CodeGenLink: A Tool to Find the Likely Origin and License of Automatically Generated Code

Daniele Bifolco
University of Sannio, Italy
d.bifolco@studenti.unisannio.it

Guido Annicchiarico
University of Sannio, Italy
g.annicchiarico@studenti.unisannio.it

Pierluigi Barbiero
University of Sannio, Italy
p.barbiero3@studenti.unisannio.it

Massimiliano Di Penta
University of Sannio, Italy
dipenta@unisannio.it

Fiorella Zampetti
University of Sannio, Italy
fzampetti@unisannio.it

***Abstract*—Large Language Models (LLMs) are widely used in software development tasks nowadays. Unlike reusing code taken from the Web, for LLMs' generated code, developers are concerned about its lack of trustworthiness and possible copyright or licensing violations, due to the lack of code provenance information. This paper proposes CODEGENLINK, a GitHub CoPilot extension for Visual Studio Code aimed at (i) suggesting links containing code very similar to automatically generated code, and (ii) whenever possible, indicating the license of the likely origin of the code. CODEGENLINK retrieves candidate links by combining LLMs with their web search features and then performs similarity analysis between the generated and retrieved code. Preliminary results show that CODEGENLINK effectively filters unrelated links via similarity analysis and provides licensing information when available.**

Tool URL: https://github.com/danielebifolco/CodeGenLink

Tool Video: https://youtu.be/M6nqjBf9_pw

***Index Terms*—Large Language Models; Code Provenance; Licensing; Trustworthiness**

I. INTRODUCTION

Large Language Models (LLMs) are nowadays widely used to perform several development tasks, ranging from code generation, testing, re-documentation, or support to the software development process [4], [7], [14].

Considering code generation, *i.e.,* the most prominent among the previously mentioned tasks, there is one key element that makes LLM promising and worrisome at the same time, when compared to code reuse from the Web. For the latter, ownership can often be inferred, enabling an assessment of the source's trustworthiness. However, this is not always possible for LLM-generated code. Furthermore, reuse and redistribution of LLM-generated artifacts can lead to license or copyright violations. Yang *et al.* [17] were the first assessing the "memorization" effect within pre-trained models, *i.e.,* models memorizing and producing source code verbatim. The reuse of this code may lead to copyright violations, as well as to issues with code attribution [3], [9]. Specifically, Colombo *et al.* [3] found that more context increases ChatGPT's reproduction of copyleft code, while higher temperature reduces this risk. Last but not least, researchers have also looked at the reuse of code from training sets [11], [16].

In a previous work [2], we analyzed the extent to which LLMs, if properly prompted, could provide relevant links to the code they generate. However, although LLMs generate multiple links, only a few are likely to be the "true origin" for the generated code.

In this paper, we propose CODEGENLINK, a tool integrated in GitHub Copilot [5] (hereafter referred to as Copilot) as a Visual Studio (VS) Code extension, that assists developers in determining the likely origin (when available) of the generated code, and the license under which it should be redistributed. The tool has two operational modes:

1) **Direct code + link retrieval (Mode 1):** The user, through the Copilot chat, asks the LLM to generate a code snippet. The LLM output is automatically used by CODEGENLINK to retrieve links similar to the automatically generated code by enabling the web search.
2) **Link retrieval (Mode 2):** The user selects the automatically generated code directly within the IDE and asks CODEGENLINK to retrieve relevant links for it. Note that in principle this mode can be used to seek the provenance of any code snippet, even if it is not automatically generated.

CODEGENLINK combines text similarity and clone detection to identify likely code origins, then inspects each relevant link to extract associated license information.

Unlike tools such as LiCoEval [16], which use code similarity analysis to check whether generated code is part of an LLM's training set, CODEGENLINK searches the Web for code snippets that match the generated code and identifies their associated licenses, allowing it to work even when the LLM's training data is inaccessible.

We have performed a preliminary evaluation of CODEGENLINK using 101 coding tasks from CODESEARCHNET [8] and 100 coding tasks from CODEREVAL [18]. The evaluation results are promising since, even if LLMs do not always provide relevant links, the similarity mechanism filters them out and only provides the users with the ones that are "likely" the origin of the code.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00358

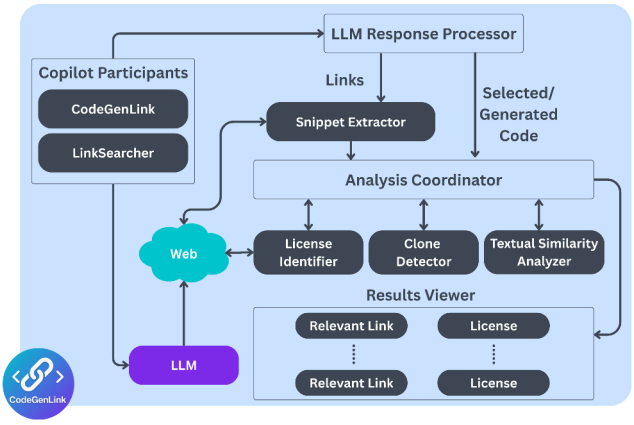

Fig. 1. CODEGENLINK Architecture

II. ARCHITECTURE

Fig. 1 depicts the CODEGENLINK's architecture. CODEGENLINK integrates with Copilot through the Chat Participants [15] interface provided by VS Code. A participant is an agent interacting with the LLM, referred to in the chat by prepending its name with a "@". CODEGENLINK has two participants as entry points supporting the two operational modes outlined in the introduction, *i.e.,* ***CodeGenLink*** (**Mode 1**) and ***LinkSearcher*** (**Mode 2**). To provide links, CODEGENLINK relies on an LLM with web search capabilities. We used OpenAI's *web-search-preview* [12], though other LLM APIs with similar support can be used.

GitHub Copilot Participants. We developed two specialized participants to support distinct operational modes:

- ***CodeGenLink***. This participant is invoked when the user interacts with the Copilot chat, prefixing the prompt with `@CodeGenLink`. It represents operational **Mode 1**, where the model first generates code that meets the user's requirements. After that, the provided output is used to prompt the model again to search the Web for relevant links related to the generated code. The prompt for code generation is inherited from our previous study [2] (details in [1]). To retrieve relevant links from the Web, the suggested prompt is:

  "Search the Web to find links where I can get more information about the generated code present in this response."

- ***LinkSearcher***. This participant is activated when the user selects a code fragment within the IDE and triggers operational **Mode 2**. It constructs a structured prompt similar to the previous one, but copying the selected code from the IDE and specifying the programming language (details can be found at [1]).

LLM Response Processor. This component converts the unstructured LLM output, *i.e.,* automatically generated code and retrieved links, into structured data that can be used by the *Analysis Coordinator* component for both the similarity analysis and the license identification.

Snippet Extractor. This component parses the HTML content from retrieved links to extract code snippets using domain-specific strategies, due to the heterogeneity of web content structures. For instance, it pulls Stack Overflow code from nested `<pre>` and `<code>` tags, and accesses GitHub code via raw URLs. CODEGENLINK currently supports popular domains such as *GitHub, Stack Overflow, GeeksForGeeks*, with a design allowing easy extension of extraction rules.

Analysis Coordinator. This component pairs code from retrieved links with selected or generated code snippets, then runs similarity checks to filter out irrelevant links and keep only those with closely matching code. The current version of CODEGENLINK allows the user to filter out irrelevant links relying on textual similarity metrics and/or clone detection techniques. Specifically, the clone detector component interfaces with CCFINDERSW [13] to identify potential code clones between paired snippets. The textual similarity analyzer uses a vector-based approach: it tokenizes each code snippet, converts it into a vector, and computes cosine similarity between vectors.

License Identifier. This component aims at detecting the license to be used when reusing/redistributing a code snippet taken from the Web. To this aim, CODEGENLINK supports several detection strategies. For URLs pointing to GitHub repositories, the component first queries the GitHub API to obtain metadata, including license information. If no license is found in the metadata, the component searches the repository for a `LICENSE` (or similar names) file and, if found, analyzes it using an external license classification tool, *i.e.,* GOOGLE'S LICENSE CLASSIFIER [6], to identify the license. In other cases, the license is determined based on the policy of the source website. For instance, all code snippets on Stack Overflow are licensed under the CC BY-SA 4.0 license. To do this, CODEGENLINK maintains a list of domain/license pairs. For other links, CODEGENLINK scans the webpage's HTML to search for license keywords from a predefined SPDX-based list [10] and uses any matches to infer the code's license.

Results Viewer. Once all similarity and license data are collected, a filtering step is applied. Based on thresholds defined in the configuration, only code pairs exceeding a given *similarity threshold* are retained. This helps to reduce noise and ensures that only potentially relevant links are presented to the user. The results are aggregated and shown using a WebView, showing a table containing the URL(s) of the likely origin of the generated code snippet, with its related license information, if any. Last, the component is responsible for presenting the results inside the IDE.

A. CODEGENLINK *Limitations*

CODEGENLINK has limitations we acknowledge:

- It provides evidence of similarity between generated code and code on the Web, not a guarantee that the generated code originates from the provided URL. Also, it currently does not tell which one was generated by the AI.
- Its performance may depend on the Web search capability of the used LLM and, to a minor extent, on the used clone detector and licensing classifier.

III. CODEGENLINK IN ACTION

To use CODEGENLINK, first, we must install the extension in VS Code and set all the parameters needed to use it successfully. The installation procedure, detailed in the Repository README [1], requires the installation of Python dependencies and external tools (CCFINDERSW and GOOGLE LICENSE IDENTIFIER).

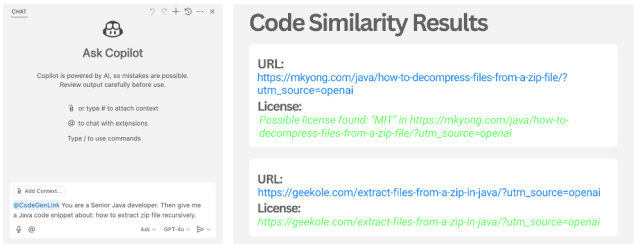

Fig. 2. Usage example of the @CodeGenLink participant

CODEGENLINK supports developers in their coding activities by providing two operational modes: (i) via Copilot Chat using the `@CodeGenLink` participant, or (ii) by selecting code directly within the IDE using the `@LinkSearcher` participant.

@CodeGenLink. As depicted in the left view in Fig. 2, a developer can ask Copilot to generate a code snippet for a specific task directly in its chat by prefixing the prompt with `@CodeGenLink`. The specification of the participant will trigger the CODEGENLINK extension. In this example, the developer asks to generate a code snippet to "extract zip file recursively". Once the LLM output is received, CODEGENLINK asks the LLM to search for relevant links related to the previously generated code and will process their contents to come up with a set of links that are likely the origin of the generated code. Depending on the similarity thresholds configured by the developer, the set of initial retrieved links is filtered out from noisy data, *e.g.,* links related to the coding tasks but without any reference to the automatically generated code snippets. Furthermore, for each link likely related to the generated code, CODEGENLINK will identify the license associated with the code snippet to let the developer be aware of how the code could be reused and/or redistributed. During this activity, there is no need for further interactions with the user while the process runs in the background.

The right view in Fig. 2 shows the visualization of the links likely being the origin for the generated code from CODEGENLINK via an HTML panel embedded in the IDE. Specifically, for each link surviving the filter, the panel reports the URL and the License (if any). Since there might be errors in the identification of the license, CODEGENLINK only provides a suggestion for developers, *i.e., Possible license found:"<<LICENSE>>"*.

@LinkSearcher. CODEGENLINK can also be used when the automatically generated code is created directly within the IDE, *e.g.,* using Copilot's code completion. Note that this operational mode can be used for any code snippet in the codebase. In this case, the developer manually selects a code snippet in the VS Code editor and activates the dedicated command, **Send Selected Code to CODEGENLINK**, through the extension interface.

Once activated, the extension invokes the `@LinkSearcher` participant, which automatically constructs a request (described in Section II) to the LLM to retrieve links to potential sources that contain code similar to the selected code. The model's response contains a list of links pointing to potentially relevant web pages. From this point onward, the pipeline proceeds as described in the previous operational mode. Also in this case, the likely origins of the code are shown to the developer through an embedded HTML view within the IDE.

TABLE I
CODEGENLINK PRECISION FOR VARYING THRESHOLDS (IN PARENTHESES, THE AVERAGE NUMBER OF RETURNED LINKS PER QUERY)

CODESEARCHNET				
	0.5	**0.6**	**0.7**	**0.8**
Cloning Ratio	0.88 (1)	0.88 (1)	0.94 (1)	1.00 (1)
Textual Similarity	0.58 (2)	0.59 (2)	0.70 (2)	0.81 (1)
CODEREVAL				
	0.5	**0.6**	**0.7**	**0.8**
Cloning Ratio	0.90 (1)	0.94 (1)	0.94 (1)	1.00 (1)
Textual Similarity	0.58 (2)	0.63 (2)	0.67 (1)	0.86 (1)

IV. PRELIMINARY EVALUATION

We have performed a preliminary evaluation of CODEGENLINK. Specifically, we evaluated the CODEGENLINK's precision in a scenario where, given a code snippet, we asked for the retrieval of relevant links. Note that we have not evaluated the recall since the complete set of true source links (ground truth) for automatically generated code is typically unknown, mostly because there are no existing benchmarks, *i.e.,* labeled dataset, of generated code snippets and their exact source links to compare against.

For the evaluation, we started with 101 coding tasks (46 Java and 55 Python) in our previous work [2], retrieved from the CODESEARCHNET dataset [8], for which at least one relevant link was associated with the automatically generated code. After that, we extended the evaluation, randomly selecting 100 coding tasks (50 for each programming language) from CODEREVAL [18], a benchmark curated from real-world projects mostly used in the literature to evaluate the effectiveness of LLMs for code generation [7].

For each task, we first asked Copilot to generate the code. After that, we used the automatically generated code to ask the Copilot chat to retrieve links related to it. To assess the tool's precision, two authors independently assessed each link in the sample and discussed disagreements.

Since CODEGENLINK provides only links that are greater than the user-determined thresholds for the maximum cloning ratio and textual similarity, Table I reports the precision of CODEGENLINK in the two datasets, varying the thresholds for the two metrics in the range [0.5-0.8], together with the average number of links shown to the users in parentheses. Although we have no objective means to assess CODEGENLINK's recall,

such numbers are consistent when considering the tasks from the CODEREVAL benchmark, where the tool provided 263 links, with only 14 (related to 11 different code snippets) being considered as "highly relevant" by the manual annotators. For the CODESEARCHNET tasks, the tool provided 278 links, of which only 29, belonging to 26 snippets, have been considered "highly relevant" to the automatically generated code. Looking at the precision of CODEGENLINK shown in Table I, it is possible to state that, in general, the tool has promising performance, *i.e.,* the precision increases when the thresholds increase, while the total number of visualized links decreases.

Finally, manual annotators also evaluated the tool's ability to identify the appropriate license when reusing a code snippet sourced from a web link. Among the 278 links from the CODESEARCHNET tasks, CODEGENLINK retrieved a license for 109 links, of which only six were wrongly assigned. For the CODEREVAL benchmark, instead, the manual annotators deemed 120 of the 131 retrieved licenses as correct. Incorrect detection is often due to the retrieval of a license contained in a web page that, while suitable for the web content, is not the actual source code license. Furthermore, the relatively low percentage of links to which it is possible to associate a license ($\simeq 44\%$) is because very often the retrieved links point to blogs/tutorials from which it is not easy to automatically determine the license. In such cases, it will be the responsibility of the developers to investigate the license and, if any, determine the redistribution criteria.

V. CONCLUSIONS AND FUTURE WORK

This paper describes CODEGENLINK, a GitHub Copilot extension that integrates with the VS Code editor. The tool can be used by developers during tasks where they seek assistance from LLMs to generate a code snippet and want to obtain information about the "trustworthiness" of the automatically generated code, as well as license information, helping them to assess how the generated code can be reused/redistributed within their code base.

A preliminary evaluation of CODEGENLINK highlights its promising performance. Indeed, even if the number of coding tasks from which the tool can come up with at least one possible link having a code snippet highly similar to the generated one is low, the filtering mechanism implemented with several similarity metrics helps filter out noisy data. Regarding license identification, the identified licenses are generally accurate, although, for several links—e.g., those pointing to some web pages—no clear license could be inferred.

Future work aims at (i) improving the code search capability of CODEGENLINK by combining multiple heuristics, (ii) identifying whether the code on the web was, itself, AI-generated, and (iii) performing an extensive empirical evaluation.

ACKNOWLEDGMENTS

This project has been financially supported by the European Union NEXTGenerationEU project and by the Italian Ministry of University and Research (MUR), a Research Project of Significant National Interest (PRIN) 2022 PNRR, project n. D53D23017310001 entitled 'Mining Software Repositories for enhanced Software Bills of Materials (MSR4SBOM)'. Daniele Bifolco is partially funded by the PNRR DM 118/2023 Italian Grant for Ph.D. scholarships.

REFERENCES

[1] D. Bifolco, G. Annicchiarico, P. Barbiero, M. Di Penta, and F. Zampetti, "CODEGENLINK." [Online]. Available: https://github.com/danielebifolco/CodeGenLink

[2] D. Bifolco, P. Cassieri, G. Scanniello, M. Di Penta, and F. Zampetti, "Do LLMs Provide Links to Code Similar to what they Generate? A Study with Gemini and Bing CoPilot," in *Proceedings of the 22nd ACM/IEEE International Conference on Software Repositories (MSR 2025), April 2025, Ottawa, ON, Canada*, 2025.

[3] G. Colombo, L. Mariani, D. Micucci, and O. Riganelli, "On the possibility of breaking copyleft licenses when reusing code generated by chatgpt," *arXiv preprint arXiv:2502.05023*, 2025.

[4] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," in *IEEE/ACM International Conference on Software Engineering: Future of Software Engineering, ICSE-FoSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 31–53.

[5] GitHub, "GitHub Copilot," Last Accessed: May 2025. [Online]. Available: https://copilot.github.com

[6] Google, "Google License Classifier," Last Accessed: May 2025. [Online]. Available: https://github.com/google/licenseclassifier

[7] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. C. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *CoRR*, vol. abs/2308.10620, 2023.

[8] H. Husain, H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *CoRR*, vol. abs/1909.09436, 2019.

[9] J. Katzy, R. M. Popescu, A. van Deursen, and M. Izadi, "An exploratory investigation into code license infringements in large language model training datasets," in *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering, FORGE 2024*. ACM, pp. 74–85.

[10] Linux Foundation, "Spdx: Software package data exchange," https://spdx.dev, 2025, accessed: 2025-05-25.

[11] V. Majdinasab, A. Nikanjam, and F. Khomh, "Trained without my consent: Detecting code inclusion in language models trained on code," *CoRR*, vol. abs/2402.09299, 2024.

[12] OpenAI, "Web search," Last Accessed: May 2025. [Online]. Available: https://platform.openai.com/docs/guides/tools-web-search?api-mode=responses

[13] Y. Semura, N. Yoshida, E. Choi, and K. Inoue, "CCFinderSW: clone detection tool with flexible multilingual tokenization," in *24th Asia-Pacific Software Engineering Conference, APSEC 2017, Nanjing, China, December 4-8, 2017*. IEEE Computer Society, 2017, pp. 654–659.

[14] R. Tufano, A. Mastropaolo, F. Pepe, O. Dabic, M. Di Penta, and G. Bavota, "Unveiling ChatGPT's usage in open source projects: A mining-based study," in *21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024*. ACM, 2024, pp. 571–583.

[15] Visual Studio Code, "Chat extensions," Last Accessed: May 2025. [Online]. Available: https://code.visualstudio.com/api/extension-guides/chat

[16] W. Xu, K. Gao, H. He, and M. Zhou, "LiCoEval: Evaluating LLMs on License Compliance in Code Generation," in *Proceedings of the 47th ACM/IEEE International Conference on Software Engineering (ICSE 2025), April 2025, Ottawa, ON, Canada*, 2025.

[17] Z. Yang, Z. Zhao, C. Wang, J. Shi, D. Kim, D. Han, and D. Lo, "Unveiling memorization in code models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[18] H. Yu, B. Shen, D. Ran, J. Zhang, Q. Zhang, Y. Ma, G. Liang, Y. Li, Q. Wang, and T. Xie, "Codereval: A benchmark of pragmatic code generation with generative pre-trained models," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024*. ACM, 2024, pp. 37:1–37:12.