Explainable AI For Source Code Applications

Design Document

SDMAY25-30

Client: Dr. Ali Janesari
Advisor: Arushi Sharma

Team:
Manjul Balayar
Sam Frost
Akhilesh Nevatia
Ethan Rogers
Rayne Wilde

sdmay25-30@iastate.edu
https://sdmay25-30.sd.ece.iastate.edu/

Revised: 5/04/2025

Executive Summary

Project Summary

Our project focused on enhancing the interpretability and explainability of large language models (LLMs) trained on source code. As these models become increasingly integral in software development tasks—such as code completion, error detection, and automated code generation—their "black-box" nature poses significant challenges. Developers and researchers often struggle to understand the reasoning behind the models' outputs, which can lead to mistrust, inefficiencies, and potential errors in critical software systems.

Problem Importance

The opaque decision-making processes of LLMs hinder their effective integration into software development workflows. Without clear insights into how these models interpret code and generate outputs, it becomes difficult to ensure code reliability, maintainability, and security. Improving model transparency is essential for building trust, facilitating debugging, and enhancing collaboration between AI systems and human developers.

Key Design Requirements

Our primary design requirements were:

- **Scalability and Efficiency**: The library must handle large datasets and complex models efficiently, optimizing computational resource usage.
- **Modularity and Maintainability**: The library must be a structured, installable Python library following best coding practices (PEP8 guidelines) for ease of maintenance and expansion.
- **Comprehensive Functionality**: The library must include modules for activation extraction, clustering algorithms, automated labeling, and visualization tools.
- **High-Performance Computing (HPC) Compatibility**: The library must operate effectively in HPC environments to leverage powerful computational resources.
- **Ethical Standards Compliance**: The team must adhere to ethical guidelines and engineering standards to address biases and promote responsible AI practices.

Design Overview

We developed a Python library that provides tools for latent concept analysis of code-based LLMs. Key approaches and technologies included:

- **Activation Extraction Module:** Extracted neuron activations from specific layers of LLMs using NeuroX and related extraction techniques, enabling analysis of internal model representations.
- **Clustering Algorithms:** Implemented multiple algorithms, including K-means, BIRCH, Agglomerative Clustering, Leaders Clustering, and Hyperbolic Clustering, to group similar activations and uncover latent code concepts.

- **Code Parsing with Tree-sitter:** Parsed source code across multiple programming languages, generating abstract syntax trees (ASTs) for structured analysis.
- **Automated Labeling:** Leveraged large language models and prompt optimization techniques (e.g., DSPY2) to automatically label code clusters, significantly reducing manual effort.
- **Visualization Tools:** Developed graphical interfaces featuring interactive sunburst and dendrogram visualizations (built with Dash, Plotly, and SciPy) to help users interpret clustering results and model behaviors.
- **HPC Integration:** Adapted the library for use on HPC clusters like Iowa State's Nova, optimizing for parallel processing and large-scale data handling.

Delivered Functionality

To date, we have:

- **Refactored Existing Code:** Adapted scripts from the CodeConceptNet library, enhancing structure and compliance with best coding practices.
- **Implemented Core Modules:** Completed robust implementations of data parsing, activation extraction, clustering algorithms, and automated labeling modules.
- **Integrated HPC Compatibility:** Configured and validated the library to effectively run on HPC clusters, addressing performance and scalability requirements.
- **Web Application:** Developed and deployed a Flask-based Python web application using Dash, Plotly, and SciPy to visualize activation data via interactive sunburst and dendrogram diagrams.
- **Focused on Ethical Practices:** Consistently incorporated ethical considerations, adhering to relevant standards, and addressing biases in model interpretability.

Design Effectiveness

Our design meets the requirements by providing a modular, scalable library that facilitates in-depth analysis of LLMs. We have validated functionality through testing and initial deployments on HPC environments. User feedback and testing results indicate that the library effectively aids in interpreting model behaviors, aligning with user needs for transparency and explainability.

Learning Summary

Development Standards & Practices Used

- **PEP8 – Style Guide for Python Code**: Followed PEP8 guidelines for consistent and readable Python code.
- **Agile Development Methodology**: Employed Agile practices, including iterative sprints, regular meetings, and adaptive planning.
  Version Control with Git and GitLab: Used Git for version control and hosted our repository on GitLab for collaboration, issue tracking, and CI/CD pipelines.
- **Continuous Integration/Continuous Deployment (CI/CD)**: Implemented CI/CD pipelines using GitLab Runners and configuration files to automate testing and deployment.
- **Software Testing Standards**: Applied principles from IEEE 1028-2008 for software reviews and audits, including unit testing, integration testing, and code reviews.
- **High-Performance Computing (HPC) Practices**: Followed best practices for HPC environments, optimizing resource utilization and job scheduling on clusters like Nova.
- **Ethical Standards in AI**: Incorporated guidelines from ISO/IEC TR 24027:2021 to address bias in AI systems and ensure ethical AI development.
- **Software Quality Standards**: Referred to ISO/IEC 25010:2011 for defining and evaluating software quality attributes like reliability and usability.

Summary of Requirements

Functional Requirements:

- Develop a scalable and maintainable Python library for latent concept analysis on source code.
- Implement activation extraction from neural network models trained on code.
- Include efficient clustering algorithms to group activations and uncover latent code concepts.
- Provide support for parsing source code files in multiple programming languages, including those not natively supported by Tree-sitter.
- Develop an automated labeling module using large language models (LLMs) and prompt optimization techniques.
- Create visualization tools and a graphical user interface (GUI) for data analysis and interaction.
- Ensure compatibility with HPC environments for processing large datasets.

Non-Functional Requirements:

- Achieve at least 60% code coverage with unit tests.
- Follow PEP8 coding standards for Python code.
- Provide comprehensive documentation and usage examples.Optimize code for performance and scalability.
- Address ethical considerations, including bias mitigation and adherence to professional standards.

Constraints:

- Use Agile methodology for project management.
- Use Iowa State's GitLab for version control and project tracking.
- Complete the project within the given timeline and limitations.

Applicable Courses from Iowa State University Curriculum

- COM S 227: Introduction to Object-Oriented Programming
- COM S 228: Introduction to Data Structures
- COM S 311: Design and Analysis of Algorithms
- COM S 474/574: Introduction to Machine Learning
- SE 329: Software Project Management
- SE 339: Software Architecture and Design
- ENGL 314: Technical Communication

New Skills/Knowledge acquired that was not taught in courses

- **Advanced Deep Learning Techniques**: Working with large language models (LLMs) like GPT and BERT, including fine-tuning and prompt engineering.
- **High-Performance Computing (HPC)**: Gained experience with HPC environments, including job scheduling with SLURM and optimizing code for parallel execution on clusters like Pronto.
- **Tree-sitter and Code Parsing**: Learned to use Tree-sitter for parsing source code and extending its capabilities to support languages and constructs not natively handled, such as OpenMP.
- **Ethical AI Standards**: Familiarized with ISO/IEC TR 24027:2021 guidelines for addressing bias in AI systems and applying ethical considerations in AI development.
- **Automated Labeling Techniques**: Developed methods for automated dataset labeling using LLMs and prompt optimization, including the use of DSPY2.
- **Advanced Clustering Algorithms**: Explored and implemented clustering algorithms beyond standard coursework, such as BIRCH and agglomerative clustering.

Table of Contents

Images/Tables

1. Introduction

1.1. Problem Statement

Developers are increasingly relying on artificial intelligence (AI) models, specifically large language models (LLMs), to assist with writing, understanding, and optimizing code. However, these AI models are often reduced to "black boxes," preventing users from achieving the understanding they need to complete their work. Our project addressed this problem by making AI models that work with source code more explainable and interpretable. We developed methods to automatically label code datasets using tools that analyze code structure and patterns. By clustering and examining the concepts that AI models learn from these datasets, we aligned them with human-understandable code properties. This revealed the patterns and logic the AI uses, effectively opening up the black box.

Peering into the LLM black box provided a drastically increased level of understanding with the potential to impact various fields. Examples of this include (1) developers who use AI better understanding why the LLM makes certain suggestions with newly added context, (2) developers of AI models who need to peer inside their products to better design the next iteration for improved performance, and (3) a more theoretical impact in inter-AI communication for self-learning and self-testing.

1.2. Intended Users

Client: Dr. Ali Jannesari/ISU Software Analytics and Pervasive Parallelism (SwAPP) Lab

We identified three main user personas who benefit from our solution. Due to the academic nature of the project, our main users are researchers of various backgrounds and experience

1. ML Researchers and Engineers:

User Description:

Machine Learning (ML) researchers and engineers are professionals working on cutting-edge AI technologies, particularly in areas such as generative AI and large language models (LLMs). These individuals are often well-versed in deep learning, natural language processing (NLP), and high-performance computing (HPC). Their roles require them to not only develop and deploy models but also deeply understand the behavior of these models. A critical part of their work involves interpreting complex model outputs and fine-tuning models to ensure their accuracy and reliability. They are highly analytical, detail-oriented, and constantly seeking ways to improve model performance and transparency.

Needs Statement:

ML researchers and engineers have a strong need for advanced tools to evaluate and interpret the inner workings of models. While there are existing libraries for analyzing general ML models, there is often a gap when it comes to interpreting deep NLP models or working with specific HPC

environments. They require a solution that can streamline the evaluation of ML models, providing meaningful insights into how models handle large datasets, optimize parameters, and make predictions. Additionally, they need a way to efficiently generate or analyze code in HPC environments where current tools might be lacking or inefficient.

How They Benefit:

The product, by integrating functionalities such as activation extraction, clustering optimization, and enhanced visualization, enables ML researchers to accelerate their analysis processes. It addresses their needs by offering a tailored solution for evaluating the latent concepts that underlie model behavior, improving interpretability, and speeding up experimentation cycles. This allows researchers to focus on refining models and pushing the boundaries of AI, rather than being bogged down by manual or inefficient analysis methods. In the context of the overarching problem statement, this benefit aligns with the project's goal of creating a tool that aids in the latent concept analysis of source code. By offering a solution that meets these specific needs, the product ensures that the models researchers build and work with are better understood, more interpretable, and easier to improve.

2. Prompt Engineers and Researchers:

User Description:

User Description:

Prompt engineers and researchers are specialized professionals focusing on optimizing interactions with AI models, particularly in the context of generative AI and large language models (LLMs). They are responsible for designing and refining the input prompts that guide these models to produce accurate, relevant, and high-quality outputs. These individuals are typically highly skilled in understanding model architecture, NLP techniques, and the nuances of prompt construction. They have a deep understanding of the context and intent behind prompts and are continuously experimenting with prompt variations to enhance model performance and generate more coherent, contextually appropriate responses.

Needs Statement:

Prompt engineers and researchers need sophisticated tools for evaluating and interpreting the effects of their prompt designs on model behavior. In particular, they require mechanisms to analyze how prompts influence model outputs, identify areas for improvement, and understand the inner workings of deep NLP models. These users are often dealing with complex models, such as LLMs, which require precise and well-constructed prompts to generate valuable outputs. A key challenge is determining how specific prompt variations can impact the model's understanding and overall performance, which is critical for tasks such as model fine-tuning and ensuring ethical AI behavior.

How They Benefit:

The product offers prompt engineers and researchers the tools necessary to streamline their prompt evaluation processes. By providing features like activation extraction, clustering of model outputs

based on prompt variation, and detailed visualizations, it enables these users to deeply analyze how prompts affect the behavior of generative AI models. This empowers them to quickly iterate on and refine their prompt designs, ensuring that the models produce optimal results in real-world applications. This benefit directly aligns with the overarching problem statement, as it contributes to the broader goal of improving model interpretability and performance. The product helps prompt engineers save time, improve their models' reliability, and ensure that their designs are driving meaningful, impactful outputs from advanced AI systems.

3. HPC Researchers:

User Description:

HPC researchers are professionals working on the intersection of machine learning (ML) and HPC systems. These individuals focus on leveraging large-scale computing resources to run complex simulations, process vast datasets, and evaluate advanced ML models such as Generative AI and LLMs. HPC researchers are typically skilled in parallel computing, distributed systems, and specialized hardware configurations like GPUs and TPUs. Their primary role is to optimize the performance of ML models by using HPC resources to handle computationally intensive tasks, which enables them to work with massive datasets and conduct detailed evaluations of model interpretability and behavior at a scale that would be impractical with traditional computing.

Needs Statement:

HPC researchers need tools that enable them to utilize their high-performance infrastructure to evaluate and interpret ML models effectively. They require the ability to run large-scale experiments, process complex models (such as deep NLP models), and analyze model outputs in an efficient manner. Their current challenge lies in the lack of streamlined, integrated tools that support both model interpretability and high-performance computing. The existing solutions often either lack scalability or do not offer deep insights into model behavior when applied in HPC environments.

How They Benefit:

The product allows HPC researchers to capitalize on their high-performance systems to evaluate and interpret ML models, including generative AI and deep NLP models. By providing features like scalable activation extraction, clustering of large datasets, and tools for efficient parallel processing, the product ensures that these users can conduct their research at scale without sacrificing interpretability. This enhances their ability to derive meaningful insights from complex models, improving both performance and understanding. This benefit aligns with the overarching problem statement by supporting latent concept analysis in large-scale environments, ensuring that models are interpretable even when operating with vast datasets and distributed systems. Ultimately, the product empowers HPC researchers to unlock the full potential of their infrastructure, enabling faster analysis and deeper insights.

2. Requirements, Constraints, And Standards

2.1. Requirements & Constraints

The primary goal was to develop a library that enables researchers and practitioners to perform latent concept analysis on source code, offering functionalities from data input to visualization.

2.1.1 Functional Requirements (Specification)

a. Activation Extraction

- Requirement: The library shall provide methods to extract activation data from neural network models.

- Users must be able to specify which layers or components of the model to extract activations from.

b. Clustering Algorithms

- Requirement: The library shall include efficient and maintainable clustering algorithms.

- Agglomerative Clustering: Improve implementation for better performance.

- K-means Clustering: Optimize methods for handling large datasets.

- Leaders Clustering: Ensure code is modular and reusable.

- BIRCH Clustering: Integrate smoothly into the existing framework.

- Hyperbolic Clustering: Refactor for better integration with overall clustering strategies.

- Documentation: Provide comprehensive documentation for all clustering algorithms.

c. Alignment and Metrics

- Requirement: The library shall provide tools for evaluating the alignment of clusters with known concepts.

- Apply lexical and contextual criteria.

- Support enhanced alignment metrics (specifications to be provided).

- Functionality: Users must be able to specify which metrics to use for analysis.

d. Analysis Functionalities

- Requirement: Enhance analysis functionalities to provide deeper insights.

- Reporting: Generate comprehensive reports summarizing findings and insights.

- Custom Analyses: Enable users to perform tailored analyses based on specific requirements and datasets.

- Functionality: Users must be able to analyze clustering results and visualize relationships between clusters effectively.

e. Input Data Support

- Requirement: The library shall support additional input formats.

- Source code files for all languages supported by Tree-sitter.

- Support for OpenMP code (even though Tree-sitter does not currently support it) (constraint).

- Datasets in CSV and JSON formats.

- Functionality: Users must be able to load data efficiently and handle errors related to unsupported formats.

f. Ground Truth Datasets Integration

- Requirement: Integrate ground truth datasets for validation purposes.

- Ontologies: Semantic ground truths for Software Engineering SE and HPC.

- Existing Datasets: SE and HPC downstream tasks.

- Static Analysis Tools: Incorporate Tree-sitter as part of the base pipeline.

- Functionality: Users must be able to access and utilize these datasets for enhanced analysis and validation.

g. Model Interfaces

- Requirement: Introduce interfaces for additional models.

- Loading custom models from sources other than Hugging Face.

- Fine-tuning models, loading fine-tuned models, and performing experiments.

- Functionality: Users must be able to specify model parameters and configurations.

h. Automated Labeling

- Requirement: Implement automated labeling of clusters.

- Utilize LLM labeling with DSPY2 for automatic prompt improvement.

- Functionality: A GUI must be available for interacting with and editing existing datasets.

- Users must be able to define labeling strategies and incorporate manual labels as necessary.

i. Probing and Reverse Feature Engineering

- Requirement: Include functionalities for probing and reverse feature engineering.

- Functionality: Users must be able to extract meaningful features and relationships from latent representations to improve model understanding and performance.

2.1.2 Usability Requirements

a. Documentation

- Requirement: Provide comprehensive documentation, including:

- Installation instructions.

- Usage examples.

- API references.

b. Example Scripts

- Requirement: Include example scripts demonstrating typical use cases.

2.1.3 Performance Requirements

a. Efficiency

- Requirement: The library must handle large datasets efficiently, minimizing computational resource usage.

- Focus on optimizing clustering algorithms and data handling processes.

2.1.4 Testing Requirements

a. Unit Tests

- Requirement: Include unit tests for all major functionalities to ensure robustness and reliability.

b. Code Coverage

- Requirement: Achieve at least 60% code coverage in testing (constraint).

2.1.5 Constraints

a. Code Coverage

- Requirement: The library must achieve a minimum of 60% code coverage in its testing suite (constraint).

b. OpenMP Support

- Requirement: The library must support OpenMP code, even though Tree-sitter does not currently support it (constraint).

2.2. Engineering Standards

2.2.1 Why Standards?

Engineering standards are important, because they provide specific, detailed methods of using tools in a safe and effective way. By providing common guidelines and specifications, these standards help engineers create designs that are consistent in quality and can work seamlessly with other products or systems, even if produced by different manufacturers. In addition to increased performance and safety, engineering standards have the potential to reduce costs by preventing the need to reinvent solutions, and promote innovation by allowing engineers to build upon established foundations.

2.2.2 Key Standards

a. ISO/IEC 25010:2011 - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models

- Description: This standard defines a quality model for software and systems. It outlines characteristics such as functionality, reliability, usability, efficiency, maintainability, and portability. The model provides a structured approach to specifying and evaluating software product quality.

- Intended Purpose: The standard aims to ensure that software products meet the required quality levels by providing a common framework for developers and stakeholders to assess and improve software quality throughout the development lifecycle.

b. ISO/IEC TR 24027:2021 - Information Technology - Artificial Intelligence (AI) - Bias in AI Systems and AI-Aided Decision Making

- Description: This technical report addresses the identification and mitigation of bias in AI systems. It discusses types of biases that can occur in data and algorithms, their sources, and the impact they may have on AI decision-making processes.

- Intended Purpose: The standard aims to promote fairness and transparency in AI systems by providing guidelines to detect, evaluate, and reduce bias, thereby enhancing the trustworthiness and ethical considerations of AI applications.

c. IEEE 1028-2008 - Standard for Software Reviews and Audits

- Description: This standard specifies the process for conducting software reviews and audits, including inspections, walkthroughs, technical reviews, and management reviews. It defines the roles, procedures, and outcomes expected from each type of review.

- Intended Purpose: The standard intends to improve software quality and project management by providing structured methods for detecting and correcting defects, ensuring compliance with requirements, and enhancing communication among team members.

2.2.3 Relevance

a. ISO/IEC 25010:2011

- Relevance: Our project involves developing software tools for auto-labeling datasets and evaluating LLMs. Applying this standard helps us define and measure the quality attributes of our software products, ensuring they meet user and stakeholder expectations.

- Specific Impact: By focusing on characteristics like reliability and usability, we can create robust tools that are easy to use and maintain, which is crucial for the success and adoption of our project deliverables.

b. ISO/IEC TR 24027:2021

- Relevance: Since our project deals with AI systems and decision-making processes, addressing bias is essential. This standard provides us with guidelines to identify and mitigate biases in our models and datasets.

- Specific Impact: Incorporating this standard ensures that our evaluations of LLMs are fair and unbiased, which enhances the credibility and ethical standing of our project outcomes.

c. IEEE 1028-2008

- Relevance: Effective communication and quality assurance are vital in our project, especially with multiple team members working on different components. This standard offers structured review processes to improve collaboration and software quality.

- Specific Impact: By conducting regular software reviews and audits as per the standard, we can identify defects early, ensure compliance with requirements, and facilitate better teamwork.

2.2.4 Other Standards

In discussing with our team, other standards were considered, but ultimately are not being prioritized.

a. IEEE 829-2008 - Standard for Software and System Test Documentation

- Description: This standard provides a set of documents that cover the testing process, including test plans, designs, procedures, and reports.

- Reasoning: This standard could potentially be important for structuring our testing documentation to ensure thorough validation of our software components.

b. SO/IEC 27001:2022 - Information Security, Cybersecurity and Privacy Protection — Information Security Management Systems — Requirements

- Description: This standard specifies requirements for establishing, implementing, maintaining, and continually improving an information security management system.

- Reasoning: Given that we handle code datasets, some of which may be sensitive, applying this standard would help in securing our data and managing risks related to information security.

While we are not fully adopting IEEE 829-2008 and ISO/IEC 27001:2022, we will still incorporate essential elements from these standards where applicable to maintain good practices in testing documentation and data security.

2.2.5 Implementation

a. ISO/IEC 25010:2011

- Quality Attribute Specification: We will define specific quality attributes for our software, such as usability and reliability metrics for the auto-labeling pipeline.

- Quality Assessment: Develop procedures to assess these attributes regularly, using tools and techniques like user testing and performance monitoring.

- Documentation: Maintain detailed documentation of quality requirements and assessment results to guide ongoing improvements.

b. ISO/IEC TR 24027:2021

- Bias Identification Process: Establish a process to identify potential biases in our datasets and models, including regular audits and reviews of data sources and labeling practices.

- Bias Mitigation Strategies: Implement techniques such as data balancing, algorithm adjustments, and inclusion of diverse data samples to reduce bias.

- Transparency Measures: Document and communicate our efforts to address bias, enhancing the transparency and ethical considerations of our project.

c. IEEE 1028-2008

- Structured Reviews: Schedule regular software inspections and walkthroughs to identify defects and areas for improvement.

- Audit Trails: Keep detailed records of review findings, decisions made, and actions taken to provide accountability and track progress.

3 Project Plan

3.1 Project Management/Tracking Procedures

Our team adopted the Agile methodology due to the adaptive nature of our project. Agile's iterative cycles and emphasis on flexibility aligned well with our field, where evolving customer needs and shifting requirements required an adaptable approach. Agile allowed us to respond to changes efficiently without disrupting significant portions of the project.

To track our progress, we utilized GitLab, which provided detailed task decomposition, tracking, and workflow automation through its CI/CD pipelines. GitLab also offered wikis and robust issue-tracking capabilities, facilitating organized project documentation and streamlined development processes. For remote, asynchronous communication, we relied on a dedicated Discord channel, enabling our team to stay connected and aligned despite varying schedules.

3.2 Task Decomposition

We utilized GitLab's built-in features for task decomposition. Epics represented overarching project requirements and goals, which were further broken down into Milestones capturing key functional components and deliverables. Each Milestone was subdivided into Issues—specific, actionable tasks assigned to team members to complete Milestones.

We tracked Issues on an issue board, assigning each Issue to relevant Milestones and Epics. This hierarchical organization helped us manage interdependencies among tasks, systematically addressing all project aspects. Progress on Issues was continuously monitored within GitLab, providing a clear log of task completion and overall project advancement.

We mapped the linear progression of completed requirements to our sprint cycles, maintaining alignment with Agile methodology, ensuring that the project's development remained responsive to changes.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

In our Agile development process, milestones were refined iteratively with each sprint, integrating regular progress updates and customer feedback. Each Agile cycle comprised two main sprints:

- **Design Sprint**: Focused on implementing feedback from the previous cycle and planning upcoming tasks. We reviewed and adjusted designs to align with customer input, creating a roadmap for implementation.
- **Implementation Sprint (MVP)**: Updated design documentation based on customer feedback and implemented planned tasks. This included creating and testing minimum viable products meeting customer requirements.

Tasks were considered complete once they successfully met sprint-specific customer requirements. Progress metrics were issue-based, with individual Issues and Milestones serving as checkpoints. This allowed accurate progress measurement, ensured quality, and refined milestones iteratively based on evolving requirements.

Due to the adaptive nature of this project, precise success metrics were developed iteratively. Much of our work involved defining metrics tailored specifically to LLMs and generative AI models. As these metrics evolved through ongoing customer feedback, they guided milestone evaluation and overall project success.

3.4 Project Timeline/Schedule

Due to the project's nature and Agile adoption, our project timeline remained flexible. Essential features were prioritized, while some requirements moved in or out of scope as necessary.

Image 3.4-a and 3.4-b illustrates our timeline for completing subtasks, consistent with Agile practices. Critical implementation tasks were defined and tracked within GitLab.

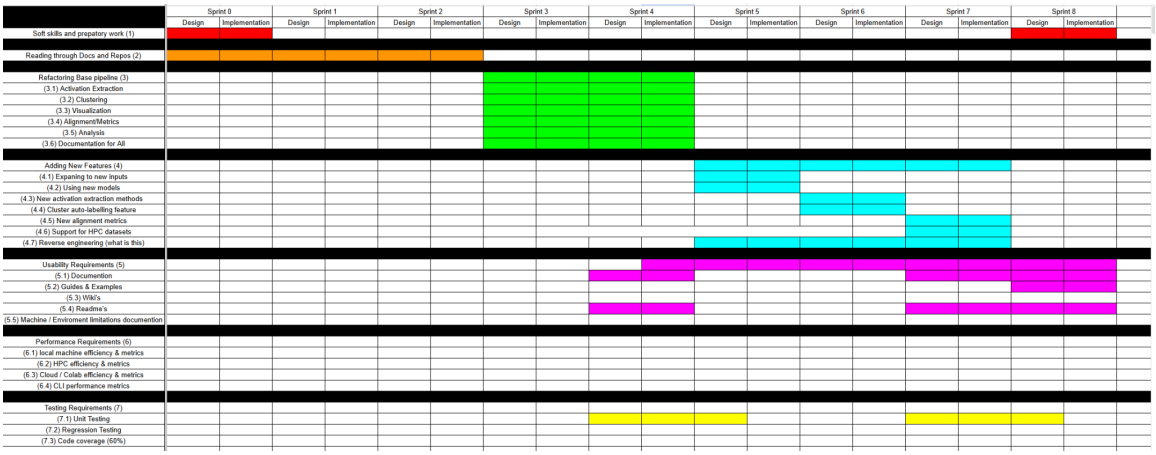


Image 3.4-a: 4910 Gantt Chart

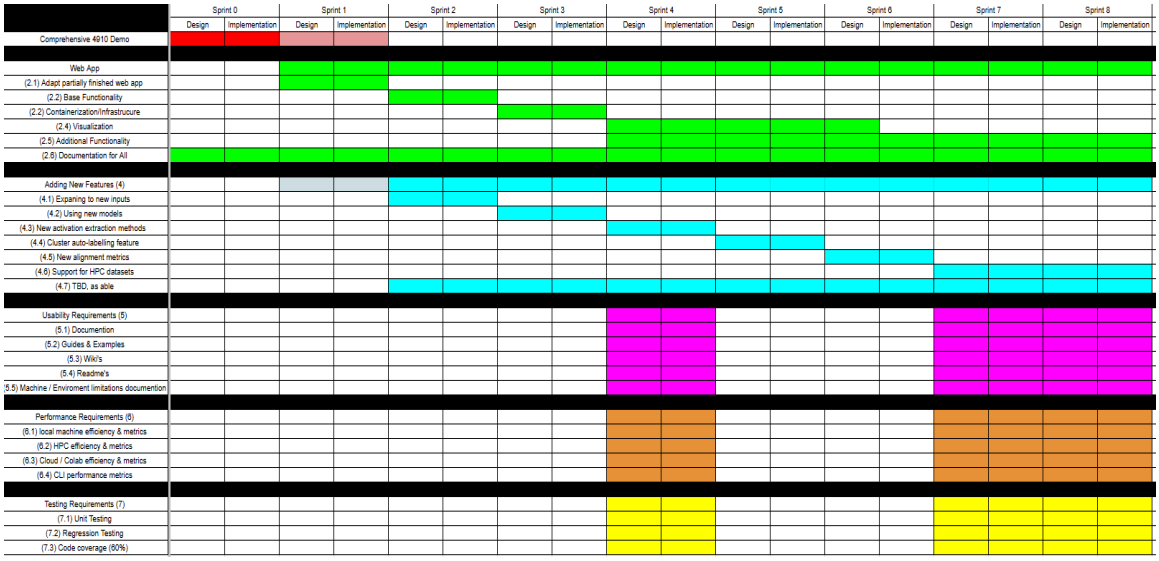


Image 3.4-b: 4920 Gantt Chart

Unit Testing
#9 · created 6 days ago by frost2
To Do

Pipeline Integration
#8 · created 6 days ago by kbouwman Nov 7, 2024
To Do

HPC Start
#7 · created 6 days ago by kbouwman Nov 7, 2024
To Do

Auto Labelling
#6 · created 6 days ago by kbouwman Nov 7, 2024
To Do

Activation
#5 · created 6 days ago by kbouwman Nov 7, 2024
To Do

Clustering interactive Visualizations
#4 · created 6 days ago by kbouwman Nov 7, 2024
To Do

Configuration for Pipeline Automation
#3 · created 6 days ago by kbouwman Nov 7, 2024
To Do

Clustering Analysis
#2 · created 6 days ago by kbouwman Nov 7, 2024
To Do

Clustering
#1 · created 6 days ago by kbouwman Nov 7, 2024
To Do

Image 3.4-c: GitLab Issues

3.5 Risks and Risk Management/Mitigation

We identified several risks potentially impacting the project's success and delivery timeline, including rapid advancements in artificial intelligence, technical challenges with AST tools and LLMs, and time constraints. Each risk was analyzed with a probability estimate and a comprehensive mitigation plan developed accordingly. For high-probability or high-impact risks, we established alternative solutions, ensuring project resilience.

1. Risk of Rapid Advancements in AI Rendering Current Approaches Obsolete

- **Risk Probability:** 0.6
- **Mitigation:**
    - Conducted continuous monitoring of AI research through weekly discussions. Agile allowed project adjustments to incorporate beneficial emerging technologies promptly.

2. Technical Challenges with AST Tools or Large Language Models (LLMs)

- **Risk Probability:** 0.7
- **Mitigation Plan:**
    - Engaged in early prototyping and thorough testing. Maintained alternative tools ready to ensure project continuity and effectiveness in case primary tools encountered issues.

3. Time Constraints Impacting Project Delivery and Quality

- **Risk Probability:** 0.8
- **Mitigation Plan:**
    - Adhered strictly to incremental Agile development with clearly defined sprints. Established regular communication loops with clients and team members to proactively manage scope and address issues swiftly, ensuring project timelines were maintained without sacrificing quality.any deviations from the original plan are identified and addressed swiftly, keeping the project on track.

3.6 Personnel Effort Requirements

Due to Agile practices and the evolving project scope, precise task-based person-hour estimates were challenging. Instead, we estimated efforts using a general sprint-based framework:

Table 3.6-a

Design Sprint (1 week)	Time (hrs /person)	Description
Feedback	1-2	Taking client feedback, and implementing requested changes.
Design	4-5	Researching Technologies, choosing tools, building an implementation plan
Implementation	2-3	Starting to write initial code, finding pain-points to discuss before the implementation sprint

Table 3.6-b

Implementation Sprint (1 week)	Time (hrs / person)	Description
Feedback	1-2	Taking client feedback, and implementing requested changes.
Design	1-2	Further Research, discussion within team
Implementation	5-6	Writing code, active testing, building functional subsets of final deliverable

3.7 Other Resource Requirements

Critical resources utilized included Iowa State's Nova HPC clusters, crucial for testing in realistic HPC environments used by our intended users. Additionally, we utilized ECpE's (Electrical and Computer Engineering) GitLab, providing enhanced security and specialized functionality. Access to current research and projects on LLMs or DNNs was imperative, as each new analytical method potentially enhanced our pipeline's capabilities and clarity of results.

4 Design

4.1 Design Context

4.1.1 Broader Context

Our project was situated at the intersection of software engineering and artificial intelligence, enhancing the explainability and interpretability of large language models (LLMs) used in code analysis and generation. Our intended audience included machine learning researchers, prompt

engineers, HPC researchers, and software developers who utilize AI models for code-related tasks. By improving AI transparency, our project benefits the broader software development community and indirectly impacts end-users relying on reliable and secure software applications.

Table 4.1.1

Area	Description	Examples
Public health, safety, and welfare	Our project enhances the reliability and safety of software systems by making AI models more interpretable. By reducing software errors, especially in safety-critical applications, we positively affect the well-being of users and communities dependent on such software. Improved code understanding leads to more secure and efficient software solutions.	- **Reducing risks in medical devices:** By enabling developers to understand AI-generated code, we help prevent software failures in medical equipment, ensuring patient safety. - **Enhancing automotive safety:** Improved AI interpretability aids in developing safer autonomous driving systems. - **Improving cybersecurity:** Better understanding of AI models helps identify and mitigate vulnerabilities in software.
Global, cultural, and social	By promoting transparency and ethical AI practices, our project aligns with global efforts to foster responsible AI development. It supports developers worldwide, regardless of cultural or language backgrounds, in understanding AI tools, thus encouraging inclusivity and collaboration in the global tech community.	- **Facilitating international collaboration:** Tools that improve AI explainability can be used by developers globally, bridging language and cultural gaps. - **Supporting ethical standards:** Our project helps adhere to professional codes of ethics by making AI decisions transparent. - **Building trust in AI:** Enhancing interpretability reduces the "black box" perception, fostering societal trust in AI technologies.
Environmental	While our project involves computationally intensive tasks that may increase energy consumption, it also has the potential to reduce the environmental impact of software development in the long term. By improving code efficiency and reducing development time, we contribute to lowering the overall environmental footprint associated with software processes.	- **Energy consumption in HPC clusters:** High-performance computing required for model analysis increases energy usage. - **Reducing development cycles:** Improved tools can lead to faster development and less resource-intensive processes. - **Promoting efficient coding practices:** By understanding AI models better, developers can write

		more efficient code, indirectly reducing energy consumption.
Economic	Our project can lead to significant cost savings in software development by enhancing developer productivity and reducing time spent on debugging and code comprehension. It also opens up new economic opportunities in the market for AI explainability tools and services, potentially contributing to job creation and economic growth in the tech industry.	- **Lowering development costs:** By reducing debugging time, companies save resources. - **Accelerating time-to-market:** More efficient development processes enable faster product releases, increasing competitiveness.. - **Economic growth:** Improved tools can boost productivity across the industry, contributing to economic advancement.

4.1.2 Prior Work/Solutions

Background and Literature Review

The field of explainable artificial intelligence (XAI) has become increasingly important, especially regarding large language models (LLMs) trained on source code. Understanding how these models make decisions is crucial for developers to trust and effectively utilize them in software engineering tasks. Several prior works have attempted to address the challenges of interpreting and analyzing neural networks in the context of code.

One significant tool in this domain is **NeuroX**, a toolkit designed for analyzing individual neurons in neural networks by extracting and visualizing activations [1]. NeuroX facilitates the examination of hidden representations within neural networks, aiding in the interpretation of model behaviors. However, while NeuroX provides general tools for neural network analysis, it is not specifically tailored for code-based LLMs.

Another relevant project is **Code2Vec**, which introduces a method for representing code snippets as continuous distributed vectors and utilizes attention mechanisms to highlight important code elements influencing the model's predictions [2]. Code2Vec focuses on code classification tasks and offers some level of interpretability through attention weights but does not provide a comprehensive framework for latent concept analysis in LLMs.

Additionally, **LIME (Local Interpretable Model-agnostic Explanations)** has been used to explain predictions of machine learning classifiers, including in some code analysis contexts [3]. LIME approximates the model locally with an interpretable model to explain individual predictions. However, its application to deep learning models handling source code is limited due to the complexity and high dimensionality of such models.

Similar Products and Solutions

- **GitHub Copilot**: An AI coding assistant developed by GitHub and OpenAI that suggests code snippets and functions based on the context. While Copilot is powerful in code generation, it lacks transparency in how suggestions are generated, providing minimal insight into the underlying decision-making process of the model.
- **DeepExplain**: A unified framework that implements various gradient-based attribution methods for explaining deep neural networks [4]. While DeepExplain offers tools for model interpretability, it is primarily focused on image data and does not directly address code-specific challenges.

Advantages and Shortcomings of Prior Work

NeuroX

- Pros:
    - Provides tools for neuron-level analysis of neural networks.
    - Supports activation extraction and visualization.
- Cons:
    - Not specifically designed for code-based models.
    - Limited support for handling source code structures and syntax.

Code2Vec

- Pros:
    - Introduces code representation techniques using distributed vectors.
    - Utilizes attention mechanisms for partial interpretability.
- Cons:
    - Focused on code classification tasks.
    - Does not offer comprehensive tools for activation extraction or clustering in LLMs.

LIME

- Pros:
    - Model-agnostic and can be applied to any classifier.
    - Helps explain individual predictions.
- Cons:
    - Less effective with complex, high-dimensional models like LLMs.
    - Limited applicability to sequential and structured data like source code.

Differentiation of Our Project

Our project addressed existing gaps by developing a specialized library emphasizing interpretability specifically for LLMs trained on source code. We successfully delivered a comprehensive solution comprising activation extraction, efficient clustering algorithms, automated labeling, and visualization tools tailored explicitly for code analysis.

Pros of Our Solution:

- **Code-Specific Design**: Specifically handles source code intricacies, syntax, and semantics across multiple programming languages.
- **Comprehensive Pipeline**: Provides an integrated workflow from data parsing to visualization.
- **Support for Advanced Features**: Includes preprocessing for OpenMP directives and optimized compatibility with HPC environments.
- **Automated Labeling**: Employs LLMs and DSPY2 for scalable automatic labeling.

Cons of Our Solution:

- **Development Maturity**: Initial limitations in features compared to established libraries.
- **Resource Requirements**: Significant computational resources needed for large-scale processing.

Works Cited:

[1] V. Dalvi, A. Gautam, N. Durrani, H. Sajjad, Y. Belinkov, and J. Glass, "NeuroX: A Toolkit for Analyzing Individual Neurons in Neural Networks," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, Minneapolis, MN, USA, Jun. 2019, pp. 169–174.

[2] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning Distributed Representations of Code," in *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, Article 40, Jan. 2019.

[3] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?' Explaining the Predictions of Any Classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 1135–1144.

[4] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Towards Better Understanding of Gradient-Based Attribution Methods for Deep Neural Networks," in *6th International Conference on Learning Representations (ICLR)*, Vancouver, Canada, Apr. 2018.

4.1.3 Technical Complexity

Our project demonstrated substantial technical complexity through the integration of multiple sophisticated components, each utilizing distinct scientific, mathematical, and engineering principles. These modules collectively addressed challenges at or above current industry standards.

Data Input Module

- **Principles Involved:** Compiler Design, Abstract Syntax Trees (ASTs)

- **Description:** Parses and preprocesses various source code formats—including raw files, OpenMP code, and structured data (CSV/JSON)—using Tree-sitter to create syntax-aware ASTs, enabling structured code analysis across multiple languages.

Model Interface Module

- **Principles Involved:** Neural Networks, Deep Learning, Model Management
- **Description:** Provides a flexible interface for loading and configuring neural network models from repositories (e.g., Hugging Face) or custom-trained sources, facilitating fine-tuning and tailored analysis.

Activation Extraction Module

- **Principles Involved:** Neural Networks, Deep Learning, High-Performance Computing
- **Description:** Extracts neuron activation data from specified layers of large language models (LLMs), optimized for efficient performance in local and HPC environments to ensure scalability with extensive datasets.

Clustering Module

- **Principles Involved:** Cluster Analysis, High-Dimensional Data Processing
- **Description:** Implements diverse clustering algorithms, including K-means, BIRCH, Agglomerative, Leaders, and Hyperbolic Clustering, to group activation patterns representing latent concepts within neural network models.

Alignment and Metrics Module

- **Principles Involved:** Statistical Analysis, Evaluation Metrics
- **Description:** Employs statistical metrics such as purity scores and adjusted Rand indices to evaluate the quality and alignment of generated clusters against labeled datasets, enabling detailed and robust performance assessments.

Analysis and Visualization Module

- **Principles Involved:** Data Visualization, Human-Computer Interaction, Web Development
- **Description:** Provides interactive visualizations, notably sunburst and dendrogram diagrams, generated using Dash, Plotly, and SciPy within a Python web application built using Flask and deployed via Docker. These visualizations help users intuitively explore hierarchical relationships and interpret complex cluster structures.

Automated Labeling Module

- **Principles Involved:** Natural Language Processing (NLP), Prompt Engineering
- **Description:** Automates cluster labeling using large language models (LLMs) and optimized prompts (via DSPY2), complemented by a GUI allowing interactive refinement,

significantly reducing manual effort and enhancing interpretability.

Probing and Reverse Feature Engineering Module

- **Principles Involved:** Feature Engineering, Interpretability Analysis, Neural Networks
- **Description:** Enables users to systematically probe and analyze neuron activations, facilitating the extraction and reverse-engineering of meaningful latent features to enhance understanding of model decision-making and concept interpretation.

Challenging Requirements:

- **Model Interpretability:** Enhancing transparency of LLMs trained on code, addressing the inherent challenge of understanding "black box" AI behavior.
- **Automated Dataset Labeling:** Developing precise, scalable automated labeling methodologies using NLP and prompt engineering techniques.
- **Multi-Language Parsing:** Extending robust parsing capabilities across diverse programming languages and syntactic constructs, including OpenMP directives.
- **Performance and Scalability:** Optimizing computational efficiency and scalability, particularly within HPC environments.

Justification:

The integration of these modules demands expertise in advanced machine learning, statistical evaluation, compiler theory, visualization techniques, and HPC methodologies, collectively representing significant technical sophistication. The project's comprehensive scope positions it firmly at the intersection of current academic research and cutting-edge industry standards.

4.2 Design Exploration

4.2.1 Design Decisions

Consolidating Distributed Libraries

Initially, foundational code was dispersed across multiple locations. We consolidated these into a single standardized library for enhanced maintainability and scalability, facilitating easier collaboration and future expansion.

Developing a Standardized Library

We standardized the library structure and coding style to enable seamless integration of future updates and uniform application of optimizations, ensuring project scalability.

Defining Machine Testing Parameters

We established testing parameters early in the process for compatibility across local and distributed computing environments, optimizing the library's performance and reliability.

4.2.2 Ideation

To support OpenMP code parsing—since Tree-sitter lacks native support—we brainstormed potential solutions and identified five options:

- Extend Tree-sitter with OpenMP Grammar: Develop a custom grammar to include OpenMP constructs.
- Use an Alternative Parsing Library: Find a different parser that natively supports OpenMP.
- Preprocess Code to Remove OpenMP Directives: Strip OpenMP directives before parsing.
- Integrate Compiler Front-Ends (e.g., Clang): Use compiler tools to parse OpenMP code.
- Combine Tree-sitter with an OpenMP-Specific Parser: Use Tree-sitter alongside a specialized OpenMP parser.

4.2.3 Decision-Making and Trade-Off

We evaluated each option based on implementation complexity, maintenance, performance, and compatibility. Option 3—Preprocess Code to Remove OpenMP Directives—was chosen because it is simple to implement, maintains compatibility with our existing system, and requires minimal maintenance. While it may exclude OpenMP-specific details, this approach efficiently allows us to support OpenMP code within our project's constraints.

4.3 Final Design

4.3.1 Overview

We developed a structured, adaptable Python library supporting scalable software solutions. The library adhered to PEP8 style guidelines and ISO 27001 standards for robust security. It is installable, customizable, and equipped with a versatile Command Line Interface (CLI). The library was optimized for both local and high-performance computing environments, ensuring efficiency in complex, data-intensive workflows.

4.3.2 Detailed Design and Visuals

Our project delivered a comprehensive Python library tailored for latent concept analysis of source code, seamlessly integrating various modules to provide a complete analytical pipeline. The detailed architecture of the system consisted of several interconnected components, each serving a specific role within the pipeline.

Data Input Module:
The pipeline began with the Data Input Module, designed to efficiently handle various data

formats. This module allowed users to load and preprocess datasets, including raw source code files, OpenMP code (preprocessed to ensure compatibility with Tree-sitter), and structured datasets provided in CSV or JSON formats. Once loaded, the data was parsed into Abstract Syntax Trees (ASTs) using the Tree-sitter parser, enabling structured, syntax-aware analysis of source code across multiple programming languages.

Model Interface Module:
The Model Interface Module provided a flexible and robust mechanism for loading neural network models. Users could select from pre-trained models available from repositories like Hugging Face, or load custom models they had trained independently. Additionally, this module offered support for fine-tuning and adjusting model parameters, enabling tailored model configurations to meet specific analytical requirements.

Activation Extraction Module:
After models were loaded, the Activation Extraction Module systematically extracted neuron activation data from specified layers within these neural network models. These activations, representing latent representations learned by the models, served as inputs for subsequent analysis steps. The extraction process was optimized to run efficiently both locally and on HPC environments, ensuring scalability.

Clustering Module:
Extracted activation data was then processed by the Clustering Module, which implemented multiple clustering algorithms, such as K-means, BIRCH, Agglomerative Clustering, Leaders Clustering, and Hyperbolic Clustering. These algorithms grouped similar activation patterns into clusters, representing latent concepts or code features learned by the neural network models. The selection of algorithms allowed users to choose the most appropriate clustering method for their specific dataset characteristics and analytical goals.

Alignment and Metrics Module:
This module provided quantitative analysis to assess the effectiveness and meaningfulness of the generated clusters. It compared clustering results against ground truth datasets, applying statistical metrics such as purity scores and adjusted Rand indices to measure alignment and cluster quality. Users could configure and specify which metrics to apply, facilitating detailed and customized evaluations of clustering performance.

Analysis and Visualization Module:
To help interpret and explore clustering results, the Analysis and Visualization Module provided a suite of interactive visual tools. Users can generate sunburst and dendrogram charts to visually assess the relationships between clusters, examine the distribution and quality of clusters, and identify meaningful patterns. This graphical interface significantly improved usability, making the results more accessible and actionable for users. This interface was delivered in the form of a Python web application.

The web application was built using Python and Flask, deployed via Docker containers hosted on Iowa State University's servers. It employs Dash, Plotly, and SciPy libraries to generate interactive visualizations, notably sunburst and dendrogram diagrams. These visualizations allow users to

intuitively explore hierarchical relationships and cluster structures within activation data, significantly enhancing the interpretability of latent concepts derived from large language models.

Automated Labeling Module:
The Automated Labeling Module leveraged large language models (LLMs), specifically employing DSPY2 for optimized prompt generation. This approach automated the labeling process for clusters, dramatically reducing manual effort and improving scalability. Additionally, a user-friendly graphical user interface (GUI) allowed for interactive refinement of automatically generated labels, enabling users to incorporate domain-specific knowledge or correct inaccuracies as needed.

Probing and Reverse Feature Engineering Module:
Lastly, this module allowed users to delve deeper into model interpretability by enabling extraction of meaningful features from latent representations. By systematically probing activations and analyzing their relationships, users could reverse-engineer features, gaining deeper insights into model decision-making processes and enhancing overall understanding of how the model interprets and generates source code.

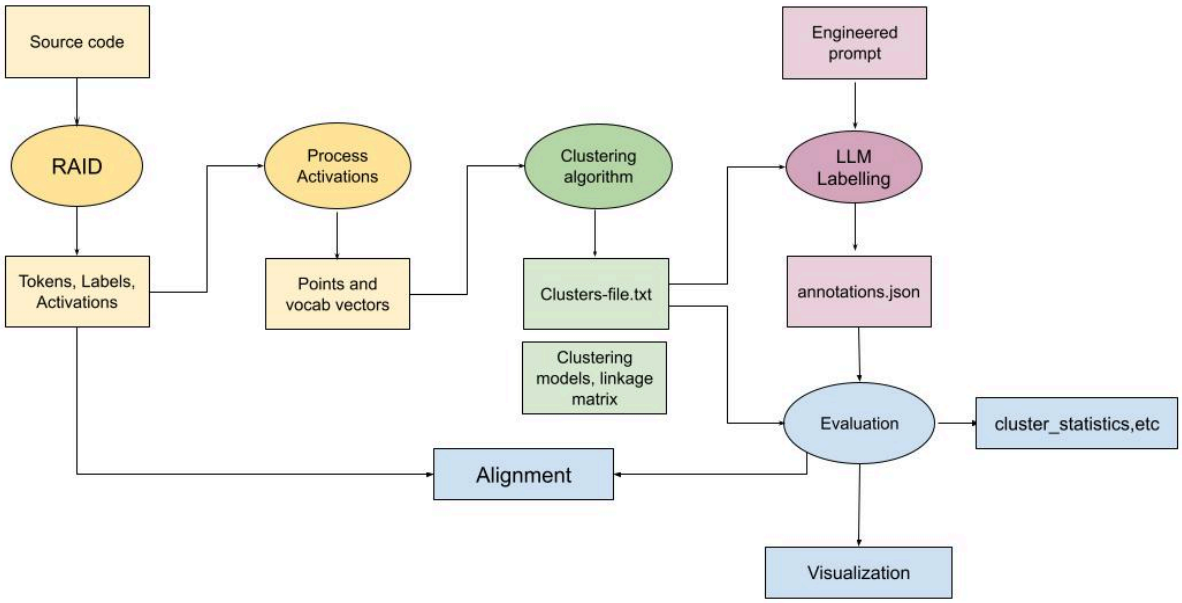


Image 4.3.2-a: Module Pipeline

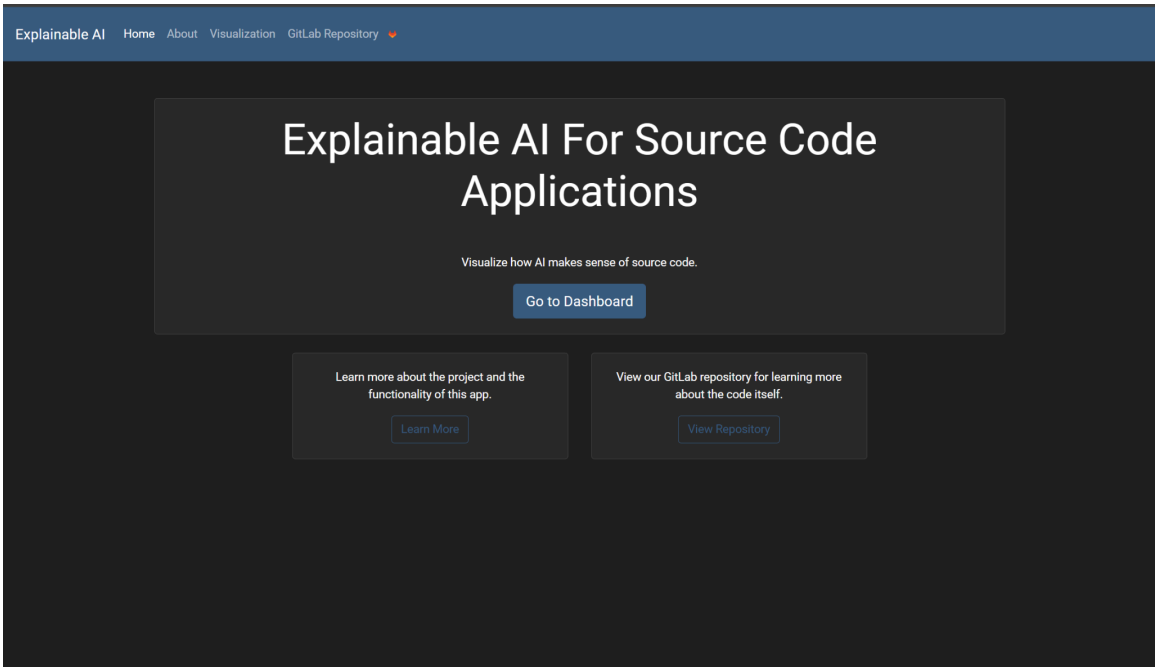


Image 4.3.2-b: Web Application Home Page

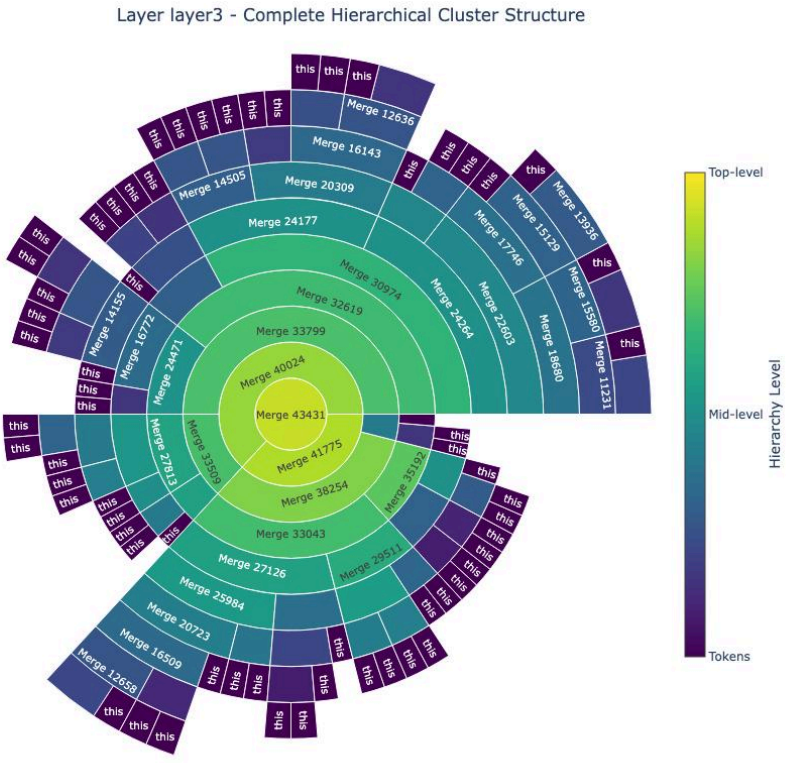


Image 4.3.2-c: Sunburst Visualization Example

4.3.3 Functionality

We provided users with three primary interaction modes: local installation, cloud execution (e.g., Google Colab), and execution on Nova HPC clusters.

Installation

To install, use the following command:

```
pip install cocoa
```

Packaging the Project

To generate a pip-installable tarball and wheel file:

1. Run the following command to create the distribution files:

   ```
   python3 setup.py sdist bdist_wheel
   ```

2. You will find the `.tar.gz` and `.whl` files in the `dist/` directory.

Installing the Package

To install the package from the generated wheel file, use:

```
pip install dist/cocoa-0.0.1-py3-none-any.whl
```

Usage

Once installed, you can use the command-line interface. For example, to extract activations:

```
cocoa extract_activations --model bert-base-uncased --input input.txt
--output output.txt
```

This will run the activation extraction from the specified model and save the results to the output file.

4.3.4 Areas of Challenge

We encountered a variety of obstacles as the project progressed, particularly around managing the sheer volume of activation data and ensuring our system remained responsive across local, cloud, and HPC environments. Fine-tuning our automated labeling process introduced variability that

required repeated adjustments to prompts and occasional manual review to maintain quality. Extending our parsing framework to handle unconventional or emerging code constructs also proved nontrivial, necessitating additional preprocessing steps and tool refinements. Balancing the complexity of clustering and alignment metrics with the need for clear, human-interpretable results led to ongoing experimentation with different evaluation strategies. Finally, coordinating dependencies and resource allocation across diverse platforms highlighted the importance of robust configuration management and scalable deployment practices.

4.4 Technology Considerations

We integrated several distinct technologies, carefully evaluating their strengths, weaknesses, and trade-offs:

- **Nova HPC Clusters**: Provided scalability for analyzing large datasets.
- **Neural Network Models**: Enabled flexible model loading and customization.
- **Clustering Algorithms**: Offered versatile algorithm choices suitable for varying data structures.
- **Data Parsing with Tree-sitter**: Facilitated comprehensive language support despite limitations with OpenMP directives.
- **Automated Labeling (LLMs & DSPY2)**: Enhanced labeling efficiency despite potential external dependencies.

Strengths:

- Efficiency: Accelerates the labeling process, reducing manual effort.
- Quality Improvement: DSPY2 enhances prompt effectiveness, leading to better labeling outcomes.

Weaknesses:

- Bias Risk: LLMs may introduce biases into the labeling process.
- Dependence on External Models: Reliance on LLMs may pose challenges if models change or become unavailable.

Trade-offs:

- Increased efficiency versus potential bias and reliance on external services.

5 Testing

Testing was a critical component throughout our project's lifecycle, ensuring each aspect of the software library functioned correctly and adhered to specified requirements. We executed a comprehensive testing strategy tailored explicitly to our project design and requirements. Frequent,

automated testing aligned closely with our Agile development methodology, enabling prompt issue identification and resolution.

Our testing approach centered on test automation using continuous integration and continuous deployment (CI/CD) pipelines with GitLab runners. This practice maintained code quality, ensured at least 60% code coverage, and prevented new code from breaking existing functionality.

Unique testing challenges included handling large datasets, integrating various machine learning models, and ensuring compatibility with HPC environments. We addressed these with specialized tests to validate performance, scalability, and correctness across different computing environments.

5.1 Unit Testing

Unit testing verified individual functions and methods within modules, ensuring isolation of each component's correctness.

Implementation:
Within the `../src` directory, two subdirectories were established:

- **cocoa:** Contains library source code.
- **cocoaTest:** Mirrors `cocoa`'s structure, containing corresponding unit tests.

Tests utilized Python's `unittest` framework, with each module in `cocoa` paired with a dedicated test module in `cocoaTest`.

Coverage Goals:
We achieved our stated goal of at least 60% code coverage using tools like `coverage.py` integrated into the `unittest` framework.

5.2 Interface Testing

Interface testing validated correct interactions between various modules and system components.

Interfaces Tested:

- **Data Input and Parsing Interface:** Interaction between data loading and parsing modules.
- **Model Interface:** Interaction between user-defined models and activation extraction.
- **Clustering Interface:** Flow of activations into clustering algorithms.
- **Evaluation Interface:** Transmission of clustering results to alignment and metrics module.

- **Automated Labeling Interface:** Integration between clustering modules and automated labeling with LLMs.

Testing Approach:

- Employed integration tests to confirm accurate data transfer between modules.

5.3 Integration Testing

Integration testing ensured modules functioned collectively, focusing on critical paths.

Critical Integration Paths Tested:

- **End-to-End Data Flow:**
  Data input → Parsing → Activation extraction → Clustering → Labeling → Evaluation.
- **HPC Environment Compatibility:**
  Integration tests performed on Nova HPC clusters to validate distributed computing functionality.

Testing Strategy:

- Simulated real-world use cases with representative datasets.
- Automated scripts execute tests locally and on HPC environments.
- Outputs at each stage were compared against expected results.

Tools Utilized:

- GitLab CI/CD pipelines automated integration tests.

5.4 System Testing

System testing evaluated the fully integrated system to confirm overall compliance with project requirements.

System-Level Testing Strategy:

- Conducted realistic full-scale tests processing large code datasets, supporting multiple programming languages, and generating comprehensive analysis reports.
- Verified compliance with non-functional requirements: performance, scalability, and usability.

Requirements Coverage:

- **Functional Requirements:** Verified each specified functionality (activation extraction, clustering, labeling, visualization, etc.).
- **Usability Requirements:** Evaluated the command-line interface (CLI) for user-friendliness and reliability.
- **Performance Requirements:** Monitored processing times and resource utilization to ensure efficiency targets were met.

Tools Utilized:

- Automated testing frameworks and monitoring tools collected performance data.
- User acceptance testing gathered direct user feedback.

5.5 Regression Testing

Regression testing ensured existing functionalities remained stable following updates.

Approach:

- Established automated regression test suites executed upon merging code into the main branch.
- Included comprehensive unit, interface, and integration tests covering established features.

Automation Tools:

- GitLab CI/CD pipelines triggered automated regression tests.

Critical Features Protected:

- Core functionalities (activation extraction, clustering) maintained stability.
- Ensured accuracy and reliability despite ongoing performance enhancements.

5.6 Acceptance Testing

Acceptance testing demonstrated system compliance to the client, confirming adherence to specified requirements.

Process:

- Developed test cases directly from Section 2.1 requirements.
- Collaborated with client (Dr. Ali Jannesari/ISU SwAPP Lab) for test plan reviews and result evaluations.
- Conducted live system demonstrations using representative datasets.

Client Involvement:

- Scheduled regular client meetings to present progress and incorporate feedback.
- Provided client access for independent system evaluation.

Criteria:

- Functional requirements satisfactorily met.
- Performance and usability achieved client expectations.

5.7 User Testing

We conducted user testing primarily through iterative demonstrations and feedback sessions with our faculty advisor. During these sessions, our advisor exercised both the command-line interface and the web-based visualizations—running activation extraction, navigating clusters via the dendrogram and sunburst, and reviewing automatically generated labels.

The advisor's reactions were consistently positive; they confirmed that the tools addressed core interpretability needs and appreciated the intuitive visual layouts. Their interactive exploration revealed minor usability issues—such as unclear button labels and navigation controls—which we promptly refined. Overall, observing the advisor interact with the system validated our design choices and guided final adjustments to improve clarity and workflow efficiency.

5.8 Results

We completed comprehensive testing and achieved the following outcomes:

- **Unit Testing:**
  Successfully identified and resolved defects in clustering algorithms and parsing modules.
- **Integration Testing:**
  Achieved seamless processing of datasets through the full pipeline on local and HPC environments. Addressed and resolved data format compatibility issues across modules.
- **System Testing:**
  Confirmed the system met functional requirements for small to medium datasets.

Performance tests demonstrated acceptable system efficiency.

- **Regression Testing:**
  Implemented comprehensive automated regression tests in GitLab CI/CD pipelines. Successfully identified and corrected regressions, notably within the activation extraction module.
- **Acceptance Testing:**
  Demonstrated system functionality met client requirements. Received positive feedback regarding system performance, usability, and overall quality.
- **Security Testing:**
  Identified no critical vulnerabilities through static code analysis and dependency scanning, affirming robust security practices.

The conducted tests effectively validated compliance with specified requirements, demonstrating that our delivered software meets both functional and non-functional expectations.

6 Implementation

Our implementation focused primarily on adapting and enhancing existing scripts from the CodeConceptNet library. This refactoring served two core purposes: providing practical assistance to a related team and allowing us to acquire essential familiarity with code concept extraction methodologies central to our project.

Objective:

- Adapt and enhance existing scripts to align fully with project requirements, ensuring improved code quality, modularity, and maintainability

Process:

- **Code Analysis:** Conducted an extensive review of CodeConceptNet scripts, thoroughly documenting existing functionalities.
- **Modularization:** Refactored existing scripts to promote modularity, clarity, and maintainability, laying a robust foundation for future development.
- **Standardization:** Ensured all scripts adhered strictly to PEP8 style guidelines, fostering code consistency throughout the project.

Outcome:

- Established a structured, maintainable, and well-documented codebase aligned with our architectural standards and project goals.
- Gained in-depth insights into methodologies related to activation extraction, clustering algorithms, and automated labeling for latent concept analysis.

6.1 Design Analysis

Our implemented design successfully delivers core functionality with high reliability. Activation extraction and clustering modules consistently produce meaningful groupings, as confirmed by unit and integration tests (>60% coverage) and positive feedback from our advisor during user testing. The modular pipeline structure and HPC integration on Nova ensure efficient processing of moderately large datasets, and our CI/CD–driven deployments demonstrate stability across environments.

However, some components underperform at scale: hierarchical clustering and real-time rendering in the web app can become sluggish with very large activation sets, and prompt-driven automated labeling occasionally produces inconsistent labels without manual correction. These issues stem from late-stage performance tuning and reliance on external LLM services. In future iterations, earlier profiling, incremental clustering strategies, and a caching layer for visualizations would mitigate these limitations.

7 Ethics and Professional Responsibility

Engineering ethics and professional responsibility were integral throughout our project. We adhered rigorously to professional and ethical standards, ensuring our decisions and actions consistently prioritized public welfare, environmental sustainability, and social responsibility.

7.1 Areas of Professional Responsibility/Codes of Ethics

Table 7.1 outlines how our team actively engaged with defined areas of professional responsibility according to the Software Engineering Code of Ethics.

Table 7.1

Area of Responsibility	Definition (in our own words)	Relevant Item from Software Engineering Code of Ethics	Description of Team Interaction
Work Competence	Performing tasks with high quality, integrity, and professional skill.	**Principle 3.01:** "Ensure that their products and related modifications meet the highest professional standards possible."	We rigorously test our code, follow best coding practices, and continually update our skills to deliver a reliable and efficient software library.
Financial Responsibility	Providing valuable products and services	**Principle 5.05:** "Ensure that any	We manage resources efficiently, optimizing

	that are cost-effective.	document upon which they rely has been approved, when required, by someone authorized to approve it."	code to reduce computational costs, especially important when utilizing HPC resources.
Communication Honesty	Sharing information truthfully and clearly with stakeholders.	**Principle 2.05:** "Keep private confidential information gained in their professional work, where such confidentiality is consistent with the public interest and consistent with the law."	We maintain transparent and honest communication within the team and with our client, providing accurate progress reports and addressing issues promptly.
Health, Safety, Well-Being	Minimizing risks to people's safety and well-being in our work.	**Principle 1.02:** "Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life."	By improving AI model interpretability, we contribute to safer software systems, reducing the risk of errors in critical applications.
Property Ownership	Respecting others' property, ideas, and information.	**Principle 1.06:** "Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods, and tools."	We respect intellectual property by using licensed tools and datasets, citing sources appropriately, and not misusing proprietary information.
Sustainability	Protecting the environment and promoting sustainable practices.	**Principle 1.08:** "Be encouraged to volunteer professional skills to good causes and contribute to public education concerning the	We acknowledge the environmental impact of HPC usage and aim to write efficient code to minimize energy consumption.

		discipline."	
Social Responsibility	Creating products that benefit society and communities.	**Principle 1.01:** "Accept full responsibility for their own work."	Our project enhances AI transparency, promoting ethical AI use and building public trust, thereby benefiting society at large.

Performance and Improvement Areas

Performing Well: Communication Honesty
We excelled in maintaining transparent and accurate communication through regular meetings, detailed documentation, and prompt issue resolution, which promoted stakeholder alignment and trust.

Area for Improvement: Sustainability
Recognized the need to better manage environmental impacts of high-performance computing. Future efforts will prioritize algorithmic optimizations for enhanced energy efficiency and environmentally responsible computing practices.

7.2 Four Principles

Table 7.2 maps each broader context area explicitly to four ethical principles—beneficence, nonmaleficence, respect for autonomy, and justice:

Table 7.2

Broader Context Area	**Beneficence** (Do Good)	**Nonmaleficence** (Do No Harm)	**Respect for Autonomy**	**Justice**
Public Health, Safety, and Welfare	Enhancing software reliability to improve public safety.	Preventing software errors that could lead to harm.	Empowering users with understandable AI models.	Providing equitable access to safe technology.
Global, Cultural, and Social	Promoting ethical AI practices worldwide.	Avoiding biases in AI that harm specific groups.	Respecting cultural differences in AI applications.	Ensuring fair benefits of AI across societies.

Environmental	Developing energy-efficient software to benefit the environment.	Minimizing energy consumption to reduce environmental harm.	Supporting stakeholder choices for sustainable options.	Advocating for environmental justice in technology use.
Economic	Reducing development costs to benefit organizations.	Avoiding economic harm through cost-effective solutions.	Allowing clients to make informed financial decisions.	Contributing to economic opportunities across communities.

Important Context-Principle Pair (Beneficence, Public Health, Safety, Welfare):
Actively enhanced public safety through improved AI model interpretability, ensuring rigorous testing and ethical adherence to support safer software applications.

Context-Principle Pair Needing Improvement (Nonmaleficence, Environmental):
Recognized environmental impacts of computationally intensive processes. Moving forward, we will optimize energy usage and adopt sustainable computing practices to mitigate negative effects.

7.3 Virtues

Important Team Virtues

- **Integrity:** Maintained honesty, transparency, and strong moral principles across all actions, fostering trust and accountability within the team.
- **Collaboration:** Promoted effective teamwork and shared responsibilities, ensuring productive cooperation and the achievement of common goals.
- **Excellence:** Pursued the highest quality standards through rigorous testing, continuous improvement, and professional growth.

Our commitment to engineering ethics and professional responsibility has remained consistent throughout the project. We have continuously applied standards such as ISO/IEC TR 24027 for bias mitigation and IEEE 1028 for quality assurance, and conducted regular reviews with our faculty advisor to validate data handling, model interpretation, and resource use. As no significant ethical issues emerged during development, we have maintained and reinforced our original ethical framework through ongoing monitoring, documentation, and collaborative feedback.

8 Conclusions

8.1 Summary of Progress

Throughout our project, we successfully delivered a structured Python library focused on enhancing the explainability and interpretability of large language models (LLMs) trained on source code. Our efforts resulted in:

- **Refactored Existing Code:** Adapted scripts from the CodeConceptNet library to establish foundational infrastructure and domain understanding.
- **Implemented Core Modules:** Delivered comprehensive components, including data parsing with Tree-sitter, activation extraction utilizing NeuroX, advanced clustering algorithms, and an automated labeling module using DSPY2.
- **Integrated High-Performance Computing:** Successfully configured our system to run efficiently on HPC clusters (Nova), addressing performance and scalability demands.
- **Established Comprehensive Testing Frameworks:** Developed and executed robust testing strategies (unit, interface, integration, system, regression, acceptance, security testing), ensuring system reliability and quality through automated CI/CD pipelines.
- **Adhered to Ethical and Professional Standards:** Prioritized public welfare, ethical AI practices, sustainability considerations, and societal responsibility.

Reiterating Project Goals:

Our completed objectives include a scalable, user-friendly library enabling:

- Precise activation extraction from neural network models.
- Application of optimized clustering algorithms to identify latent concepts within source code representations.
- Automated labeling of code datasets, significantly enhancing interpretability.
- Comprehensive evaluation methods aligning machine-learned concepts with human-understandable properties.

Best Plan of Action (Future Directions):

- **Algorithmic Optimization:** Continuously improve the performance and efficiency of activation extraction and clustering algorithms, especially in HPC contexts.
- **Enhanced Testing and Coverage:** Expand beyond baseline testing coverage to ensure sustained robustness and reliability across future enhancements.
- **Improved Documentation and User Experience:** Continuously refine user documentation, enhance API clarity, and optimize the graphical user interface for greater accessibility and ease of use

8.2 Value Provided

Our design effectively addresses user needs by making large language model (LLM) interpretability more accessible, particularly benefiting researchers and developers who rely on transparency to trust AI-generated code. By integrating clustering algorithms, activation visualization, and automated labeling, the solution significantly reduces the complexity of understanding model behaviors. Faculty advisor testing verified our solution successfully uncovers meaningful code-level patterns, demonstrating clear potential to improve code reliability, accelerate debugging processes, and promote safer software development within the broader context of explainable AI.

8.3 Next Steps

Future work should prioritize further optimization of the clustering and visualization processes for large-scale datasets, improving scalability within high-performance computing environments. Additionally, extending support for additional programming languages and integrating enhanced real-time interactive visualizations could greatly expand usability and practical value. Finally, deeper exploration into ethical implications, such as automated bias detection in model activations, would strengthen the broader societal impact of this work.

9 References

[1] V. Dalvi, A. Gautam, N. Durrani, H. Sajjad, Y. Belinkov, and J. Glass, "NeuroX: A Toolkit for Analyzing Individual Neurons in Neural Networks," in *Proc. 2019 Conf. North American Chapter of the Association for Computational Linguistics (Demonstrations)*, Minneapolis, MN, USA, Jun. 2019, pp. 169–174.

[2] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning Distributed Representations of Code," in *Proc. ACM on Programming Languages*, vol. 3, no. POPL, Art. 40, Jan. 2019.

[3] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?' Explaining the Predictions of Any Classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD)*, San Francisco, CA, USA, Aug. 2016, pp. 1135–1144.

[4] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Towards Better Understanding of Gradient-Based Attribution Methods for Deep Neural Networks," in *6th Int. Conf. Learning Representations (ICLR)*, Vancouver, Canada, Apr. 2018.

[5] International Organization for Standardization, *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*, ISO/IEC 25010:2011, 2011.

[6] International Organization for Standardization, *Information Technology—Artificial Intelligence (AI)—Bias in AI Systems and AI-Aided Decision Making*, ISO/IEC TR 24027:2021, 2021.

[7] IEEE, *IEEE Standard for Software Reviews and Audits*, IEEE Std 1028-2008, 2008.10 Appendices

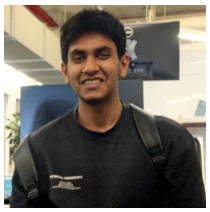
Appendix 1: Operating Manual & Code

Detailed instructions for use of our software, as well as all production code is available for viewing on our GitLab repository.

https://git.ece.iastate.edu/kbouwman/Explainable_AI_for_Source_Code_Applications

Appendix 2: Team Contract

A2.1 Team Members

				
Manjul Balayar	Rayne Wilde	Sam Frost	Akhilesh Nevatia	Ethan Rogers

A2.2 Required Skill Sets for Your Project

Software Engineering and Development
Related Requirements: Develop a scalable, maintainable, and well-documented Python library (Requirements 2.1.1.a, 2.1.2.a, 2.1.2.b).

Skills Utilized:

- Proficiency in Python programming.
- Experience applying software design principles, modular code development, and adhering to PEP8 style guidelines.
- Effective use of version control (Git) and project management tools (GitLab).
- Ability to implement comprehensive unit tests, ensuring at least 60% code coverage.
- Experience packaging, distributing, and deploying Python libraries.

Machine Learning and Data Analysis
Related Requirements: Activation extraction, implementation of clustering algorithms, model interfaces, automated labeling (Requirements 2.1.1.a, 2.1.1.b, 2.1.1.g, 2.1.1.h).

Skills Utilized:

- Deep understanding of neural network architectures and deep learning frameworks (e.g., PyTorch, TensorFlow).
- Proficiency with clustering techniques suitable for high-dimensional activation data.
- Knowledge and application of model interpretability methods.
- Capability to analyze and process large-scale datasets efficiently.

Code Parsing and Analysis
Related Requirements: Parsing source code files, including extending support for languages not natively handled by Tree-sitter (Requirements 2.1.1.e, 2.1.1.f).

Skills Utilized:

- Expertise in using Tree-sitter for code parsing.
- Strong understanding of abstract syntax trees (ASTs) and compiler theory.
- Ability to preprocess and analyze source code across various programming languages.
- Problem-solving capabilities to extend parsing functionalities for unsupported languages (such as OpenMP).

High-Performance Computing (HPC) and Optimization
Related Requirements: Efficiently processing large datasets, ensuring scalability, HPC integration (Requirements 2.1.3.a, 2.1.1.a).

Skills Utilized:

- Familiarity with HPC environments and job scheduling tools like SLURM, specifically on Iowa State's Nova HPC clusters.
- Experience in parallel computing and distributed systems.
- Proven ability to optimize code for enhanced performance and scalability.
- Handling computational challenges associated with large-scale data processing.

Ethical AI Practices and Professional Responsibility
Related Requirements: Addressing biases in AI systems, compliance with ethical guidelines and engineering standards (Requirements 2.2.3.b, 2.1.1.c).

Skills Utilized:

- Strong understanding of AI engineering fundamentals and responsible AI principles.
- Familiarity with standards such as ISO/IEC TR 24027:2021 and IEEE ethical codes.
- Demonstrated commitment to ethical conduct, professional responsibility, and public welfare.
- Skill in integrating ethical considerations into technical project deliverables.

A2.3 Skill Sets covered by the Team

Skill Area	Team Members Covering This Skill Area
Software Engineering and Development	All members
Machine Learning and Data Analysis	All members
Code Parsing and Analysis	All members
HPC and Optimization	All members
Ethical AI Practices	All members

A2.4 Project Management Style Adopted by the team

Our team adopted and successfully implemented Agile methodologies throughout the project's lifecycle, enabling flexibility, adaptability, and responsive management of evolving project requirements.

A2.5 Initial Project Management Roles

Team members rotated the Scrum Master/Team Lead role weekly, ensuring that all members contributed equally to leadership and project management responsibilities. Regardless of the current Scrum Master, each member actively participated in engineering tasks and decision-making.

A2.6 Team Contract

Team Members:

- Manjul Balayar
- Rayne Wilde
- Sam Frost
- Akhilesh Nevatia
- Ethan Rogers

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings: **Wednesday, 12:30 PM, Parks Library / WebEx**
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face): **Discord**
3. Decision-making policy (e.g., consensus, majority vote): **Majority vote**
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived): **Team members are responsible for tracking their own hours throughout the semester, but a set weekly rotation will be used to divide tasks like summarizing group meetings, and non individual portions of the weekly report.**

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings: **Attend all weekly meetings, unless mentioned to the team otherwise**
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines: **Complete assigned issues/tasks as well as rotating responsibilities (weekly meeting summary, etc)**
3. Expected level of communication with other team members: **Be respectful, Provide constructive feedback, Take meeting minutes, Keep documentation on git and discord, Respond in a reasonable amount of time.**
4. Expected level of commitment to team decisions and tasks: **Take full ownership of responsibilities assigned, Communicate with team members regarding progress made, Take active part in team meetings and discussions.**

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.): **As issues are created, the person who is assigned to an issue is responsible for the issue and takes "ownership" of the issue.**
2. Strategies for supporting and guiding the work of all team members:
    a. **Support can come in the form of**
        i. **Taking a sub-task**

            1. **task from an issue can be delegated to other team members but the team member responsible for the issue is still responsible for the quality/checking the task**
        b. **Collaboration**
            i. **pair-programming**
            ii. **sounding boards**
        c. **Reviewer/Tester**
3. Strategies for recognizing the contributions of all team members:
    a. **Look at commits**
    b. **Author tags**
    c. **wiki's**

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

- Manjul Balayar:
    - Majoring in Software Engineering with a minor in Data Science. Core experience and interests lie in Data Science, ML, and DL. Have experience in data processing, modeling, full stack development, etc.
- Rayne Wilde:
    - ML Experience:
        - Academic: 4 years
        - Industry: 3 years
    - Compiler Engineering:
        - Industry: 2 years
    - Data Science / Statistics:
        - Industry: 7 years
    - Quantum Computing:
        - Academic: 1 year
        - Industry: 2 years
- Sam Frost:
    - Academic Experience
        - Full stack web development, embedded systems, various programming languages
    - Industry Experience
        - Cloud Software Development (AWS)
        - Linux
        - Python, Java
        - IT hardware/software support
- Akhilesh Nevatia:
    - Love building and rapid prototyping, prefer backend side of things, enjoy decoding complicated algorithms and have worked in Applied-AI, ML and Stego Research, Avionics and Venture Capital Research

- Ethan Rogers:
    - Experience in the hardware/software codesign of machine learning algorithms for efficient yet accurate results
    - Hardware background from EE curriculum
    - Strong research interests in the deep learning and unsupervised learning space

2. Strategies for encouraging and supporting contributions and ideas from all team members: **Our strategy for encouraging ideas or contributions from each team member will come in the form of promoting learning and skills development by creating a productive and collaborative environment. By ensuring all team members have their ideas heard and their questions answered, more creative and productive discussions will take place, leading to an improved project solution.**

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
    a. **To identify collaboration/inclusion issues, we will use each member's productivity and communication as a measurement tool. A team member who is lacking in communication and results should be asked if they have any roadblocks in the way of their success.**
    b. **To resolve issues, the team will ask whoever it is that might be facing a challenge what the team can do to aid them. It should be made a priority by each member of the team to resolve such issues.**

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
    a. **Make progress that aligns with the goals and objectives of our advisor and client**
    b. **Build and develop professional skills**
    c. **Implement project management skills learned from school and industry.**

2. Strategies for planning and assigning individual and team work:
    a. **Tasks will be distributed based on the needs of the group, individual strengths and weaknesses, and the preference of team members.**
3. Strategies for keeping on task:
    a. **Use the Epic/Milestone/Issue/Task features to make incremental progress towards larger goals.**

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
    a. **Verbal/Discord warning. Advisor will be made aware of infraction.**
2. What will your team do if the infractions continue?
    a. **Inform professors of the repeated infractions.**

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Akhilesh Nevatia DATE 9/19/24
2) Ethan Rogers DATE 9/19/24
3) Sam Frost DATE 9/19/24
5) Rayne Wilde DATE 9/19/24
6) Manjul Balayar DATE 9/19/24