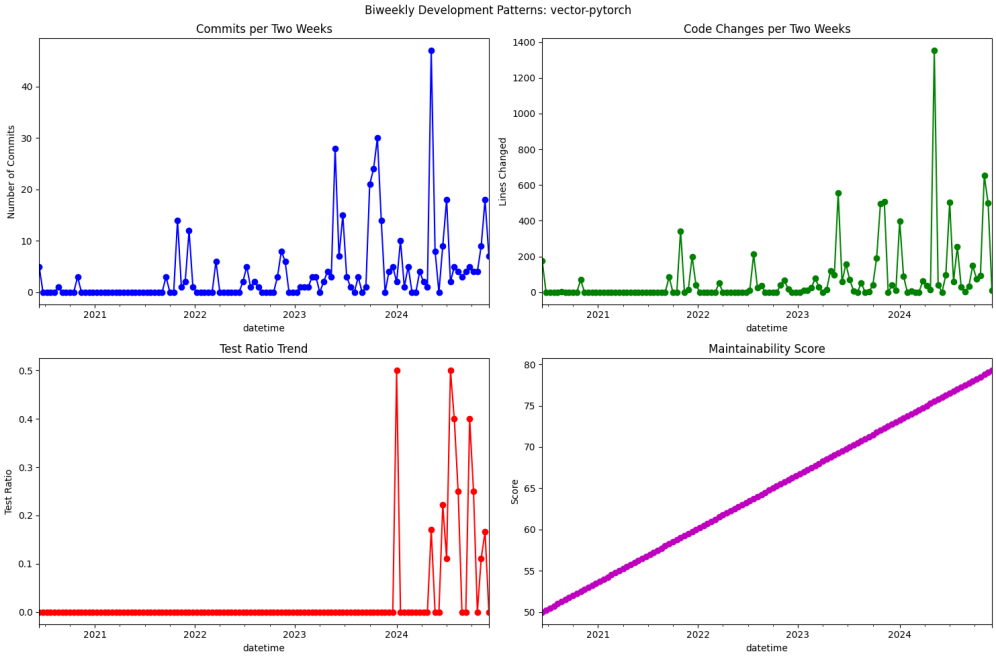

Figure 6: State Distribution Heatmap

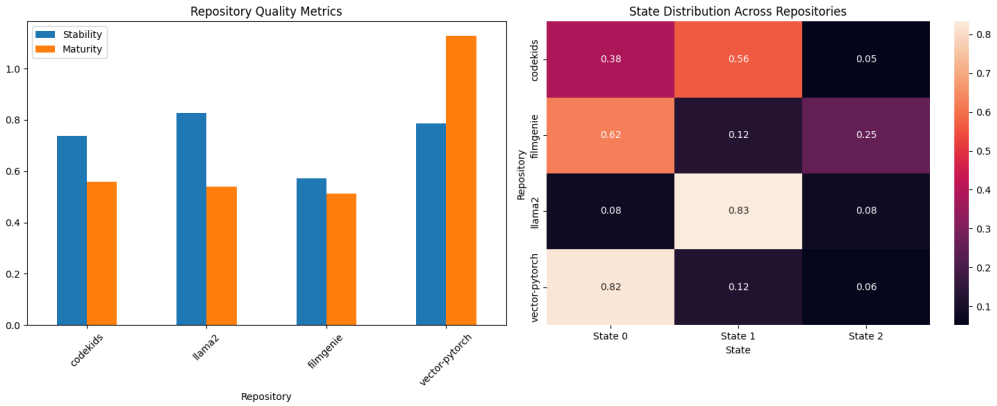

Figure 7: Detailed Biweekly Patterns for vector-pytorch