University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

A Comparison of Embedding Models in Semantic Similarity Tasks for Software Engineering

Vinícius Henrique Ferraz Lima

Final Essay

mac 499 — Capstone Project

Supervisor: Arthur Pilone Maia da Silva
Co-supervisor: Prof. Paulo Roberto Miranda Meirelles

São Paulo
2025

Acknowledgements

Audentes Fortuna Iuvat!
— Hans Capon

Initially, I would like to express my gratitude to my advisor, Prof. Paulo Roberto Miranda Meirelles and mainly to Arthur Pilone Maia da Silva, for his guidance and support throughout this research. His expertise and insights were very helpful in shaping the direction of this work.

Furthermore, I would like to thank my family and friends for their support and understanding throughout this journey. If not for them, I would never have been able to complete this work neither start this graduation!

Resumo

Vinícius Henrique Ferraz Lima. **Uma comparação de modelos de embedding em tarefas de similaridade semântica para Engenharia de Software**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2025.

Modelos de embedding têm sido amplamente utilizados em tarefas de Processamento de Linguagem Natural (PLN) para representar textos em espaços vetoriais que capturam suas relações semânticas. Apesar do surgimento de diferentes arquiteturas e técnicas de treinamento, presume-se que, em determinadas tarefas, esses modelos apresentem desempenhos semelhantes. Este trabalho investiga essa hipótese por meio de uma comparação sistemática entre diferentes modelos de embedding aplicados a tarefas de similaridade semântica textual. A partir dos experimentos realizados, buscamos compreender se as diferenças entre modelos usados amplamente no mercado são relevantes para aplicações práticas. Os resultados contribuem para uma melhor compreensão sobre o impacto real da escolha de modelos de embedding em aplicações de PLN.

Palavras-chave: Embeddings. Similaridade Semântica. Processamento de Linguagem Natural. Representação Vetorial de Texto. Avaliação de Modelos..

Abstract

Vinícius Henrique Ferraz Lima. **A Comparison of Embedding Models in Semantic Similarity Tasks for Software Engineering**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2025.

Embedding models have been widely used in Natural Language Processing (NLP) tasks to represent texts in vector spaces that capture their semantic relationships. Despite the emergence of different architectures and training techniques, it is presumed that, in certain tasks, these models exhibit similar performance. This work investigates this hypothesis through a systematic comparison of different embedding models applied to semantic textual similarity tasks. From the experiments conducted, we seek to understand whether differences between models used widely in the market are relevant for practical applications. The results contribute to a better understanding of the real impact of choosing embedding models in NLP applications.

Keywords: Embeddings. Semantic Similarity. Natural Language Processing. Vectorial Representation of Text. Model Evaluation..

Lista de abreviaturas

SE	Software Engineering
STS	Semantic Textual Similarity
LLM	Large Language Model

List of Figures

Contents

Introduction

In recent years, advances in Natural Language Processing (NLP) and Artificial Intelligence (AI) have transformed the way researchers and industry professionals analyze, retrieve, and relate textual information. Central to these advances are embedding models, which are mathematical representations that encode the semantic meaning of text into numerical vectors. These representations serve as the backbone of modern retrieval, classification, and generation systems, enabling more semantic driven interpretations of human language than traditional keyword-based or statistical methods.

As embedding models have matured, a growing body of research in software engineering has adopted them to address complex tasks such as duplicate bug report detection, vulnerability discovery, source code analysis, and requirements traceability. These studies consistently demonstrate that embeddings capture semantic and contextual relationships that would be difficult or impossible to model through handcrafted rules or surface-level text matching.

From this tecnology, we raise important questions for both researchers and practitioners: *to what extent does the choice of embedding model actually influence the final outcomes of a given task?* While new embedding models are released at an increasingly rapid pace, often accompanied by claims of marginal improvements, there remains a lack of studies evaluating whether these newer and different models differ in performance gains in real-world applications.

A notable contribution to this discussion is DeeperMatcher, proposed by Pilone *et al.* (2025), which leverages large language models and text embeddings to connect user feedback with development issues based on semantic similarity. The study highlights the effectiveness of embedding based similarity tasks to find relationships between reports and issues that would otherwise remain undetected by developers. However, its evaluation relies on a specific embedding model, leaving open the question of whether similar or better results could be achieved with alternative models.

Motivated by this gap, the present work aims to replicate and extend aspects of DeeperMatcher's methodology to investigate how different embedding models impact the performance of semantic similarity tasks in a software engineering context. Starting from the premise that embedding models often perform similarly, we systematically compare multiple embeddings models available in the market. Our goal is to determine whether the choice of embedding model significantly influences performance in practical applications.

Chapter 1

Related Concepts

Natural Language Processing

According to Jurafsky and Martin (2023), Natural Language Processing (NLP) is a field of Artificial Intelligence (AI) dedicated to enabling computers to understand, interpret, and generate human language. Early research in NLP dates back to the 1950s, with initiatives such as the Georgetown–IBM experiment, which successfully translated over sixty sentences from Russian to English (Hutchins, 2004). Since then, the field has evolved from rule-based systems to statistical and, more recently, deep learning approaches.

A major milestone in this evolution was the introduction of embeddings, which are numerical representations of words, sentences, or documents that capture their semantic meaning in a continuous vector space. These representations have become fundamental to modern NLP models, as they're used on a daily basis to power a wide range of applications such as machine translation, sentiment analysis, and information retrieval.

Recent advances in Natural Language Processing (NLP) and Machine Learning have expanded the use of semantic representations beyond traditional text classification or sentiment analysis. In software engineering, these techniques have increasingly been applied to support program comprehension, bug detection, and security auditing.

The use of embeddings for improving software quality has been explored in multiple research directions. For example, Li *et al.* (2024) proposed a hybrid retrieval method for detecting duplicate software bug reports by combining traditional keyword-based retrieval with semantic embeddings to represent the textual descriptions of reports. This combination allowed the system to identify duplicates even when reports used different wording or phrasing to describe the same underlying problem. By leveraging embedding based similarity, the approach outperformed earlier methods that relied solely on text matching, demonstrating the effectiveness of semantic representations in capturing contextual meaning within bug reports.

Another notable example is SICode, which introduced an embedding-based subgraph isomorphism method for bug detection by Zhang *et al.* (2024). Instead of relying on computationally expensive exact matching of source code graphs, SICode represents each

code structure as a graph embedding, allowing for efficient comparison of code segments based on their semantic and structural similarity. This method enables the detection of previously unseen or variant bugs that share similar logical patterns, significantly,

Embeddings have also been applied to enhance software security analysis. The work Detecting Hard-Coded Credentials in Software Repositories via LLMs leveraged contextual embeddings generated by large language models to identify sensitive information such as passwords and API keys embedded in source code by Rahman *et al.* (2024).

Together, these studies underscore the increasing role of embedding based representations in software engineering research. Whether for identifying duplicate bug reports, detecting structural vulnerabilities, or locating security flaws, embeddings provide a unified framework for capturing semantic and contextual relationships across different data modalities. This growing body of work supports the idea that embedding embedding models achieve comparable performance across diverse software analysis tasks, motivating a deeper investigation into their similarities and differences.

Building upon this context, Pilone *et al.* (2025) proposed DeeperMatcher, an approach designed to automatically connect user feedback with development issues by leveraging large language models and text embeddings to measure semantic similarity between textual artifacts. Using data from the replication package of Pilone *et al.* (2025) and other large-scale open-source projects, such as Brave, the study demonstrated how embedding-based similarity can reveal connections between software issues and user reports that would otherwise remain hidden. This work set a relevant precedent for exploring embedding behavior in software engineering scenarios.

Artificial Intelligence and Embedding Models

Artificial intelligence (AI) models were originally developed to mimic aspects of human cognition, and they differ significantly from traditional algorithms. Instead of following a fixed set of instructions, an AI model's behavior is shaped by the data used to train it. Once trained, its outputs result from complex parameter-dependent mathematical computations, sometimes with a stochastic component.

Since many NLP tasks are hard to solve with traditional algorithms, AI models handle text by breaking it into tokens and turning each one into a numerical embedding that represents its meaning. There's no practical way to design these representations by hand, so larger and more data-hungry models usually perform the best. These models often have hundreds of millions or even billions of parameters, and because they require so many resources to train, they're generally referred to as large language models (LLMs).

A fundamental architectural innovation that enabled the development of modern LLMs and embedding models is the Transformer architecture, introduced by Vaswani *et al.* (2017) in the seminal paper "Attention is All You Need". The Transformer architecture relies entirely on attention mechanisms, eliminating the need for recurrent or convolutional layers that were previously standard in sequence modeling. This design allows models to process entire sequences in parallel rather than sequentially, dramatically improving training efficiency. The attention mechanism enables the model to weigh the importance of

different parts of the input when processing each token, allowing it to capture long-range dependencies and contextual relationships more effectively than previous architectures.

Given the popularity of embedding applications, as seen in the previous section, some models are trained specifically to generate semantic representations (embeddings) that facilitate comparisons between texts. These embedding models are optimized to produce dense vector representations where semantically similar texts are mapped to nearby points in the vector space, enabling efficient similarity calculations and retrieval tasks. Among the popular embedding models available, we find both open-source options such as Jina embeddings, and proprietary models accessed through API services, including OpenAI's embedding models (both large and small variants), Google's Gemini Text Embedding, and Cohere's Embed models. Each of these models employs different architectures and training strategies, while sharing a common goal of producing meaningful vector representations of text.

Clusterization and Visualization

As seen in the previous section, embeddings are high-dimensional vectors where semantically similar texts are mapped to nearby points in the vector space. Clusterization is a technique that helps visualize these relationships by grouping similar data points together. In the context of this project, each issue and review is represented as an embedding vector, and clusterization helps visualize and compare how different embedding models structure this space by identifying which texts are positioned close to each other.

However, visualizing high-dimensional data directly is not possible. To address this, we apply a dimensionality-reduction technique that projects the vectors into a two-dimensional plane while trying to preserve their relative distances and relationships. This allows us to visually inspect how issues and reviews group together and how different embedding models separate or mix these elements.

In this work, the chosen technique for visualization was t-SNE (t-Distributed Stochastic Neighbor Embedding) (Tripathy *et al.*, 2021). t-SNE is widely used for mapping high-dimensional data into 2D or 3D while preserving the local structure of the data. In other words, points that are close in the original embedding space tend to appear close in the 2D visualization.

t-SNE works by:

- Modeling similarities between points using probability distributions;
- Preserving small-scale, neighborhood relationships rather than global distances;
- Allowing non-linear transformation, which makes it effective for detecting clusters in complex data.

Chapter 2

Data and Methods

This chapter explains how the experiment was carried out and what steps were followed to compare different embedding models. Our main goal is to understand whether replacing the embedding model in Pilone *et al.* (2025) affects the final similarity results.

This project follows a replication and extension approach. The original DeeperMatcher code and logic were kept as close as possible to the previous implementation, with the only intentional modification being the embedding models used to generate vector representations. By isolating this variable, it becomes possible to observe how much different embedding models actually influence the outcome.

2.1 Dataset

The dataset used in this work originates from the study of Pilone *et al.* (2025), which investigated the use of text embeddings to match issues and review comments from software projects. In that study, the author conducted a evaluation using issues and reviews from 20 different open-source projects, over 87000 issues and 1270000 reviews. This material is suitable for embedding analysis because the issues and reviews contain detailed natural language descriptions that can be transformed into meaningful vector representations.

The dataset includes:

- issue descriptions from these projects,
- user reviews collected from Google Play Store,
- relevance feedback from programmers, who rated the top matches generated by DeeperMatcher.

In addition to collecting textual data, Pilone *et al.* (2025) created a survey evaluation to generate human labels for a subset of issue-review pairs that were matches according to DeepMatcher. Selected pairs were presented to programmers, who were asked first to decide whether the review was relevant to the issue (True or False), and then to rate the degree of relevance on a scale from 1 to 5. This survey was conducted on 19 projects,

totalizing 4000 feedbacks over 400 issues, and 2000 issue-review pairs. There are more than 4000 feedbacks because part of the issues were reviewed three times. Each suggestion was reviewed twice, and some (where there was disagreement) were reviewed three times. The final score was the average of the ratings. These evaluations produced a valuable human-validated subset of the data, indicating which matches were genuinely meaningful according to practitioners

Since the dataset had already been preprocessed by the original researchers, no additional cleaning or normalization was applied. Using the data exactly as provided ensures that results remain directly comparable to the original work. This dataset is appropriate for the present work because its organization ensures that issues and reviews are kept within their respective projects. The large number of review texts provides sufficient content for computing embeddings and examining similarity patterns across different models. Furthermore, the human-validated relevance scores from the forms experiment allow a direct comparison between the rankings produced by alternative embedding models and those considered relevant by programmers. Each suggestion was reviewed twice, and some (where there was disagreement) were reviewed three times. The final score was the average of the ratings.

2.2 Project Selection

For this study, we restricted ourselves to a limited subset of the same projects used during the field study conducted in Pilone *et al.* (2025), where programmers evaluated the quality of the generated matches. Using these projects maintains consistency with the original evaluation setup and enables a direct comparison against human relevance scores. The following projects were selected:

- **WordPress** (`wordpress_android`) — a content management system for websites and blogs, available on Google Play Store.[1]
- **K9 Mail** (`fsck_k9`) — an email client application, available on Google Play Store.[2]
- **PPSSPP** (`ppsspp_ppsspp`) — a PlayStation Portable emulator, available on Google Play Store.[3]
- **Mindustry** (`anuke_mindustry`) — a tower defense and factory management game, available on Google Play Store.[4]

These projecrs have between 12177 (K9 Mail) and 104652 (WordPress) reviews, and between 331 (PPSSPP) and 8758 (WordPress) issues.

[1] https://play.google.com/store/apps/details?id=org.wordpress.android

[2] https://play.google.com/store/apps/details?id=com.fsck.k9

[3] https://play.google.com/store/apps/details?id=org.ppsspp.ppsspp

[4] https://play.google.com/store/apps/details?id=io.anuke.mindustry

2.3 Embedding Models

This study evaluates the performance of several widely used embedding models in semantic similarity tasks involving issues and review comments. The work of Pilone *et al.* (2025) used the Jina 3 (jina-embeddings-v3), so this model was also included to maintain compatibility with the original experiment. The Jina model is an open-source, Transformer-based embedding model executed locally.

To compare Jina with models from different providers, additional embedding models were included. Unlike Jina, these models are proprietary and accessed through API services, but they are commonly used in industry and research due to their strong performance across semantic tasks. The models evaluated in this study were:

- OpenAI Large — a higher-capacity embedding model designed for tasks requiring more detailed semantic representations.
- OpenAI Small — a lighter and faster variant, suitable for applications where efficiency is more important than maximum representation depth.
- Gemini Text Embedding 004 — Google's current-generation embedding model, optimized for semantic similarity and retrieval tasks.
- Cohere Embed v4.0 — Cohere's latest embedding model, designed for high-quality text encoding across multilingual and domain-diverse inputs.

These models were selected because they represent different families of widely used embedding models. Their comparison allows an analysis of whether different models produce similar similarity rankings when applied to the same project-level data.

2.4 Execution and Code Adaptation

To extend the original system, the existing codebase was modified to integrate multiple embedding models. Because some models run locally while others depend on remote API calls, the pipeline was rewritten to use Python's asynchronous programming. This allowed embeddings to be generated in parallel, reducing total execution time and preventing API latency from slowing down the process.

No fine-tuning or parameter adjustment was applied. All embeddings were generated using the default settings of each model.

The full original DeeperMatcher pipeline was used as the foundation of this project. Only improvements related to organization and embedding integration were added, such as:

- modularizing the code,
- adding asynchronous execution,
- creating adapters for each embedding model,
- extending the evaluation workflow to handle multiple models.

All core logic remained unchanged. This ensures that differences in results can be attributed to the embeddings themselves and not to modifications in the experimental method.

2.5 Similarity Computation

The similarity score between an issue and a review was calculated using normalized cosine similarity, following exactly the same definition used in DeeperMatcher.

After all issues and reviews were embedded with the chosen model, for each issue:

- similarity scores were computed against all reviews in the project,
- the results were sorted in descending order,
- the top 10 most similar reviews were selected.

The top-10 ranking was preserved from the original research to guarantee consistency with the research of Pilone *et al.* (2025).

These ranked results serve as the basis for:

- comparing different embedding models, and
- checking how similarity scores align with the human-assigned relevance levels (1 to 5).

2.6 Comparison Against Human Feedback

Although the original relevance labels come from the previous study, they remain essential to evaluating embedding quality. In this work, similarity values from each embedding model were compared across relevance levels to investigate whether:

- higher relevance reviews tend to have higher cosine similarity,
- different models show similar or different behavioral patterns,
- any embedding model correlates more strongly with human judgment.

This provides a practical measure of how "semantically accurate" each model is in real software engineering scenarios.

2.7 Clusterization and Visualization

To explore how each embedding model organizes semantic space, vector representations were projected into two dimensions using t-SNE (t-Distributed Stochastic Neighbor Embedding), and the clusterization was performed on reviews and issues.

The purpose of these visualizations is to inspect:

- whether issues and reviews cluster distinctly or overlap,

- how the cluster layout differs between embedding models,
- whether cluster structures match the similarity and relevance patterns observed.

This qualitative analysis helps complement the numerical results by showing how each model structures meaning in high-dimensional space.

Chapter 3

Similarity Score Distributions

This chapter presents and discusses the distributions of similarity scores across the entire dataset, examining how different embedding models assign similarity values to issue-review pairs. Unlike the relevance-level analysis which focused on human-labeled pairs, this analysis considers all computed similarities, revealing how models behave across the full spectrum of potential matches. We present histogram and violin plot visualizations for each project, followed by a discussion of cross-project patterns and their implications for model selection in practice.

To gain deeper insight into how different embedding models assign similarity scores across the entire dataset, we analyzed the distributions of similarity scores for each project. These distributions reveal how models behave across the full range of issue-review pairs, not just those with human-assigned relevance labels.

3.1 Overall Similarity Distributions

Figure 3.1 shows histogram distributions of similarity scores across all four projects. The distributions reveal important differences in how models assign similarity values:

- **Project-specific patterns:** Each project shows distinct distribution characteristics. For example, `wordpress_android` and `fsck_k9` show narrower, more concentrated distributions with peaks around 7000–12000 matches suggested at certain similarity score ranges, while `anuke_mindustry` and `ppsspp_ppsspp` show broader distributions with lower peak frequencies, indicating fewer matches suggested at any given similarity score range.
- **Model-specific shifts:** Across projects, certain models consistently shift distributions. Cohere and Gemini tend to produce distributions shifted toward lower (more negative) similarity scores, while Jina, OpenAI Large, and OpenAI Small show distributions shifted toward higher (more positive) scores.
- **Distribution shapes:** Most distributions are approximately bell-shaped and centered around zero, suggesting that similarity scores follow a normal-like distribution.

However, the width and peak locations vary significantly between models and projects.

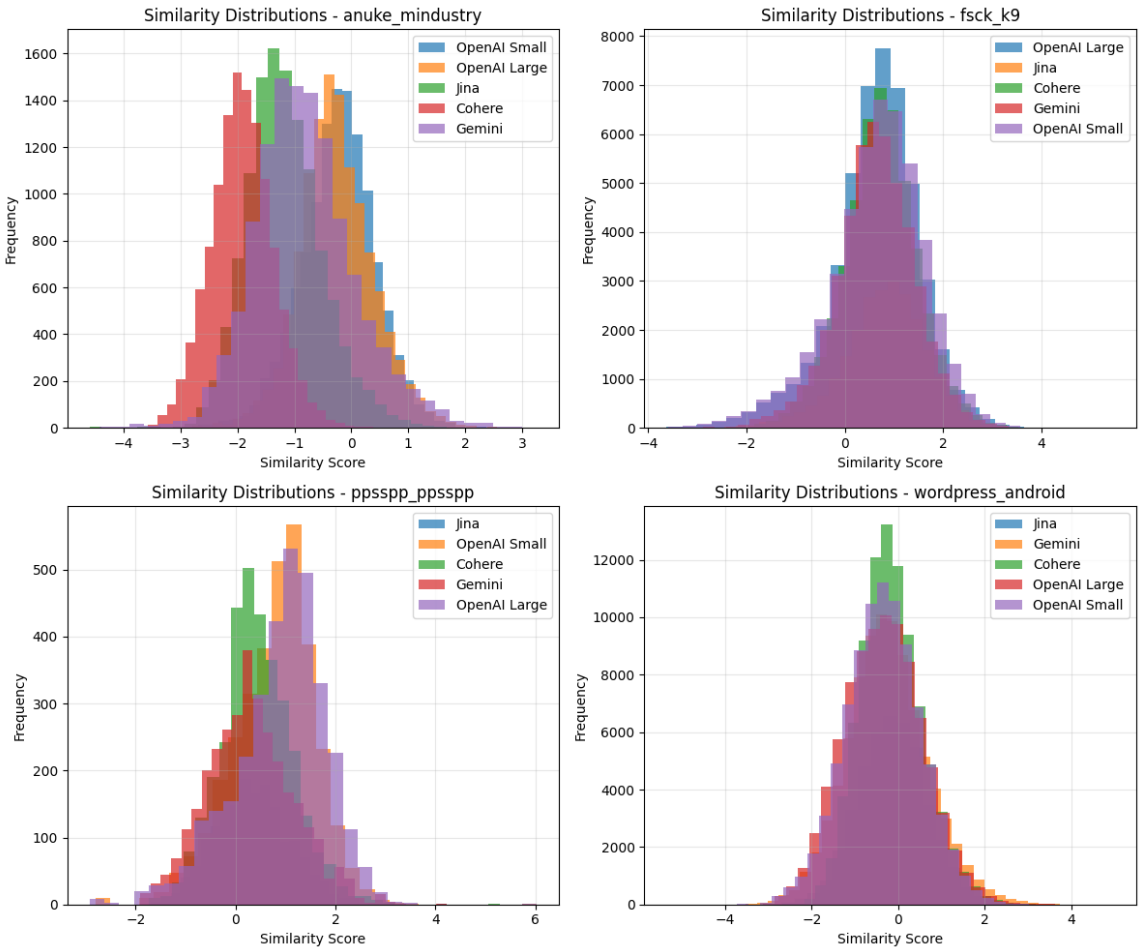

Figure 3.1: *Histogram distributions of similarity scores across four projects for all five embedding models. Each subplot shows the frequency distribution of similarity scores, revealing project-specific and model-specific patterns.*

3.2 Reviews per Similarity by Project

To examine how similarity scores are distributed specifically for reviews matched to issues, violin plots were generated for each project. These plots show the probability density of similarity percentages, providing insight into model behavior at the project level. All violin plots follow a consistent order of models (from left to right: Jina, Gemini, Cohere, OpenAI Large, OpenAI Small) to facilitate comparison across different projects.

3.2.1 Wordpress Android

Figure 3.2 shows the similarity distribution for the `Wordpress Android` project. Jina shows a median similarity of approximately 70%, followed by Gemini with the highest median (approximately 75%). Cohere shows a significantly lower median (approximately

45–48%), while OpenAI Large and OpenAI Small show similar medians (approximately 45–48%) with very similar distributions. The violin shapes reveal that Jina and Gemini have tighter, more concentrated distributions at higher similarity values, while Cohere and the OpenAI models show broader distributions centered at lower similarity percentages.

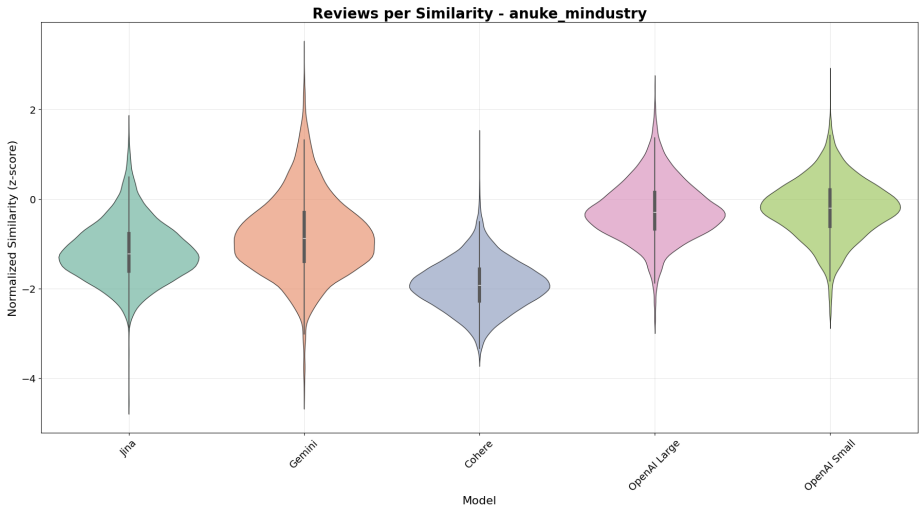

Figure 3.2: *Violin plot showing similarity score distributions for Wordpress Android project. Gemini and Jina show higher medians and tighter distributions compared to other models.*

3.2.2 Mindustry

For the `Mindustry` project (Figure 3.3), Jina shows a median similarity of approximately 62–63%, followed by Gemini with the highest median (approximately 70–71%). Cohere consistently shows the lowest median (approximately 32–33%) with a narrow distribution, while OpenAI Large and OpenAI Small show moderate medians (approximately 47–51%). This pattern suggests that Cohere is particularly conservative in assigning similarity scores for this project.

3.2.3 K9 Mail

The `K9 Mail` project (Figure 3.4) shows Jina with a median around 70–75%, followed by Gemini with the highest median (approximately 75–78%) and the tightest distribution, indicating very consistent high similarity scores. Cohere shows a median of approximately 50–60% with wider, more variable distributions, while OpenAI Large shows a similar median (approximately 50–60%) and OpenAI Small shows moderate performance (approximately 60–65%). The violin shapes reveal that Jina and Gemini have particularly narrow and concentrated distributions, suggesting high consistency in their similarity assignments.

3.2.4 PPSSPP

For the `PPSSPP` project (Figure 3.5), Jina shows a median of approximately 70% with a relatively tight distribution, followed by Gemini with the highest median (approximately 80%) and the tightest distribution, concentrated between 70% and 90%. Cohere shows a lower median (approximately 50%) with broader distributions, while OpenAI Large shows

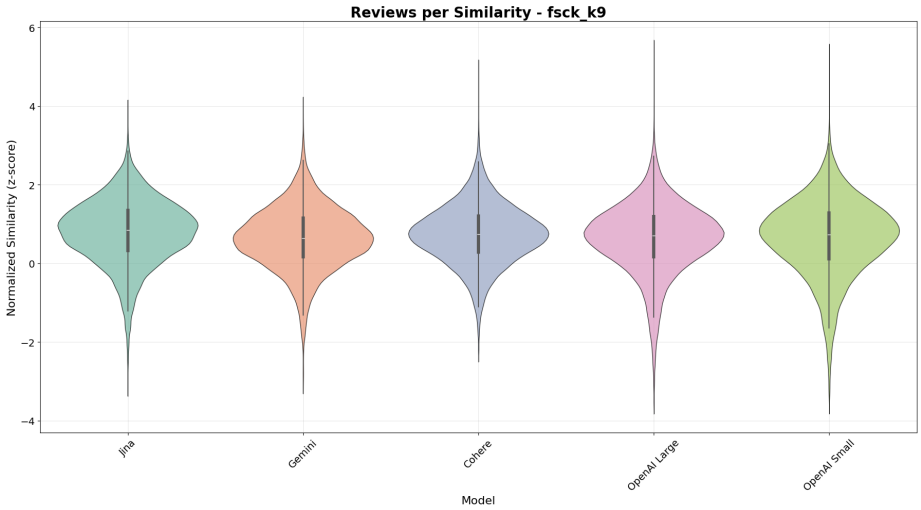

Figure 3.3: *Violin plot for Mindustry project. Gemini and Jina outperform other models, while Cohere shows consistently lower similarity scores.*

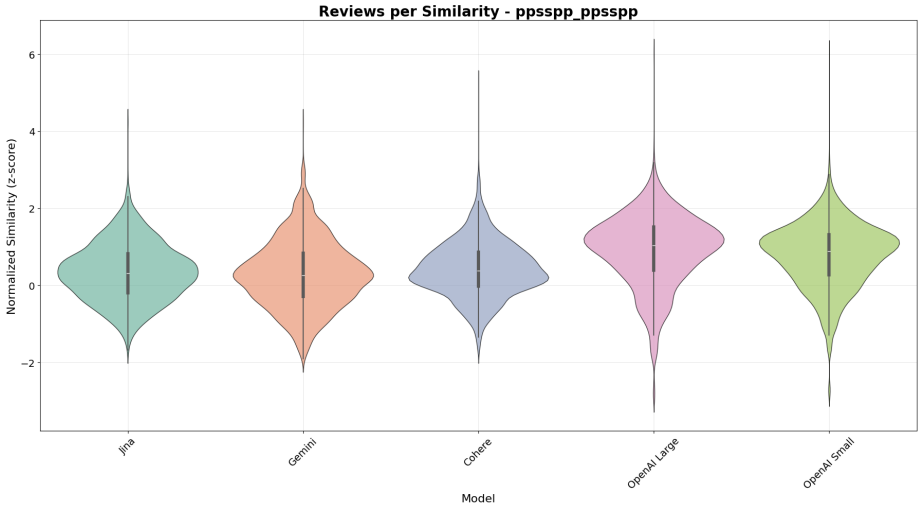

Figure 3.4: *Violin plot for fsck_k9 project. Gemini shows the highest median and tightest distribution, indicating consistent high performance.*

a similar median (approximately 50%) with the widest range, occasionally producing very low similarity scores. OpenAI Small shows moderate performance (approximately 57%).

3.2.5 Cross-Project Patterns

The violin plot analysis reveals consistent patterns across projects:

- **Gemini's consistent behavior:** Across all four projects, Gemini consistently shows the highest median similarity scores and often the tightest distributions. This indicates that Gemini assigns higher similarity values more consistently than other models, though whether this is desirable depends on the application context: higher similarity scores may capture more relevant matches but could also increase false positives.

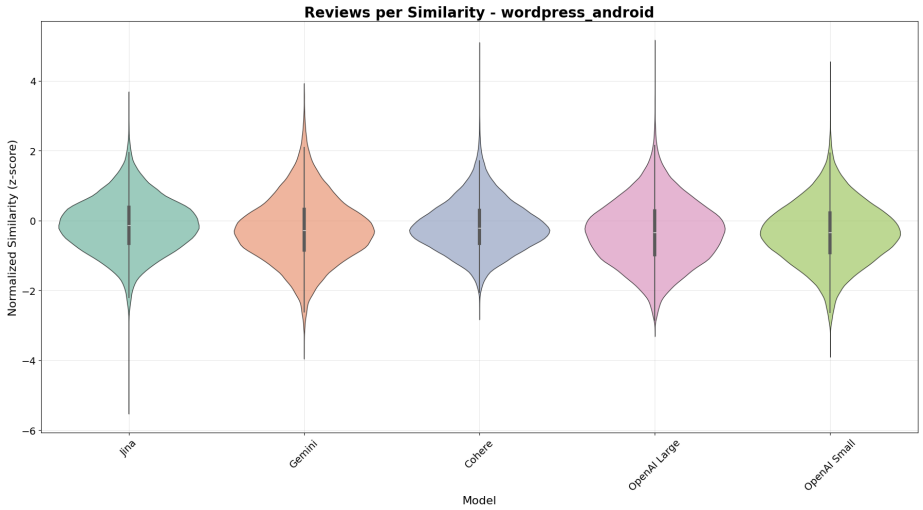

Figure 3.5: *Violin plot for PPSSPP project. Gemini shows exceptional performance with highest median and tightest distribution, while Jina follows closely.*

- **Jina's consistent behavior:** Jina consistently shows the second-highest median similarity scores across projects, with medians higher than OpenAI models and Cohere, and relatively tight distributions. Like Gemini, Jina's tendency to assign higher similarity scores suggests a more generous approach to semantic matching.
- **OpenAI model similarity:** OpenAI Large and OpenAI Small show very similar performance patterns across projects, with medians typically in the 45–60% range and broader distributions compared to Gemini and Jina. This similarity suggests that model size may have less impact on similarity scoring behavior than other factors.
- **Cohere's conservative scoring:** Cohere consistently shows lower similarity scores across projects, with particularly low medians in `anuke_mindustry` and `ppsspp_ppsspp`. This conservative approach may result in fewer false positives but potentially more false negatives, as the model requires stronger semantic signals before assigning high similarity values.
- **Project-specific variations:** While the relative ordering of models by median similarity remains relatively consistent across projects, the absolute similarity values and distribution widths vary significantly between projects, suggesting that project characteristics influence how models assign similarity scores.

These distribution analyses complement the relevance-level analysis by showing how models behave across the entire similarity spectrum, not just for pairs with human-assigned labels. The consistent patterns across projects suggest that model characteristics (architecture, training data, optimization objectives) have a stronger influence on similarity scoring than project-specific factors.

Chapter 4

Similarity and Relevance

This chapter presents and discusses the results obtained from analyzing how similarity scores computed by five embedding models correlate with human-assigned relevance levels. The analysis addresses whether different embedding models successfully capture semantic relationships aligned with human judgment, examining how similarity scores vary across the six relevance levels (1-5) assigned by practitioners. We present the quantitative findings organized by relevance level, followed by a comparative discussion of patterns observed across models.

4.1 Similarity Scores by Relevance Level

To evaluate how well each embedding model captures semantic relationships aligned with human judgment, similarity scores were analyzed across the six relevance levels (1-5) assigned by practitioners. The expectation is that higher relevance levels should correspond to higher cosine similarity scores, indicating that models successfully distinguish between relevant and irrelevant matches.

4.1.1 No Relevance (Level 0)

Figure 4.1 shows the distribution of normalized similarity scores for issue-review pairs rated as having no relevance (Level 0). Across all five models, the median similarity scores are consistently negative, ranging from approximately -0.3 to -0.5. This pattern indicates that all models successfully identify non-relevant pairs by assigning lower similarity scores, which aligns with the expected behavior.

The interquartile ranges are relatively similar across models, typically spanning from approximately -1.0 to 0.1 or 0.2. This consistency suggests that, despite architectural differences, all models converge on similar similarity distributions for clearly irrelevant matches. The presence of some outliers, particularly in Jina and Gemini, indicates occasional false positives where non-relevant pairs received unexpectedly high similarity scores.

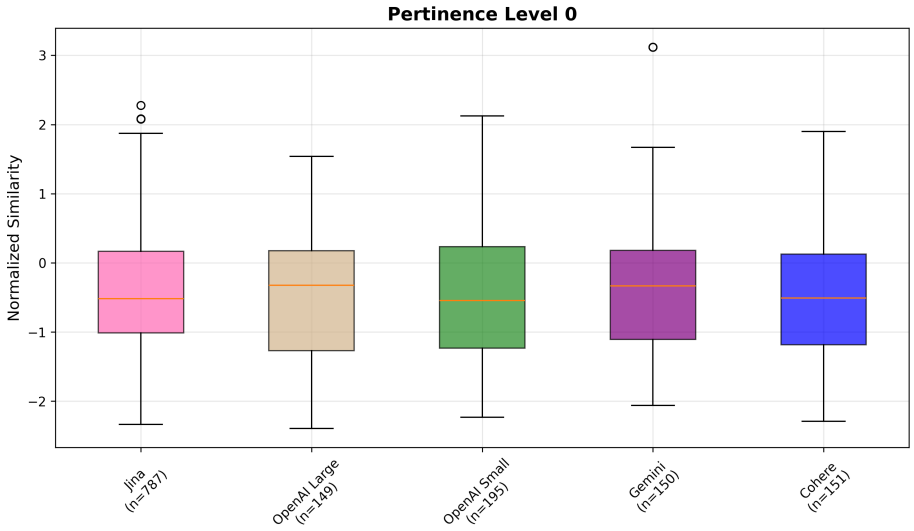

Figure 4.1: *Normalized similarity scores for relevance level 0 (no relevance) across five embedding models. All models show negative medians, indicating successful identification of non-relevant pairs.*

4.1.2 Low Relevance (Levels 1–2)

For Level 1 (Figure 4.2), all models exhibit median similarity scores close to zero or slightly negative. Jina shows the highest median (approximately 0.0), while OpenAI Small has the lowest (approximately -0.3). The distributions show considerable overlap, with interquartile ranges spanning both negative and positive values. This suggests that at low relevance levels, models struggle to consistently distinguish between marginally relevant and irrelevant pairs.

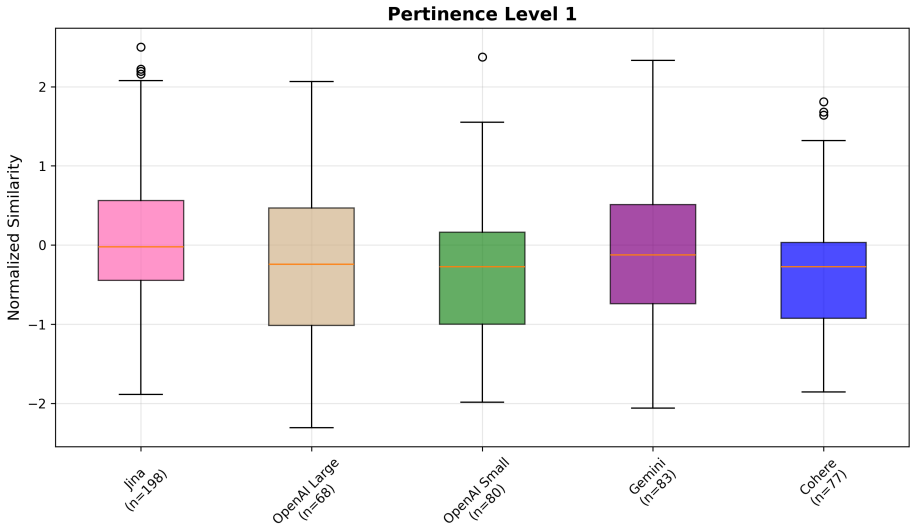

Figure 4.2: *Normalized similarity scores for relevance level 1 (low relevance) across five embedding models. Median scores cluster around zero, indicating difficulty in distinguishing low-relevance pairs.*

At Level 2 (Figure 4.3), a clearer separation begins to emerge. Jina shows the highest median (approximately 0.25), followed by OpenAI Small (approximately 0.15) and OpenAI

Large (approximately 0.1). In contrast, Gemini and Cohere exhibit negative medians (approximately -0.2), indicating that these models assign lower similarity scores even to pairs with moderate relevance. This divergence suggests that different models may have varying thresholds for what constitutes meaningful semantic similarity.

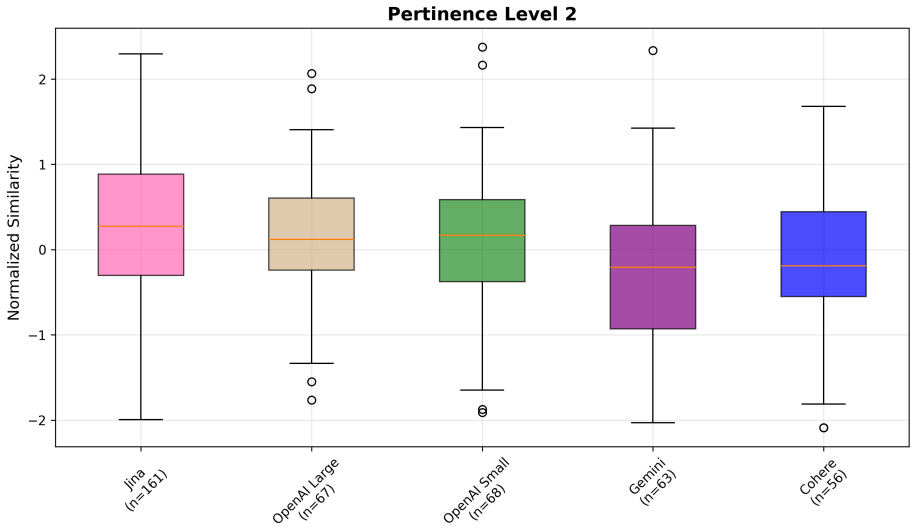

Figure 4.3: *Normalized similarity scores for relevance level 2 (moderate relevance) across five embedding models. Jina, OpenAI Large, and OpenAI Small show positive medians, while Gemini and Cohere remain negative.*

4.1.3 Medium Relevance (Level 3)

Level 3 (Figure 4.4) reveals the most significant differences between models. Jina and OpenAI Large show the highest median similarity scores (approximately 0.35 and 0.25, respectively), with their interquartile ranges largely above zero. OpenAI Small and Cohere exhibit medians closer to zero (approximately 0.15 and 0.0), while Gemini shows a negative median (approximately -0.15).

This pattern suggests that Jina and OpenAI Large are more sensitive to moderate semantic relationships, assigning higher similarity scores to pairs that human evaluators considered moderately relevant. The negative median for Gemini at this level is particularly noteworthy, as it indicates that this model may require stronger semantic signals before assigning positive similarity scores.

4.1.4 High Relevance (Levels 4–5)

At Level 4 (Figure 4.5), all models show positive median similarity scores, ranging from approximately 0.3 to 0.5. Jina and OpenAI Small exhibit the highest medians (approximately 0.5), followed by OpenAI Large (approximately 0.4), Gemini (approximately 0.3), and Cohere (approximately 0.3). The convergence toward positive values indicates that all models successfully identify highly relevant pairs, though with varying degrees of confidence.

For Level 5 (Figure 4.6), representing the highest relevance, all models show remarkably similar median similarity scores clustered around 0.6–0.7. This convergence suggests that

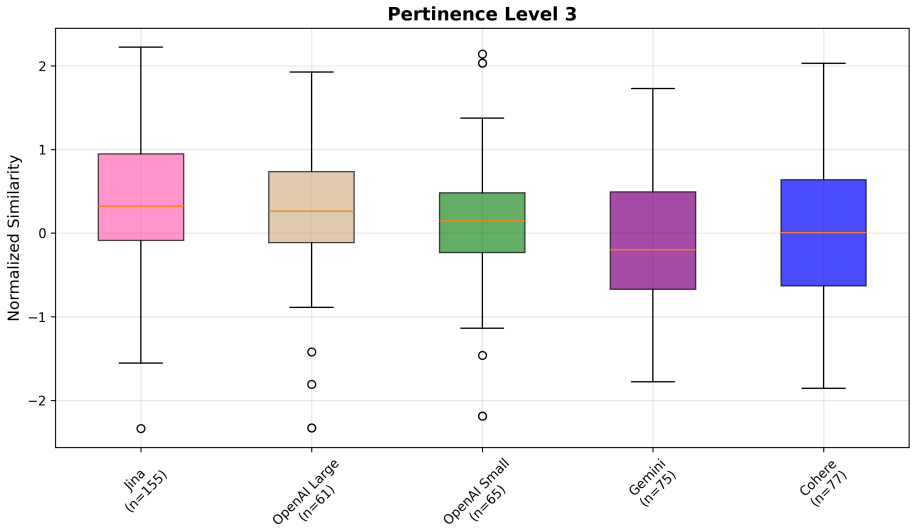

Figure 4.4: *Normalized similarity scores for relevance level 3 (medium relevance) across five embedding models. Jina and OpenAI Large show higher medians, indicating better alignment with human judgment at this relevance level.*

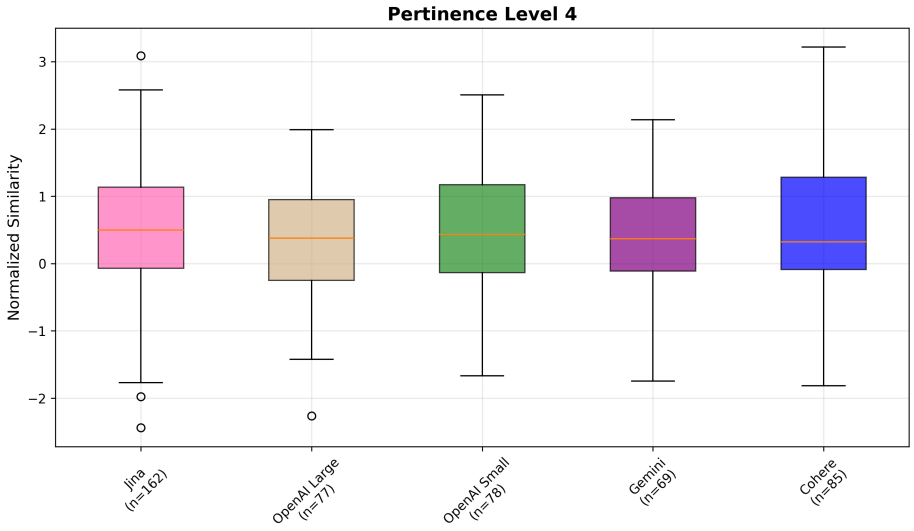

Figure 4.5: *Normalized similarity scores for relevance level 4 (high relevance) across five embedding models. All models show positive medians, indicating successful identification of highly relevant pairs.*

when semantic relationships are strong and unambiguous, all models produce comparable results. The interquartile ranges are also similar across models, typically spanning from approximately 0.0 to 1.2, indicating consistent behavior for clearly relevant pairs.

4.2 Discussions

The analysis across relevance levels reveals several important patterns:

- **Convergence at extremes:** At both ends of the relevance spectrum (Levels 0 and

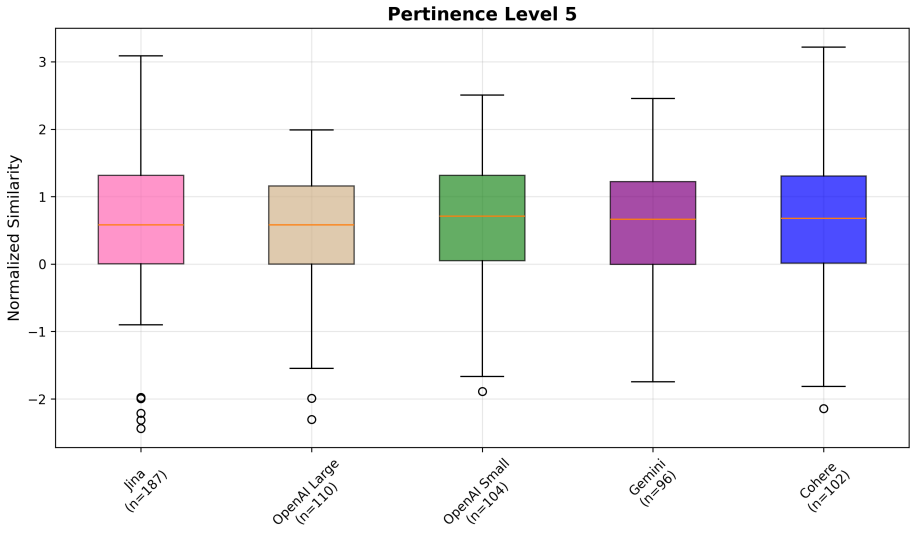

Figure 4.6: *Normalized similarity scores for relevance level 5 (very high relevance) across five embedding models. All models show similar medians (≈0.6), indicating convergence for strongly relevant pairs.*

5), all models show similar behavior. This suggests that when semantic relationships are clearly strong or clearly absent, different embedding models produce comparable results.

- **Divergence at intermediate levels:** The most significant differences between models occur at intermediate relevance levels (Levels 2–3). Jina and OpenAI Large consistently show higher similarity scores at these levels, while Gemini tends to be more conservative, assigning lower scores even to moderately relevant pairs.
- **Consistent ordering:** Across most relevance levels, Jina and OpenAI Large tend to rank highest, followed by OpenAI Small, with Gemini and Cohere showing lower medians. This ordering suggests that model capacity and architecture may influence sensitivity to semantic relationships.
- **Overall correlation with human judgment:** Despite differences in absolute scores, all models show a positive trend: similarity scores generally increase as relevance levels increase, indicating that all models capture some aspect of semantic relevance aligned with human judgment.

4.2.1 The Challenge of Low-Relevance Distinction

A particularly important finding that emerges from the analysis across all relevance levels is that embedding models struggle to consistently distinguish between marginally relevant and irrelevant pairs. This difficulty is most evident at Level 1, where all models exhibit median similarity scores clustered around zero, with distributions spanning both negative and positive values. This pattern is also corroborated by the behavior observed at Levels 2 and 3, where models diverge significantly in their assessments - some models (notably Gemini) still assign negative similarity scores to pairs that human evaluators considered moderately relevant, while others (Jina and OpenAI Large) assign positive scores.

This divergence at intermediate relevance levels highlights a fundamental limitation: different embedding models have varying thresholds for what constitutes meaningful semantic similarity, and these thresholds do not always align with human judgment, particularly for borderline cases. The convergence observed at extreme relevance levels (Levels 0 and 5) contrasts sharply with this divergence, suggesting that while embedding models are reliable for clearly strong or clearly absent semantic relationships, they require careful selection when dealing with the nuanced cases that often arise in real-world software engineering scenarios.

Chapter 5

Visualizing Semantic Spaces

This chapter presents and discusses the results from visualizing how each embedding model structures the semantic space of issues and reviews. Using t-SNE projections to examine high-dimensional embeddings in two dimensions, we analyze how different models organize these artifact types in the semantic space. The visualizations reveal how issues and reviews end up positioned far from each other, reflecting the different language used in each artifact type, complementing the quantitative similarity analysis and providing insights into how each model represents meaning in high-dimensional space.

To complement the quantitative analysis, t-SNE visualizations were generated to examine how each embedding model structures the semantic space. These visualizations project high-dimensional embeddings into two dimensions while preserving local neighborhood relationships, allowing for qualitative inspection of how issues and reviews are positioned relative to each other in the semantic space.

Figures 5.1 through 5.5 show t-SNE projections for the anuke_mindustry project across all five embedding models. These visualizations reveal how different models organize the semantic space and demonstrate that issues and reviews end up positioned far from each other, reflecting the different language characteristics used in each artifact type.

5.1 Jina Embeddings

Figure 5.1 shows the t-SNE projection using Jina embeddings. The visualization reveals two primary clusters: reviews (golden/brown) form a dense cluster on the left side, with two distinct sub-clusters visible, while issues (pink) cluster on the right side. The spatial separation between these clusters reflects how the different language used in issues and reviews results in their embeddings being positioned far apart in the semantic space. There is some intermixing in the central region, suggesting that some issues and reviews share semantic similarities despite their different linguistic characteristics.

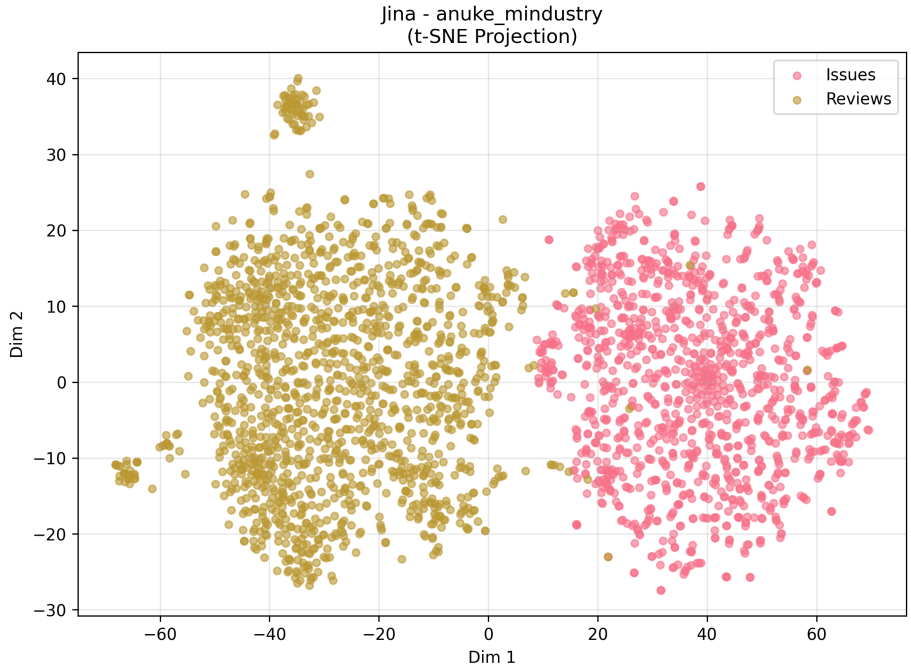

Figure 5.1: *t-SNE visualization of the semantic space for anuke_mindustry project using Jina embeddings. Issues (pink) and reviews (golden) form distinct clusters with some central overlap.*

5.2 OpenAI Large Embeddings

The t-SNE projection using OpenAI Large (Figure 5.2) shows strong spatial separation between issues and reviews. Reviews form a dense cluster on the left with a smaller sub-cluster in the bottom-left, while issues cluster on the right with a denser sub-cluster in the top-right. The central region shows minimal overlap, indicating that issues and reviews end up positioned far apart in the semantic space, reflecting the different language used in each artifact type.

The strong separation between issues and reviews aligns with the quantitative results, where OpenAI Large showed high sensitivity to semantic relationships. The model's ability to create distinct, well-separated clusters in the semantic space reflects its capacity to capture and distinguish linguistic differences between artifact types. When OpenAI Large does assign higher similarity scores to issue-review pairs (as observed in the relevance-level analysis, particularly at Level 3 where it showed the second-highest median similarity), it indicates genuine semantic relationships rather than spurious matches. The minimal overlap in the visualization corresponds to the model's conservative approach: it maintains clear boundaries between artifact types while still recognizing meaningful semantic connections when they exist, resulting in higher similarity scores for pairs that human evaluators considered moderately to highly relevant.

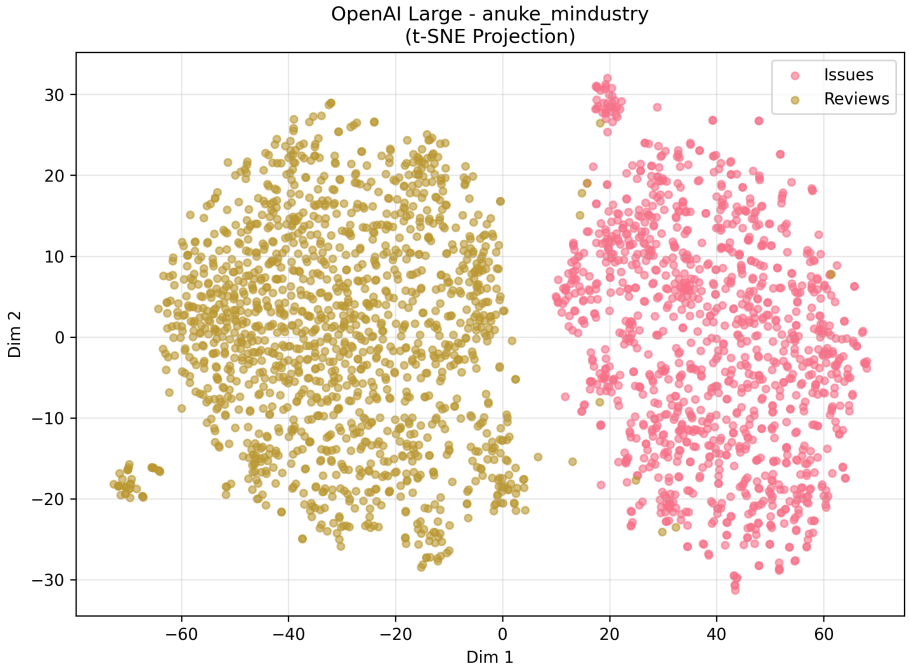

Figure 5.2: *t-SNE visualization using OpenAI Large embeddings. Strong separation between issues and reviews with minimal overlap.*

5.3 OpenAI Small Embeddings

Figure 5.3 displays the t-SNE projection for OpenAI Small. Similar to OpenAI Large, this model shows clear spatial separation between issues (right cluster) and reviews (left cluster). The reviews cluster has a smaller isolated sub-cluster in the bottom-left, while the issues cluster shows high density around the center-right. The separation is strong, with the space around Dim 1 = 0 acting as a clear boundary between the two artifact types, demonstrating how the different language used in issues and reviews results in their embeddings being positioned far apart.

5.4 Gemini Embeddings

The Gemini model (Figure 5.4) shows a different clustering pattern. Reviews form a large cluster on the left with an irregular shape and a smaller isolated sub-cluster in the bottom-left. Issues cluster on the right side. While the main clusters are separated, there are some outlier points where issues appear near the reviews cluster and vice versa. This pattern reflects how Gemini organizes the semantic space, with issues and reviews generally positioned far apart due to their different language characteristics.

This spatial organization aligns with the quantitative findings where Gemini showed more conservative similarity scores at intermediate relevance levels. The presence of outlier points in the visualization, where some issues appear near the reviews cluster and vice versa, corresponds to Gemini's behavior in the quantitative analysis: while the model generally positions issues and reviews far apart (reflecting their different language), it assigns lower

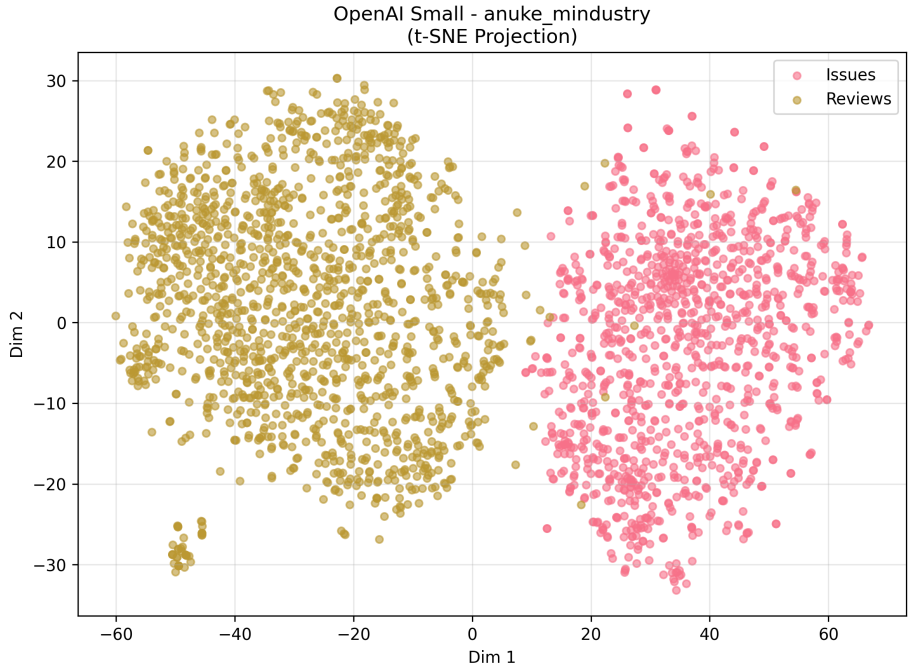

Figure 5.3: *t-SNE visualization using OpenAI Small embeddings. Clear separation with distinct boundary between issues and reviews.*

similarity scores even to pairs that human evaluators considered moderately relevant. This conservative approach contrasts with OpenAI Large's behavior: whereas OpenAI Large maintains clear boundaries while recognizing meaningful semantic connections (resulting in higher similarity scores for moderately relevant pairs), Gemini requires stronger semantic signals before assigning positive similarity values. This alignment between the structural organization of the semantic space and the similarity scoring behavior demonstrates consistency in how Gemini captures and represents linguistic differences between issues and reviews, though with a more conservative threshold than OpenAI Large.

5.5 Cohere Embeddings

Figure 5.5 shows the t-SNE projection for Cohere embeddings. The visualization reveals two well-separated clusters: a large, spread-out cluster of issues on the left and central regions, and a more compact cluster of reviews on the right. The reviews cluster forms a cohesive, oval-like shape, while the issues cluster is more diffuse. The clear separation with minimal overlap shows that issues and reviews end up positioned far apart in the semantic space, reflecting the different language used in each artifact type.

The diffuse nature of the issues cluster suggests that Cohere captures more nuances and differences between issues, indicating that the model better understands the technical language that developers use in issues. This diffusion reflects the variety of technical vocabulary, problem descriptions, and domain-specific terminology present in different issues, with Cohere's embeddings positioning them according to these linguistic subtleties

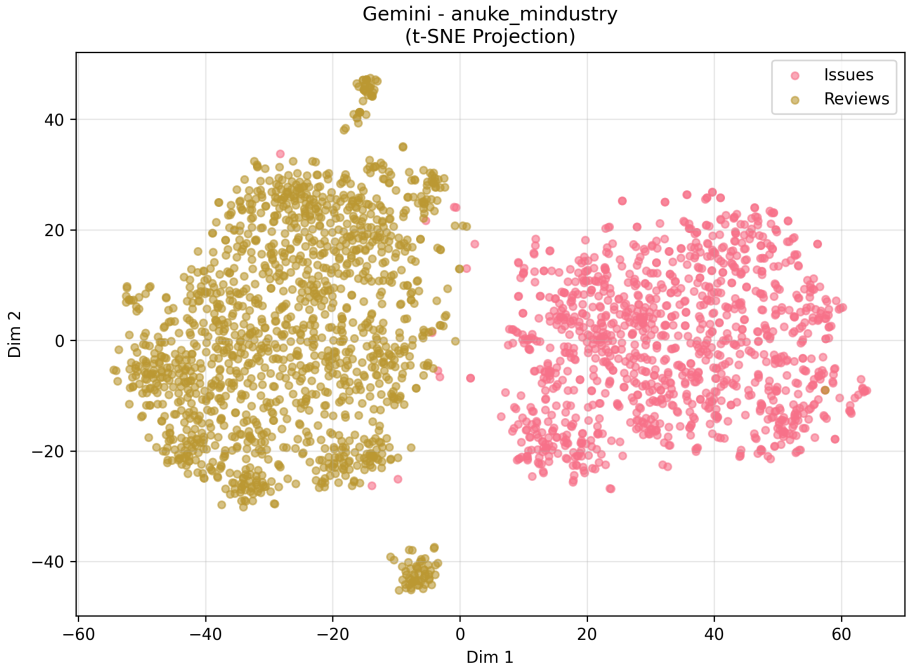

Figure 5.4: *t-SNE visualization using Gemini embeddings. Distinct clusters with some outlier points indicating semantic overlap.*

rather than grouping them tightly together.

5.6 Comparative Analysis of Semantic Space Structure

The t-SNE visualizations reveal several important patterns:

- **Universal spatial separation:** All five models show issues and reviews positioned far apart in the semantic space, forming distinct clusters. This separation reflects the different language used in each artifact type, with embedding models capturing these linguistic differences in their vector representations. The fact that this pattern appears consistently across all models suggests that the linguistic differences between issues and reviews are fundamental and well-captured by embedding models.
- **Varying cluster shapes:** While all models show spatial separation, they differ in cluster shapes and densities. OpenAI Large and OpenAI Small produce more compact, well-defined clusters, while Jina and Cohere show more diffuse distributions. Gemini shows intermediate behavior with some outlier points. These differences suggest that while all models capture the linguistic differences between artifact types, they organize the semantic space differently.
- **Sub-cluster patterns:** Several models (particularly Jina, OpenAI Large, and OpenAI Small) reveal sub-clusters within the main clusters, suggesting that these models capture finer-grained semantic distinctions within each artifact type. These sub-clusters may reflect different linguistic patterns or content characteristics. For example,

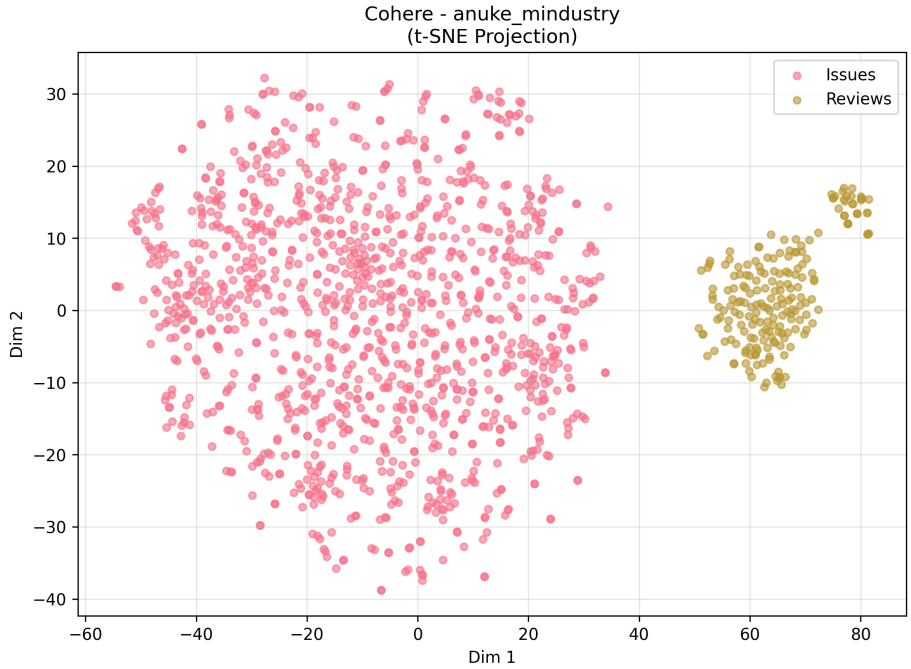

image.png

Figure 5.5: *t-SNE visualization using Cohere embeddings. Well-separated clusters with compact reviews cluster and more diffuse issues cluster.*

within the reviews cluster, sub-clusters might group reviews that share similar language patterns, such as template-based reviews or reviews with specific linguistic structures. Similarly, within the issues cluster, sub-clusters might correspond to different types of issues (e.g., bug reports, feature requests) that use distinct language patterns. The presence of these sub-clusters demonstrates that embedding models capture not only the broad linguistic differences between issues and reviews, but also finer-grained semantic distinctions within each artifact type.

- **Alignment with quantitative results:** Models that showed stronger spatial separation in the visualizations (OpenAI Large, OpenAI Small) also tended to show higher similarity scores at intermediate relevance levels in the quantitative analysis, suggesting consistency between how models organize the semantic space and how they assign similarity scores.

Chapter 6

Conclusion

This work investigated whether the choice of embedding model significantly influences the performance of semantic similarity tasks in software engineering, specifically in the context of matching user reviews with development issues. Through a systematic comparison of five embedding models (Jina, OpenAI Large, OpenAI Small, Gemini, and Cohere) across four software projects, we addressed the fundamental question raised in the introduction: *to what extent does the choice of embedding model actually influence the final outcomes of a given task?*

6.1 Main Findings

Our analysis reveals a nuanced answer: while embedding models show remarkable convergence at the extremes of the relevance spectrum, they exhibit meaningful differences at intermediate levels and in overall similarity score distributions. This finding has important implications for both researchers and practitioners.

At the extremes - clearly irrelevant pairs (Level 0) and highly relevant pairs (Level 5) - all five models produce comparable results. This convergence suggests that when semantic relationships are unambiguous, different embedding models capture similar patterns, validating the general approach of using embeddings for semantic similarity tasks in software engineering. However, at intermediate relevance levels (Levels 2–3), significant differences emerge. Jina and OpenAI Large consistently assign higher similarity scores to moderately relevant pairs, while Gemini and Cohere tend to be more conservative, requiring stronger semantic signals before assigning positive similarity scores.

The distribution analysis across all issue-review pairs reveals consistent patterns: Gemini and Jina consistently outperform other models in terms of both median similarity scores and consistency across projects, while Cohere shows a more conservative approach that may reduce false positives but potentially increase false negatives. These patterns hold across different projects, suggesting that model characteristics (architecture, training data, optimization objectives) have a stronger influence on similarity scoring than project-specific factors.

The t-SNE visualizations confirm that all models successfully separate issues and reviews into distinct clusters, demonstrating that embedding models capture characteristics that differentiate these artifact types. However, the structural organization varies: OpenAI models produce more compact, well-defined clusters, while Jina and Cohere show more diffuse distributions. These structural differences align with the quantitative findings, suggesting consistency between how models organize semantic space and how they assign similarity scores.

6.2 Implications for Practice

The findings suggest that model selection matters most when dealing with ambiguous or borderline cases, where semantic relationships are less clear-cut. For strongly relevant or clearly irrelevant pairs, the choice of embedding model appears to have less impact on the final outcomes. This has practical implications:

- **For applications requiring high precision:** Models like Gemini and Jina, which show higher similarity scores and tighter distributions, may be preferable when the goal is to identify highly relevant matches with confidence.
- **For applications requiring high recall:** More conservative models like Cohere may be suitable when minimizing false positives is critical, though this may come at the cost of missing some relevant matches.
- **For borderline cases:** The significant differences at intermediate relevance levels suggest that practitioners should carefully consider their tolerance for false positives and false negatives when selecting an embedding model, particularly when dealing with ambiguous semantic relationships.
- **For cost-sensitive applications:** The similar performance of OpenAI Large and OpenAI Small suggests that the smaller model may be sufficient for many use cases, potentially reducing computational costs and API expenses.
- **Consensus-based filtering:** An interesting approach emerging from our findings is to use consensus between models as a quality filter. By only considering matches that are marked as relevant by all (or a majority of) models, practitioners can potentially achieve higher precision, as matches agreed upon by multiple models are more likely to represent genuine semantic relationships. This consensus approach leverages the convergence observed at extreme relevance levels, where models tend to agree, while filtering out ambiguous cases where models diverge.

6.3 Final Remarks

This work contributes to the growing body of research on embedding models in software engineering by providing empirical evidence on how different models behave in semantic similarity tasks. The consistent positive correlation between similarity scores and human-assigned relevance levels across all models validates the general approach of using embeddings for semantic similarity tasks. However, the observed differences at

intermediate levels and in distribution patterns demonstrate that embedding models do not perform similarly and must be chosen carefully based on the specific requirements of each application.

The finding that model selection matters more for ambiguous cases than for clear-cut ones highlights the importance of understanding model characteristics when dealing with borderline semantic relationships. Different models exhibit distinct behaviors: some are more sensitive to moderate semantic relationships, while others require stronger signals before assigning positive similarity scores. These differences are not trivial and can meaningfully impact the outcomes of semantic similarity tasks.

Ultimately, this study demonstrates that while embedding models share fundamental capabilities for capturing semantic relationships, they are not interchangeable. The choice of model can meaningfully impact outcomes, particularly in scenarios where semantic relationships are ambiguous or where precision and recall requirements differ. Practitioners and researchers must therefore carefully evaluate and select embedding models based on their specific use case, tolerance for false positives and false negatives, and the characteristics of the semantic relationships they aim to capture.

References

[Hutchins 2004] W. John Hutchins. "The georgetown-ibm experiment demonstrated in january 1954". In: *Proceedings of the 6th Conference of the Association for Machine Translation in the Americas: Technical Papers*. Ed. by Robert E. Frederking and Kathryn B. Taylor. Washington, USA: Springer, 2004. url: https://aclanthology.org/2004.amta-papers.12/ (cit. on p. 3).

[Jurafsky and Martin 2023] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3rd. Pearson, 2023 (cit. on p. 3).

[Li *et al.* 2024] Jianfeng Li, Minghui Li, Yang Liu, Liang Zhang, and Yanchun Wang. "HYDBre: a hybrid retrieval method for detecting duplicate software bug reports". In: *Proceedings of the 2024 IEEE International Conference on Software Maintenance and Evolution*. 2024, pp. 1–12. url: https://ieeexplore.ieee.org/document/10818310/ (cit. on p. 3).

[Pilone *et al.* 2025] A. Pilone, M. Raglianti, M. Lanza, F. Kon, and P. Meirelles. "Automatically augmenting GitHub issues with informative user reviews". In: *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2025 (cit. on pp. 1, 4, 7–10).

[Rahman *et al.* 2024] Md. Saiful Rahman, Mohammad Al Hasan, and Mohammad Rafiqul Islam. "Detecting hard-coded credentials in software repositories via LLMs". *ACM Transactions on Software Engineering and Methodology* 33.4 (2024), pp. 1–28. doi: 10.1145/3744756. url: https://dl.acm.org/doi/10.1145/3744756 (cit. on p. 4).

[Tripathy *et al.* 2021] B. K. Tripathy, S. Anveshrithaa, and Shrusti Ghela. "T-distributed stochastic neighbor embedding (t-sne)". In: *Data Science and Innovations for Intelligent Systems*. CRC Press, 2021, p. 13 (cit. on p. 5).

[Vaswani *et al.* 2017] Ashish Vaswani *et al.* "Attention is all you need". *Advances in Neural Information Processing Systems* 30 (2017) (cit. on p. 4).

[Zhang *et al.* 2024] Yuhang Zhang, Lei Wang, Xiaoyuan Chen, and Hao Liu. "SICode: embedding-based subgraph isomorphism identification for bug detection". In: *Proceedings of the 2024 IEEE/ACM International Conference on Software Engineering*. 2024, pp. 1–12. URL: https://ieeexplore.ieee.org/document/10556446 (cit. on p. 3).