Published: 17 June 2025

EASE '25: Evaluation and Assessment in Software Engineering
June 17 - 20, 2025
Istanbul, Turkiye

SHORT-PAPER

Towards Leveraging Large Language Model Summaries for Topic Modeling in Source Code

MICHELE CARISSIMI, University of Milan, Milan, MI, Italy

MARTINA SALETTA, University of Bergamo, Bergamo, BG, Italy

CLAUDIO FERRETTI, University of Milan, Milan, MI, Italy

EASE '25: Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering (June 2025)
https://doi.org/10.1145/3756681.3757026
ISBN: 9798400713859

Towards Leveraging Large Language Model Summaries for Topic Modeling in Source Code

Michele Carissimi
University of Milano-Bicocca
Milano, Italy
m.carissimi4@campus.unimib.it

Martina Saletta
University of Bergamo
Bergamo, Italy
martina.saletta@unibg.it

Claudio Ferretti
University of Milano-Bicocca
Milano, Italy
claudio.ferretti@unimib.it

Abstract

Understanding source code is a topic of great interest in the software engineering community, since it can help programmers in various tasks such as software maintenance and reuse. Recent advances in large language models (LLMs) have demonstrated remarkable program comprehension capabilities, while transformer-based topic modeling techniques offer effective ways to extract semantic information from text. This paper proposes and explores a novel approach that combines these strengths to automatically identify meaningful topics in a corpus of Python programs. Our method consists in applying topic modeling on the descriptions obtained by asking an LLM to summarize the code. To assess the internal consistency of the extracted topics, we compare them against topics inferred from function names alone, and those derived from existing docstrings. Experimental results suggest that leveraging LLM-generated summaries provides interpretable and semantically rich representation of code structure. The promising results suggest that our approach can be fruitfully applied in various software engineering tasks such as automatic documentation and tagging, code search, software reorganization and knowledge discovery in large repositories.

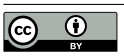
CCS Concepts

• **Information systems** → **Summarization**; • **Software and its engineering** → **Documentation**; *Automated static analysis*; Software maintenance tools.

Keywords

source code analysis, topic modeling, transformers, source code concept location

ACM Reference Format:
Michele Carissimi, Martina Saletta, and Claudio Ferretti. 2025. Towards Leveraging Large Language Model Summaries for Topic Modeling in Source Code. In *Evaluation and Assessment in Software Engineering (EASE '25), June 17–20, 2025, Istanbul, Turkiye.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3756681.3757026



EASE '25, Istanbul, Turkiye

ACM ISBN 979-8-4007-1385-9/25/06
https://doi.org/10.1145/3756681.3757026

1 Introduction

Understanding source code is a fundamental challenge in software engineering, since it helps programmers and, more in general, people that work on source code in various tasks such as code maintenance and code reuse.

At the same time, topic modeling has proven to be an effective technique for extracting structured information from large text corpora. By identifying latent themes within a collection of documents, topic modeling enables applications such as document classification, content summarization, and knowledge discovery [5, 20].

In the context of source code, topic modeling can help uncover high-level concepts, improve code organization, and help in mining large software repositories. Traditional topic modeling approaches, however, primarily rely on natural elements of code, such as identifiers and comments, which programmers introduce to enhance readability [2, 12, 13]. This dependence can limit their effectiveness when such elements are missing, inconsistent, or poorly maintained.

In this paper, we propose a novel approach that combines the ability of LLMs to comprehend source code and perform document embeddings, together with topic modeling techniques to extract meaningful topics from code. Specifically, we choose Python code and we begin by processing functions where comments and docstrings are removed, and function names are obfuscated to eliminate explicit semantic clues. We then use an LLM to generate summaries of these functions, capturing their key functionalities. Finally, we apply topic modeling to this corpus of LLM-generated summaries, allowing us to infer meaningful topics even in the absence of traditional natural-language cues. This approach offers a robust method for analyzing and organizing code, with potential applications in program comprehension tasks such as automatic documentation, software reorganization, code search, and knowledge discovery in large repositories.

Data and code for replicating our experiments are publicly available[1].

2 Summary-based Topic Modeling

Recent advances in large language models (LLMs) have demonstrated remarkable capabilities in several domains, and their potential is also evident in the program comprehension field [15]. Besides, transformer-based topic modeling techniques have proven effective in extracting semantic information from text [16].

This paper explores a novel approach that integrates these findings to automatically identify meaningful topics in a corpus of Python programs.

[1] https://zenodo.org/records/15036066

In the literature, there exist studies that explored the application of topic modeling techniques to source code, but all these approaches primarily rely on the "natural elements" (e.g. identifiers, comments, and docstrings) that programmers introduce to enhance readability and maintainability [2, 11, 12, 18].

In this study, we aim to define a topic modeling technique for source code that remains effective even when such natural elements are removed. To this end, we (1) start with Python functions where comments and docstrings are removed and function names are obfuscated to eliminate any explicit hints about their purpose. We then (2) leverage a large language model (LLM) to generate summaries of these functions, capturing their underlying semantics without relying on manually introduced textual cues. Finally, (3) we apply topic modeling to this corpus of LLM-generated summaries to extract meaningful topics, as outlined in Figure 1. This approach allows us to show whether topic modeling can effectively uncover the structure and organization of code even in the absence of conventional natural-language annotations.

In a topic modeling context, for each document d of the corpus D, the model M defines a function $topicDist_M \colon D \to \mathbb{R}^n$ that returns the topic distribution for the given document in the form of a probability distribution, with n representing the total number of modeled topics. In other words, for each $d \in D$, $topicDist_M(d) = (p_{d_1}, \ldots, p_{d_n})$ and $p_{d_1} + p_{d_2} + \cdots + p_{d_n} = 1$. Similarly, each topic $T_i \in \{T_1, \ldots, T_n\}$ is represented through a vector of weights $T_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,|W|})$, where each $w_{i,k}$ represents the relevance that the word k has for the topic T_i and $|W|$ is length of the vocabulary. Finally, we associate each document d with a topic, where a topic is indexed with an integer $1 \leq i \leq n$, with the function $topicAssign_M \colon D \to \{1, \ldots, n\}$, defined as $topicAssign(d) = i$ if $p_{d_i} = \max(p_{d_1}, \ldots, p_{d_n})$.

To validate our approach, we trained two topic models M_{summ} and M_{doc} starting from different representations of the same instances: function summaries produced by an LLM for M_{summ} and docstrings for M_{doc}, respectively. Notice that function summaries are obtained starting from functions in which comments have been removed and the name of the function obfuscated, so as to remove the intrinsic code naturalness [1].

According to this notation, we now introduce the four metrics we used to assess the robustness of our results. We consider the docstrings as the golden standard, and we measure how the inference on M_{doc} differ when made on the code summaries or on the original function names. In other words, we define four distances between the inferences of a model.

d_{MSE}. This distance is based on the mean squared error between the probability distributions associated with the topic assignment to each document. Given a model M with n topics, we define the distance between topics of two documents $topicDist_M(d) = (p_{d_1}, \ldots, p_{d_n})$ and $topicDist_M(d') = (p_{d'_1}, \ldots, p_{d'_n})$ as:

$$d_{MSE}(d, d') = \frac{1}{n} \sum_{i=1}^{n} \left(p_{d_i} - p_{d'_i}\right)^2$$

d_{TOP}. Given two topic distributions inferred for documents d and d', $d_{TOP}(d, d')$ is the number of common topics between the 10 most probable of each topic distribution; a higher value is associated to a better similarity between the topic distributions,

d_{TOP_W}. Given two topic distributions inferred for documents d and d', $d_{TOP_W}(d, d')$ is the average cosine-similarity between the 10 most probable topics of the topic distributions, as represented by the respective rows in the topic-term relevance matrix defined by the topic model; a higher value is associated to a better similarity between the topic distributions,

$d_{\cap}$. Given assigned topics $topicAssign(d)$ and $topicAssign(d')$, $d_{\cap}(d, d')$ is the number of common words between the 5 most relevant words of each topic; a higher value is associated to a better similarity between the topics; notably, this metric can be computed even between topics assigned by different topic models.

Our results are promising. From a qualitative perspective, this is evident in Table 1, where we can observe meaningful and well-separated topics. To further assess the effectiveness of our approach, we compare, by using the above defined distance measures, the obtained topics with those derived from existing docstrings, showing that our method achieves similar scores across different evaluation metrics. Finally, we demonstrate that when using only function names, the results are significantly worse. This indicates that our approach can successfully identify and model topics from source code by relying solely on code structure rather than leveraging natural-language elements.

3 Experimental Settings

The experiments described in this study have been carried out on a set 10.000 Python functions, randomly sampled from the Python partition of CodeSearchNet [10], a popular dataset widely used in the software engineering community that includes a large corpus of source code snippets and their corresponding docstrings.

The records in the dataset are JSON objects, and for each instance we considered the following fields:

- `func_name`: the attribute representing the name of the function;
- `whole_func_string`: the attribute representing the source-code snippet;
- `func_documentation_string`: the attribute representing the docstring for each function.

As fully motivated in Section 2, prior to asking the LLM to provide the summaries, all function names in their declarations were replaced with a fixed, randomly generated, placeholder string, to eliminate any semantic information that could reveal the purpose of the functions. This was achieved by using a regular expression to identify and replace the function names in their definition lines. This step ensured that the LLM focused solely on understanding the logic within the source code rather than leveraging potentially meaningful naming conventions. This preprocessing step not only ensured that the LLM's summarization focused purely on the logic of the source code, but also enabled a later comparison between the topic distributions inferred from:

(1) summarizations generated by the Gemma [7] model from the processed source code
(2) tokenized function names
(3) original docstrings

Additionally, with the same goal, all the comments were removed from the source code snippets during preprocessing with the use of

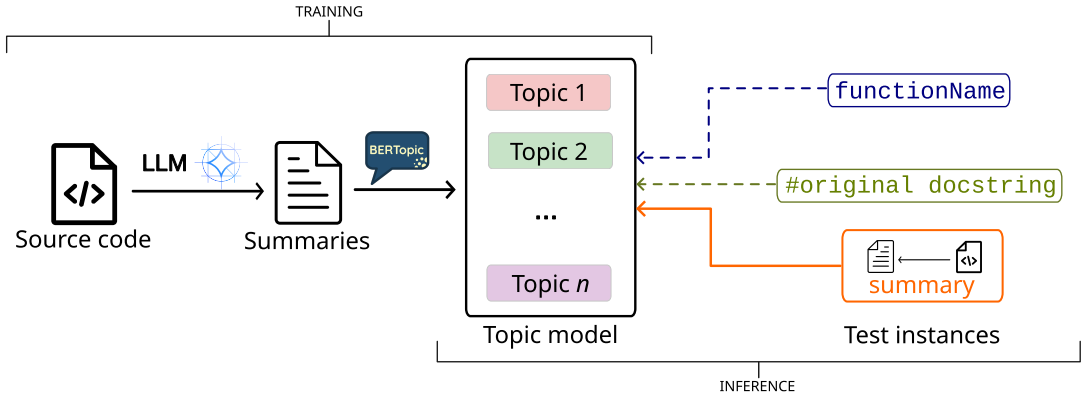


Figure 1: Study overview. Source code summaries generated by an LLM are used to identify the latent topics. This topic model can be used for inferring code information from different code representations.

regular expressions to replace comments with empty strings. This was done to eliminate auxiliary information that might simplify the summarization task, ensuring that the LLM's output relied purely on the syntactic and structural elements of the code.

In order to query the LLM, we constructed the prompt to be used by concatenating a fixed portion, which explains the task and how to format the output, and a variable portion, containing the source-code snippet to be explained.

- **Base query:** "`Consider the following source code and provide a description of its purpose.`"
- **Prompt template:** `base_query + source_code + " The output should follow this format: ##### Description: <source code description>"`

The source code description generation process was conducted using the Gemma2 2B-it model [6, 7], a 2-billion-parameter language model designed for text generation. The maximum token limit for generated descriptions was set to 1024 to generate concise and comprehensive outputs. Queries were executed via Huggingface's pipeline API [2], which provided an interface for integrating the model into the experimental pipeline. The choice of Gemma2 2B-it was driven by its balance of computational efficiency and performance.

The resulting data underwent two preprocessing steps. First, all text was converted to lowercase, and numbers, punctuation marks, and commonly defined stopwords were removed. The stopwords were identified using the default collection provided by the NLTK library [3], which is widely used for natural language processing tasks.

Next, the records were analyzed to identify words appearing in at least 75% of documents. These high-frequency words were removed under the assumption that they carry limited semantic value and contribute minimally to distinguishing topics. By excluding these terms, the preprocessing ensured that the corpus retained only those words more likely to provide meaningful insights during the topic modeling process.

BERTopic [9] is a topic modeling algorithm that uses embeddings from transformer-based models combined with clustering techniques to group semantically related documents. Unlike traditional methods like Latent Dirichlet Allocation (LDA) [4], which rely on term frequency distributions, BERTopic utilizes vector representations of text to capture contextual relationships between words and phrases. These embeddings are reduced in dimensionality using algorithms such as UMAP [14], enabling efficient clustering and topic extraction. This approach is particularly useful for analyzing large text corpora where semantic meaning is essential for identifying coherent topics.

After preprocessing, the dataset was used as a corpus for the topic modeling process. BERTopic was applied to identify coherent topics within the dataset. The dimensionality reduction step, a crucial part of BERTopic's pipeline, was performed using UMAP. All the experiments are performed with the following parameters:

- `nr_topics = 40` and `topic_size = 25` to avoid topics having few associated documents
- `metric = 'cosine'` to measure the distance among documents in the embedding space
- `nr_neighbors = 10` and `min_distance = 0.01` for the dimensionality reduction with UMAP

Selecting the optimal number of topics in topic modeling is an open problem, as there is no universally accepted method for determining this parameter. For this study, the number of topics (`nr_topics`) was set to 40 after running the model multiple times with varying values and evaluating the results. The decision was guided by a balance between coherence score, measuring the internal consistency of the topics, and interpretability, ensuring that the topics identified meaningful patterns within the corpus. This iterative approach allowed the selection of a number that provided both statistically robust and human-interpretable results.

The initial dataset, composed of 10.000 records, was split creating a 9.500 records train set and a 500 records evaluation set. After applying the topic modeling process, BERTopic generated a series of topics. Each topic is represented by a ranked list of words from the corpus, sorted by their relevance to that topic. This ranking provides insight into the defining terms for each topic, aiding in interpretation.

[2]https://huggingface.co/docs/transformers, last visit November 2024

To associate topics to a document, in what is considered the inference step, BERTopic computes a probability distribution across all modeled topics, indicating the likelihood of each topic being relevant to the document. Also, a specific topic is assigned if its probability is sufficiently high relative to the others. If no topic meets this threshold, the document is labeled as "-1" or "outlier," indicating that it does not fit well into any single topic. This mechanism ensures that only documents with strong associations to a specific topic are classified, while ambiguous or unrelated documents remain unassigned.

4 Results

The results of the topic modeling process were evaluated using multiple metrics, with a primary focus on the coherence score, which quantifies the semantic consistency among the most relevant words within each topic. A low coherence score indicates that the top words are semantically distant, making the topic harder to interpret. In contrast, a high coherence score signifies strong semantic connections among the top words, leading to topics that are easier to understand and label. In this study, the C_v coherence score was utilized. This metric combines a sliding window approach, word co-occurrence counts, and vector representations of words to measure cosine similarity among the top words in a topic [17, 19]. By leveraging both statistical co-occurrence and semantic similarity, C_v provides a robust evaluation of topic interpretability, making it suitable for assessing the quality of topics derived from domain-specific datasets.

A list of topics resulting from our modeling process are reported in Table 1. The table shows the five most relevant words for each topic, along with a tentative title that summarizes the topic scope and the coherence score of each topic. In the table, Topic 32 has a coherence value far lower than all others, but even there the top 5 words representing it all belong to the same domain related to code managing colors.

The coherence scores for all topics are also visualized in Figure 2, which also highlights the average coherence score across topics. The average coherence score was approximately 0.60, indicating moderate interpretability and the presence of meaningful word clusters. The coherence score of 0.60 suggests that the topics identified by the model are both relevant and interpretable. This level of coherence reflects the model's ability to extract meaningful patterns from a highly technical and structured corpus. We also computed the average coherence score of the topics modeled from the set of original docstrings, which we can consider as a semantically rich content, but this value resulted to be only 0.38, thus suggesting the value of our result.

Documents and their associated topics can also be visualized in a two-dimensional space (Figure 3), where each point represents a document. The positioning of the points reflects their semantic embeddings, generated through dimensionality reduction using UMAP [14]. Semantic similarity between documents is indicated by their proximity, while the color assigned to each document encodes the most relevant topic for that document.

The presence of dense and well-separated clusters indicates that the model effectively grouped semantically similar documents, suggesting that the identified topics are coherent and interpretable.

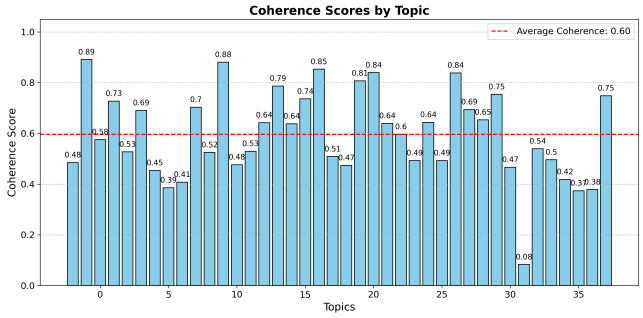


Figure 2: Coherence scores

Some overlap between clusters or the presence of scattered points reflect the nature of the data, where a single document can be associated to more than one strongly relevant topic. In fact, as detailed in Section 2, the topic inference produces, for each document, a probability distribution over the whole set of modeled topics.

5 Discussion

We assess the value of our approach according to different usage contexts. The key element is the exploitation of the internal representation of an LLM in order to create semantically rich natural language descriptions of input source code. Previous proposals regarding the creation of topic models for source code ([2, 12]) required the availability of natural language elements in the source code, such as comments or meaningful identifier. We move along a different way, since topics are modeled based on the natural text generated by the LLM from the input source code. This opens to applications to source code without accompanying documentation, or without useful identifiers. Consistently, we experimented our tool always on code where we removed any comment and obfuscated the functions' identifiers.

The method can be evaluated in terms of internal quality of the topics it creates, and we showed in Section 4 that we generate topics with high coherence measure. But also, the method exhibits advantageous performance when compared against techniques based on natural language elements from source code. To this aim, we can consider two main reference usage contexts: one where only the code is available, and one with the availability of developer authored documentation. In these two settings, the value of the topics generated or assigned with our technique has been assessed in terms of their alignment with the topics modeled based on the documentation and then assigned to some code from its own documentation. We consider these latter topics the reference target. The results, presented in Table 2, show that the summarization produced by the LLM are always close to the reference topics, and therefore it compensates the lack of documentation.

In detail, in Table 2 we show the average results produced according to the metrics defined in Section 2 when comparing topics inferred under different settings against the reference target topics. The first two lines evaluate the topic inferences produced on the model M_{doc} built from docstrings, but first taking as input the summaries generated by the LLM and then taking as input the tokenized function names, respectively. The numbers show that

Table 1: Topic models

Topic	Top words	Label	Coherence
0	request, url, respnse, api, http	HTTP Communication	0.89
1	message, device, network, system, address	Network Communication	0.58
2	array, calculates, matrix, calculation, values	Matrix Operations	0.73
3	myclass, attribute, instance, named, within	Object-oriented Programming	0.53
4	path, directory, files, module, filename	Filesystem	0.69
5	database, query, import, table, sql	Database	0.45
6	configuration, yaml, commandline, yang, arguments	Configuration Files	0.39
7	csv, gene, bioseqfeature, featuretype, sequence	Genomic Data	0.41
8	time, date, datetime, dateutilparser, parse	Date and Time Parsing	0.70
9	image, pil, images, pixel, color	Image Processing	0.52
10	model, layer, training, learning, tensor	Neural Networks	0.88
11	log, logging, message, exception, logger	Logging	0.48
12	widget, window, gui, layout, button	GUI Components	0.53
13	dictionary, key, keys, keyvalue, dictionaries	Dictionaries (data structures)	0.64
14	pattern, regular, string, match, word	Regular Expressions	0.79
15	plot, axes, import, plots, astropyvisualization	Data Visualization	0.64
16	byte, bytes, encoding, string, encoded	Data Encoding	0.74
17	node, tree, child, children, parent	Trees (data structures)	0.85
18	typing, import, dict, future, optional	Type Annotations	0.51
19	description, string, information, format, field	Field Descriptions	0.47
20	hash, key, signature, algorithm, password	Cryptographic Hashing	0.81
21	graph, nodes, node, edges, edge	Graphs (data structure)	0.84
22	package, repository, dependencies, packages, git	Package Management	0.64
23	xml, element, callback, config, configuration	XML Configuration	0.60
24	html, template, content, element, rended	Websites	0.49
25	decorator, decorated, original, wrapper, acts	Function Decorators	0.64
26	json, dictionary, python, string, path	JSON	0.49
27	command, commands, output, shell, execute	Shell Commands	0.84
28	dataframe, columns, column, pandas, rows	Pandas Dataframes	0.69
29	aws, region, bucket, service, profile	AWS (Amazon Web Services)	0.65
30	table, row, column, cell, rows	Tables	0.75
31	cache, caching, cached, torrent, key	Caching	0.47
32	color, rgb, colormathcolorconversions, colormathcolorobjects, hscolor	Colors	0.08
33	series, time, percentile, calculates, index	Time Series	0.54
34	filter, filters, filtering, criteria, filtered	Filters	0.50
35	version, string, prerelease, versions, major	Versioning	0.42
36	language, translation, translations, languages, header	Language Translation	0.37
37	email, user, roles, address, permissions	Email Management	0.38
38	download, downloaded, downloads, directory, url	File Downloads	0.75

Table 2: Comparison between reference topics and topics inferred under two models and on different inputs (metrics from Section 2, discussion in Section 5).

		d_{MSE}	d_{TOP}	d_{TOPw}	$d_{\cap}$
M_{doc}	Summaries	0.013	**3.46**	**0.54**	**3.71**
	Names	0.013	3.11	0.51	2.18
M_{summ}	Summaries	N/A	N/A	N/A	2.37

the topics based on LLM-generated summaries are more similar to reference topics than the ones generated from the function names and their natural semantic. Interestingly, a remark can be done about the apparent effectiveness of inferring topics starting from function identifiers. It is good practice that the developer coding a function assigns to it a meaningful and representative identifier. As suggested by [1], the naturalness of those identifiers is a rich information, based on which machine learning systems can be trained to successfully perform tasks such as automatic function name suggestion. On the other hand, it has been shown that when summarizing source code, the removal of natural content leads to a significant drop in performance [8]. This can explain the quality we measured for topics generated from identifiers when comparing them to the reference topics generated from the documentation. Nonetheless, the topic defined from the summary texts generated by the LLM in our method appear to be better. Finally, in the last row of Table 2 we evaluate against reference topics the ones inferred from generated summaries, but in this case the topic model is M_{summ}, which has been built from summaries itself. In this setting only the $d_{\cap}$ metric could be applied, since the available topics are

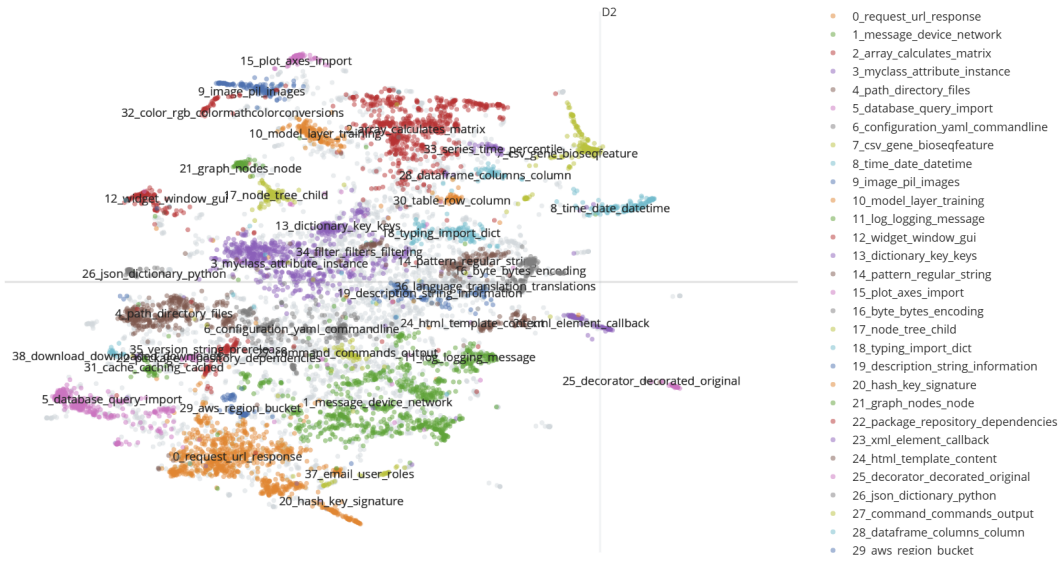


Figure 3: Document embeddings visualization. Each point represents a code instance, with colors indicating the assigned topics.

not identical to the topics of the model built from docstrings, in terms of vocabulary and associated words relevance values. The similarity to the reference topic assignment is still good, with an average number of 2.37 (over 5) top relevant terms shared with the top terms of the reference topic.

Overall, our approach offers a performance which is close to that of topic modeling with code documentation available, but more robust in settings where the code is lacking good comments or meaningful function identifiers.

6 Conclusion and Further Directions

This work introduces a method to build topic models on source code corpora without requiring the availability of natural text (comments or function identifiers), by leveraging the capability of LLMs to generate natural language summaries for code snippets. The new method has better performance, according to several metrics we chose, than more traditional techniques requiring comments and documentation to build topics.

Further research could be devoted to tailoring and applying this method to other source code knowledge extraction tasks.

References

[1] Miltiadis Allamanis, Earl T. Barr, Premkumar T. Devanbu, and Charles Sutton. 2018. A Survey of Machine Learning for Big Code and Naturalness. *ACM Comput. Surv.* 51, 4 (2018), 81:1–81:37.

[2] Ted J Biggerstaff, Bharat G Mitbander, and Dallas Webster. 1993. The concept assignment problem in program understanding. In *[1993] Proceedings Working Conference on Reverse Engineering*. IEEE, 27–43.

[3] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (2003), 993–1022.

[5] Rob Churchill and Lisa Singh. 2022. The evolution of topic modeling. *Comput. Surveys* 54, 10s (2022), 1–35.

[6] Morgane Riviere et al. (Gemma Team). 2024. Gemma 2: Improving Open Language Models at a Practical Size. *arXiv* abs/2408.00118 (2024).

[7] Thomas Mesnard et al. (Gemma Team). 2024. Gemma: Open Models Based on Gemini Research and Technology. *arXiv* abs/2403.08295 (2024).

[8] Claudio Ferretti and Martina Saletta. 2023. Naturalness in Source Code Summarization. How Significant is it?. In *31st IEEE/ACM International Conference on Program Comprehension, ICPC*. IEEE, 125–134.

[9] Maarten Grootendorst. 2022. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv* abs/2203.05794 (2022). https://arxiv.org/abs/2203.05794

[10] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *arXiv* abs/1909.09436 (2019).

[11] Martina Iammarino, Lerina Aversano, Mario Luca Bernardi, and Marta Cimitile. 2020. A topic modeling approach to evaluate the comments consistency to source code. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[12] Adrian Kuhn, Stéphane Ducasse, and Tudor Gîrba. 2007. Semantic clustering: Identifying topics in source code. *Information and software technology* 49, 3 (2007), 230–243.

[13] Anas Mahmoud and Gary Bradshaw. 2017. Semantic topic models for source code analysis. *Empirical Software Engineering* 22 (2017), 1965–2000.

[14] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. 2018. UMAP: Uniform Manifold Approximation and Projection. *J. Open Source Softw.* 3, 29 (2018), 861.

[15] Daye Nam, Andrew Macvean, Vincent J. Hellendoorn, Bogdan Vasilescu, and Brad A. Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM, 97:1–97:13.

[16] Arik Reuter, Anton Thielmann, Christoph Weisser, Benjamin Säfken, and Thomas Kneib. 2025. Probabilistic Topic Modeling With Transformer Representations. *IEEE Transactions on Neural Networks and Learning Systems* (2025).

[17] Michael Röder, Andreas Both, and Alexander Hinneburg. 2015. Exploring the Space of Topic Coherence Measures. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM 2015, Shanghai, China, February 2-6, 2015*, Xueqi Cheng, Hang Li, Evgeniy Gabrilovich, and Jie Tang (Eds.). ACM, 399–408.

[18] Amir M Saeidi, Jurriaan Hage, Ravi Khadka, and Slinger Jansen. 2015. ITMViz: Interactive topic modeling for source code analysis. In *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 295–298.

[19] Shaheen Syed and Marco R. Spruit. 2017. Full-Text or Abstract? Examining Topic Coherence Scores Using Latent Dirichlet Allocation. In *2017 IEEE International Conference on Data Science and Advanced Analytics, DSAA*. IEEE, 165–174.

[20] Ike Vayansky and Sathish AP Kumar. 2020. A review of topic modeling methods. *Information Systems* 94 (2020), 101582.