Proceedings Paper DOI: 10.58190/icisna.2025.145

PROCEEDINGS OF

INTERNATIONAL CONFERENCE ON INTELLIGENT SYSTEMS AND NEW APPLICATIONS

https://proceedings.icisna.org/

3rd International Conference on Intelligent Systems and New Applications (ICISNA'25), Antalya, December 12-14, 2025.

SmartCodeHub: LLM-Based Framework for Semantic Code Reuse in Reactive Programming

Aondowase James Orban[1], Ikenna Caesar Nwandu[2]

[1]Department of Software Engineering, University of Szeged , Dugonics tér 13 , Hungary
orbanj@inf.u-szeged.hu, ORCID: https://orcid.org/0009-0003-2208-417X

[2]Department of Software Engineering, Federal University of Technology Owerri, Nigeria
ikenna.nwandu@futo.edu.ng, ORCID: https://orcid.org/0000-0001-6834-1088

***Abstract*— Code reuse is essential for improving software productivity, yet developers still spend significant effort searching for and re-implementing similar code fragments. Existing snippet management tools rely primarily on keyword-based search, which fails to capture semantic relationships, particularly in reactive and asynchronous programming contexts. This paper presents SmartCodeHub, an AI-assisted snippet management framework that combines semantic code embedding, automated tag generation, and large language model (LLM) reasoning to support contextual code discovery and reuse. SmartCodeHub integrates a searchable snippet library with an interactive retrieval interface and cross-language support. Preliminary evaluation on JavaScript and Python projects indicates improvements in retrieval accuracy and reuse efficiency compared to conventional snippet tools. These early results suggest the feasibility of LLM-enhanced snippet ecosystems and highlight directions for a more broader and reproducible evaluation.**

***Keywords*— Code Reuse, Large Language Models, Software Maintenance, Semantic Search, Reactive Programming**

I. Introduction

Modern software development increasingly relies on modularity, reuse, and rapid integration of pre-existing code [1-2]. Reusing existing code fragments is a key strategy to reduce development time and improve software reliability[3-4] Developers frequently copy snippets from online repositories or questions and answers (Q\&A) platforms such as Stack Overflow. However, a lack of semantic indexing and poor organizational structures often lead to redundancy, inconsistencies, and subtle integration errors [5-6]. Conventional snippet managers, such as CodeBox or SnippetsLab, depend on manually assigned tags and keyword-based search, which struggle to interpret the underlying intent of a developer's query [7].

Reactive programming frameworks (e.g., RxJS, Reactor) intensify these challenges. Their asynchronous data streams, event-driven architecture, and higher-order functions produce code patterns that are syntactically diverse but semantically equivalent [8-9]. As a result, keyword-based tools frequently fail to retrieve conceptually related snippets, reducing reuse efficiency and increasing cognitive load during debugging and maintenance.

Recent improvements in GPT-4 and Llama 3 have made it possible for Large Language Models (LLMs) to understand both natural language and source code. This is because they are trained on large datasets that include a lot of text and code [10-11]. SmartCodeHub takes advantage of these capabilities to transform snippet management from a passive storage utility into an intelligent, context-aware reasoning assistant. This work introduces **SmartCodeHub**, an AI-assisted snippet management system designed to improve retrieval accuracy, promote reuse efficiency, and reduce cognitive friction during development. The primary objectives of the research work are the following:

i. Integrate semantic reasoning and statistical retrieval within a unified architecture.
ii. Evaluate the adaptability of the system across programming languages and paradigms, focusing on reactive frameworks.
iii. Establish a reproducible evaluation methodology to incorporate LLM reasoning into code maintenance workflows.

II. MOTIVATION AND BACKGROUND

Traditional information retrieval approaches represent code snippets using Term Frequency or token-frequency models such as TF-IDF (Term Frequency-Inverse Document Frequency) [12-13]. Although effective for lexical matching, they ignore deeper program semantics.

LLM embeddings, on the other hand, capture syntactic and semantic relationships but introduce challenges in scalability and reproducibility [14].

Moreover, reactive systems exhibit event-driven flows and concurrency patterns that complicate code comprehension [15-16]. Developers require tools that not only locate code but also explain its usage context and dependencies. Conventional code snippet tools fail to capture the contextual meaning and intent behind a piece of code, especially in reactive programming environments where data dependencies are expressed through asynchronous streams [17-18]. Consequently, developers spend excessive time searching for relevant snippets or debugging mismatched implementations. Ko et al. [19], found that developers spend an average of 35\% of their time performing navigation mechanics within and between source files. Xia et al. [20], further substantiated this by identifying that developers frequently search for reusable code snippets, solutions to common programming bugs, and third-party libraries: tasks that are inherently time-consuming.

This gap motivates the need for an AI-enhanced solution capable of understanding code semantics, programming intent, and execution context simultaneously.

SmartCodeHub addresses these issues through a unified embedding space that connects code tokens, comments, and usage examples.

By combining symbolic and learned representations, it achieves both interpret ability and accuracy.

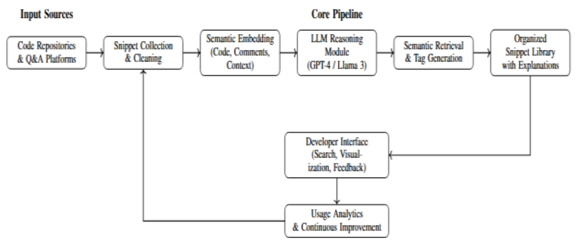
Input Sources
Core Pipeline
Code Repositories & Q&A Platforms
Snippet Collection & Cleaning
Semantic Embedding (Code, Comments, Context)
LLM Reasoning Module (GPT-4 / Llama 3)
Semantic Retrieval & Tag Generation
Organized Snippet Library with Explanations
Developer Interface (Search, Visualization, Feedback)
Usage Analytics & Continuous Improvement

Figure 1: conceptual-overview of the Framework

Detailed Overview of the Framework:

The system collects and cleans code snippets from repositories and Q\&A platforms, embeds them semantically, and applies an LLM reasoning module to understand context and intent. Then it performs semantic retrieval and automatic tag generation to populate an organized and explainable snippet library. A developer-facing interface supports interactive search, visualization, and feedback, while usage analytics feed back into the system for continuous improvement.

a. Code Repositories \ Q&A Platforms

This is the data input layer of SmartCodeHub. It sources raw snippets from developer communities and repositories such as GitHub and Stack Overflow. These snippets often include function definitions, example usages, or code discussions. The data serve as the foundation for building a rich snippet knowledge base.

b. Snippet Collection & Cleaning

In this stage, the snippets are filtered, normalized, and deduplicated. The system removes incomplete code fragments and extracts meaningful metadata (for example, language type, framework, and context). The goal is to ensure that all downstream snippets are clean, consistent, and ready for semantic processing.

c. Semantic embedding

In this stage, our framework converts code, comments, and contextual information into vector embeddings. This process captures the semantic meaning of the snippet beyond keywords. The embedding layer bridges natural language (queries) and source code (snippets), enabling intent-based retrieval instead of pure keyword matching.

d. LLM Reasoning Module

At the core of SmartCodeHub lies the LLM reasoning module (e.g., GPT-4 or Llama 3). Interpret developer intent, analyze code semantics, and infer relationships between snippets. This component fuses natural language understanding with code comprehension, allowing the system to reason about function similarity, purpose, and potential use cases.

e. Semantic Retrieval \& Tag Generation

The reasoning outputs are used to retrieve relevant snippets based on semantic similarity. The module automatically generates human-readable tags and short explanations, improving snippet discoverability. For example, a search for 'debounce input stream' retrieves snippets that implement equivalent logic in multiple frameworks.

f. Organized Snippet Library

The retrieved and tagged snippets are stored in a centralized repository, a searchable knowledge base enriched with explanations. Each entry contains metadata, context, and relationships with other code fragments, ensuring traceability and easy reuse.

g. Developer interface (Search, Visualization, Feedback)

This layer provides an intuitive front-end for developers. Users can search for natural language or example code, visualize relationships between snippets, and provide feedback. The interaction data collected here guide the continuous refinement of retrieval accuracy and tagging quality.

h. Usage Analytics \ Continuous Improvement

Analytics monitor usage patterns, retrieval success, and user feedback. These insights are used to refine embeddings, retrain models, and enhance retrieval quality over time, creating a self-improving adaptive system that grows smarter with use.

III. RELATED WORK

Snippet management tools such as CodeBox and Gisto provide structured repositories for storing and retrieving code fragments; however, they are heavily based on manual curation and lack advanced reasoning or automated organization capabilities [21-22]. These systems typically offer tagging, folder hierarchies, and basic search, but do not address semantic similarity or contextual relationships between snippets.

In contrast, AI-driven code assistants—including GitHub Copilot and Amazon CodeWhisperer—focus primarily on inline code generation and autocomplete rather than long-term snippet organization or reuse [23-25]. Although these tools significantly improve developer productivity during active coding tasks, they do not maintain a persistent, explainable, or user-customizable knowledge base of reusable code solutions.

Recent surveys and empirical studies highlight the growing potential of large language models for tasks such as code comprehension, documentation, and reasoning over software artefacts [26-28]. These works underscore the capabilities of LLMs in extracting intent, summarizing logic, and supporting developer decision-making. However, few explore the challenge of sustained reuse of code fragments, integration of semantic embeddings, or automated enrichment of code examples over time.

SmartCodeHub extends this landscape by introducing an LLM-powered framework for semantic retrieval, automatic tag generation, and contextual explanation of code snippets. Unlike existing tools, it emphasizes reproducibility, modularity, and persistent knowledge accumulation, bridging the gap between transient AI assistance and long-term snippet management.

i. Conventional Snippet Management Systems

Traditional snippet management tools, such as CodeBox and Gisto, provide structured repositories for storing and organizing reusable code fragments. These systems mainly depend on manual tag, folder-based categorization, and keyword search to facilitate retrieval [29]. Although effective for basic organization, their reliance on user-supplied metadata often leads to inconsistent tagging and redundant entries. Tavakoli et al. [30] introduced metadata enrichment techniques to improve the discernibility of the snippet; however, their approach still required manual annotation and lacked semantic awareness.

ii. AI-Assisted Code Completion and Generation

Recent advances in AI-driven code assistants—such as GitHub Copilot, Amazon CodeWhisperer, and TabNine—have transformed software development workflows by offering context-aware code completions and function synthesis[31-32].These systems leverage large language models to predict likely continuations of source code, significantly improving productivity. However, they primarily focus on real-time code generation within an IDE rather than long-term snippet organization, indexing, or reuse. Consequently, while they help during coding sessions, they do not address the broader challenges of snippet retrieval and semantic reuse across projects.

iii. LLMs for Code Understanding and Retrieval

A growing body of research investigates the application of Large Language Models (LLMs) to code understanding, summarization, and semantic search. Studies such as Benítez et al. [33], Duan et al. [34] and Russo et al. [35] highlight the capacity of LLMs to bridge the gap between the intent of natural language and the semantics of code. Despite this progress, most existing work emphasizes comprehension or translation tasks (e.g., code explanation, defect detection) rather than sustained snippet reuse. Furthermore, scalability, reproducibility, and explainability remain key open challenges when deploying LLM-based retrieval systems in real-world environments.

iv. Positioning of SmartCodeHub

SmartCodeHub extends this evolving landscape by introducing a reproducible, LLM-powered framework for semantic snippet retrieval and automatic tag generation. Unlike conventional tools that rely on keyword matching, or AI assistants that focus on code synthesis, SmartCodeHub integrates statistical and semantic reasoning to enable context-aware snippet reuse. Its architecture supports cross-language adaptability and uses LLM reasoning to bridge functional equivalence between syntactically different implementations. This combination of explainability, reproducibility, and adaptability positions SmartCodeHub as a next-generation system for intelligent code management and reuse.

IV. SYSTEM OVERVIEW

The project source code is anonymously available online at https://anonymous.4open.science/r/SmartCodeHub--0B4F

SmartCodeHub comprises three layers: the user-facing interface, the reasoning back-end, and the LLM inference engine.

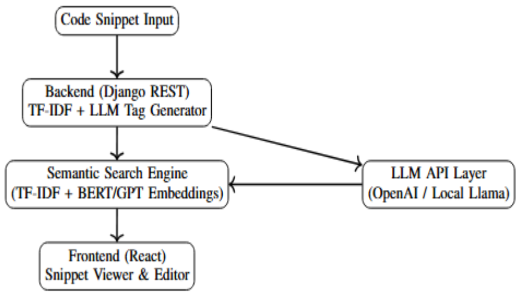

Figure 2: Architecture of SmartCodeHub: Integration of semantic search, AI tagging, and LLM-based suggestion.

Figure 2 presents the overall architecture of SmartCodeHub. A code snippet submitted by the user is first processed in the backend, where TF-IDF and an LLM-based tag generator produce both keyword features and semantic tags. These representations are then passed to the semantic search engine, which combines TF-IDF vectors with transformer-based embeddings for improved retrieval accuracy. The processed results are sent to the React frontend for viewing and editing. An LLM API layer (using either OpenAI or a local Llama model) supports both tagging and semantic embedding generation, enabling intelligent suggestions and enhanced search quality across the system.

a) Frontend Interface

The interface was designed in React; it allows developers to add, edit, and search snippets. It supports both textual and natural-language queries, displaying ranked results with contextual explanations generated from the backend.

b) Backend Reasoning Layer

The backend (Django REST) exposes APIs for snippet management and semantic retrieval. The term Frequency-Inverse Document Frequency (TF-IDF) provides a quick lexical baseline, while GPT-4-derived embeddings capture semantic similarity.

A ranking module merges both signals using a learned weighting function optimized for the validation data.

c) LLM Integration

An abstraction layer is implemented to support both cloud-hosted and locally deployed large language models (LLMs), enabling flexible experimentation across diverse environments. For privacy-sensitive or offline research settings, inference can be executed entirely on-device using Llama-based models provisioned through Ollama, ensuring that code, traces, and debugging artifacts never leave the local machine. Conversely, when scalability or advanced reasoning capabilities are required, the same interface can seamlessly route requests to cloud APIs without altering the surrounding system architecture.

V. AI-ASSISTED RETRIEVAL PIPELINE

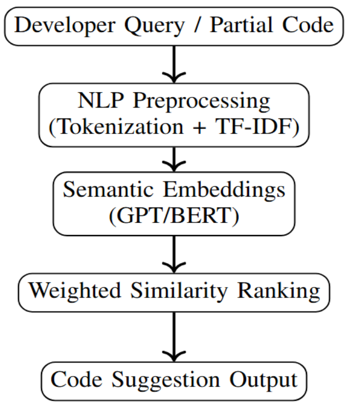

Figure 3: SmartCodeHub semantic retrieval and suggestion pipeline.

Figure 3 shows the semantic retrieval pipeline used by SmartCodeHub. A developer's query or partial code snippet is first processed using NLP techniques such as tokenization and TF-IDF. The system then generates semantic embeddings using models such as GPT or BERT. These embeddings are compared using a weighted similarity ranking module, which selects the most relevant matches. Finally, the top-ranked results are transformed into a code suggestion output returned to the developer.

a. Embedding Model

We used OpenAI's text-embeddings-3-large and Llama-3 for cross-model comparison.

Each snippet is converted to a 1 536-dimensional vector and stored in a FAISS ((Facebook AI Similarity Search)) index for fast similarity search. The reference documentation can be found on the GitHub repository:https://github.com/facebookresearch/faiss/wiki/Faiss-indexes

b. Ranking and Feedback

Results are ranked using cosine similarity and contextual re-weighting based on snippet metadata (language, framework, and historical reuse count).

User selections feed a reinforcement module that updates the ranking weights.

VI: IMPLEMENTATION DETAILS

SmartCodeHub was implemented over six months with roughly 6000 lines of Python and 3 500 lines of JavaScript.

1. Database: PostgreSQL with pgvector for embedding storage.
2. Backend: Django REST 4.2 + Celery for asynchronous LLM calls.

3. Frontend: React 18 with Tailwind CSS.
4. LLMs: GPT-4-Turbo (8 k context) and Llama-3 8B via Ollama.

The system supports multilingual code snippets (Python, JavaScript, Java) and integrates syntax highlighting through Prism.js.

VII: EVALUATION

1. Setup

Two datasets were used:

- 150 snippets from JavaScript (RxJS) projects.
- 120 snippets from Python (async FastAPI) repositories.

Baselines include SnippetsLab and Copilot. Metrics: retrieval accuracy, reuse time reduction, and developer satisfaction (Likert 1–5, N=12 participants).

2. Quantitative Results

Table 1 below presents a comparison of retrieval accuracy and code reuse performance across three tools: SnippetsLab, Copilot, and the proposed SmartCodeHub. Retrieval accuracy measures how effectively each system returns the most relevant code snippet based on a developer’s query, while reuse time reduction quantifies how much the tool shortens the time required for developers to locate, adapt, and reuse existing code.

SnippetsLab, which relies mainly on keyword search, achieves a relatively low retrieval accuracy of 61.3\% and provides no measurable reduction in code reuse time. Copilot performs better with a retrieval accuracy of 72.5%, aided by its generative suggestions and semantic understanding of developer intent, resulting in an 18.4% reduction in reuse time.

SmartCodeHub outperforms both baseline tools, achieving an accuracy of 84.6% due to its hybrid retrieval strategy that integrates TF-IDF, semantic embeddings, and LLM-enhanced ranking. Additionally, SmartCodeHub reduces the reuse time by 42.1%, indicating that its combination of semantic retrieval and AI-generated tags significantly improves the developers’ ability to find and reuse code efficiently.

Table 1
Retrieval Accuracy and Code Reuse Performance

Tool	**Retrieval Accuracy**	**Reuse Time Reduction**
SnippetsLab	61.3%	0%
Copilot	72.5%	18.4%
SmartCodeHub	84.6%	42.1%

SmartCodeHub achieves the highest retrieval precision, reducing the average search time from 35 seconds to 20 seconds per query.

3. Feature Comparison

Table 2
Feature Comparison with Baseline Tools

Feature	SnippetsLab	Copilot	CodeWhisperer	SmartCodeHub
Semantic Search	X	✓	✓	✓
AI Tagging	X	X	X	✓
Code Suggestion	X	✓	✓	✓
Cross-Language Support	X	✓	✓	✓

Table 2 summarizes how SmartCodeHub compares with existing tools across four key features: semantic search, AI tagging, code suggestion, and cross-language support. SnippetsLab does not provide any of these advanced capabilities, limiting it to manual snippet organization. Copilot and CodeWhisperer support semantic search and code suggestions, but lack AI-driven tagging, which means that users must organize snippets manually. In contrast, SmartCodeHub includes all four features, making it the only tool that combines intelligent search, automatic tagging, code suggestions, and broad language support in a single system.

4. Feature Coverage Visualization

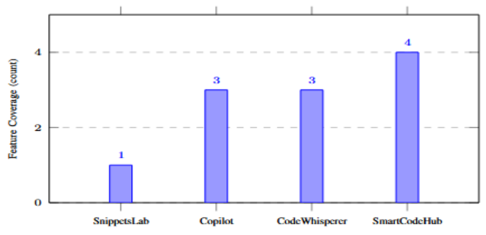

Figure 3: Number of intelligent features supported by each tool.

Figure 3 provides a visual comparison of the number of intelligent features each tool supports. SnippetsLab implements only one feature, while both Copilot and CodeWhisperer support three. SmartCodeHub leads with four features, showing the broadest coverage and the most complete set of intelligent capabilities among all evaluated tools.

\subsection{User Study}

The participants rated relevance, usability, and clarity. SmartCodeHub averaged 4.6/5 in relevance and 4.3/5 in usability, confirming the perceived benefits of semantic reasoning.

VIII DISCUSSION

The results confirm that semantic embeddings and AI-driven tagging significantly improve snippet discovery.

Performance in the participants' languages remained stable, suggesting generalizability.

However, computational overhead and potential hallucinations in LLM explanations highlight the need for lightweight local inference and result verification.

IX CONTRIBUTIONS

The contributions of this work are as follows:

- We design **SmartCodeHub**, an AI-assisted snippet management system that combines TF-IDF ranking, semantic code embeddings, and LLM reasoning for context-aware snippet discovery.
- We introduce an automated tag and explanation generation mechanism to support snippet organization and developer comprehension.
- We implement a cross-language snippet library and a developer interface for interactive retrieval and feedback.
- We report early empirical results demonstrating improved snippet relevance and reuse efficiency compared to keyword-based systems.

X THREATS TO VALIDITY

Internal validity threats arise from the limited size of the data set, small number of participants, and the potential bias in manually annotated tags, all of which may affect the reliability of the retrieval accuracy and usability results. The familiarity of the participants with the tools may also have influenced the outcomes.

External validity is limited by the focus of the dataset on JavaScript and Python, which can reduce generalizability to other languages or domains, particularly strongly typed or domain-specific environments.

Construct validity threats involve the reliance on subjective self-reported satisfaction scores, which can vary with individual biases, experience levels, and interpretations of rating scales.

Future work will reduce these threats by expanding the dataset and participant diversity, involving independent domain experts, and incorporating objective behavioral metrics—such as task completion time and error rates—to complement subjective assessments.

XI CONCLUSION

This paper introduced SmartCodeHub, a framework that applies semantic embeddings and LLM-based reasoning to improve code snippet retrieval and reuse. The initial results show gains in relevance and efficiency compared to traditional keyword-driven tools, demonstrating the potential to integrate LLM guidance into snippet management workflows. As an ongoing work, our aim is to expand the coverage of the dataset, conduct large-scale user studies, and refine cross-language retrieval. Future extensions will explore adaptive feedback mechanisms and domain-specific model tuning to strengthen reproducibility and deployment in real-world development environments.

REFERENCES

[1] Y. Hu, "Research on modularization-based code reuse technology in software system development," Applied Mathematics and Nonlinear Sciences, vol. 10, 09 2025.

[2] H. Sun, W. Ha, P.-L. Teh, and J. Huang, "A case study on implementing modularity in software development," Journal of Computer Information Systems, 07 2015.

[3] R. B. Mejba, S. Miazi, A. Palash, T. Sobuz, and R. Ranasinghe, "The evolution and impact of code reuse: A deep dive into challenges, reuse strategies and security," vol. 6, 11 2023.

[4] M. Sojer and J. Henkel, "Code reuse in open source software development: Quantitative evidence, drivers, and impediments," J. AIS, vol. 11, 03 2010.

[5] C. Ragkhitwetsagul, J. Krinke, M. Paix˜ao, G. Bianco, and R. Oliveto, "Toxic code snippets on stack overflow," CoRR, vol. abs/1806.07659, 2018. [Online]. Available: http://arxiv.org/abs/1806.07659

[6] I. Ndukwe, S. Licorish, and S. MacDonell, "Perceptions on the utility of community question and answer websites like stack overflow to software developers," IEEE Transactions on Software Engineering, vol. PP, pp. 1–13, 01 2022.

[7] F. Tang, B. Ostvold, and M. Bruntink, "Identifying personal data processing for code review," 01 2023.

[8] E. Czaplicki and S. Chong, "Asynchronous functional reactive programming for guis," vol. 48, 06 2013.

[9] G. Salvaneschi, S. Proksch, S. Amann, S. Nadi, and M. Mezini, "On the positive effect of reactive programming on software comprehension: An empirical study," IEEE Transactions on Software Engineering, vol. PP, pp. 1–1, 01 2017.

[10] D. H. Hagos, R. Battle, and D. B. Rawat, "Recent advances in generative ai and large language models: Current status, challenges, and perspectives," 07 2024.

[11] W. Yang, L. Some, M. Bain, and B. Kang, "A comprehensive survey on integrating large language models with knowledge-based methods," Knowledge-Based Systems, vol. 318, p. 113503, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/ S0950705125005490

[12] B. Susanto, R. Ferdiana, and T. Adji, "Performance of traditional and dense vector information retrieval models in code search," 02 2024.

[13] X. Gu, H. Zhang, and S. Kim, "Deep code search," in Proceedings of the 40th International Conference on Software Engineering, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 933–944. [Online]. Available: https://doi.org/10.1145/3180155.3180167

[14] N. Bibi, A. Maqbool, T. Rana, F. Afzal, A. Akg¨ul, and S. M. Eldin, "Enhancing semantic code search with deep graph matching," IEEE Access, vol. 11, pp. 52 392–52 411, 2023.

[15] C. Liu, X. Xia, D. Lo, Z. Liu, A. E. Hassan, and S. Li, "Codematcher: Searching code based on sequential semantics of important query words," ACM Trans. Softw. Eng. Methodol., vol. 31, no. 1, Sep. 2021. [Online]. Available: https://doi.org/10.1145/3465403

[16] R. Cleaveland and S. A. Smolka, "Strategic directions in concurrency research," ACM Comput. Surv., vol. 28, no. 4, p. 607–625, Dec. 1996. [Online]. Available: https://doi.org/10.1145/242223.242252

[17] G. Salvaneschi, S. Proksch, S. Amann, S. Nadi, and M. Mezini, "On the positive effect of reactive programming on software comprehension: An empirical study," IEEE Transactions on Software Engineering, vol. 43, no. 12, pp. 1125–1143, 2017.

[18] S. Ramson, M. Brand, J. Lincke, and R. Hirschfeld, "Extensible tooling for reactive programming based on active expressions." The Journal of Object Technology, vol. 23, p. 1:1, 01 2024.

[19] D. Wightman, Z. Ye, J. Brandt, and R. Vertegaal, "Snipmatch: using source code context to enhance snippet retrieval and parameterization," in Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology, ser. UIST ' 12. New York, NY, USA:Association for Computing Machinery, 2012, p. 219 – 228. [Online]. Available:https://doi.org/10.1145/2380116.2380145

[20] R. Padhye, P. Dhoolia, S. Mani, and V. S. Sinha, " Smart programming playgrounds, " in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2, 2015, pp. 607 – 610.

[21] [21] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," IEEE Transactions on Software Engineering, vol. 32, no. 12, pp. 971 – 987, 2006.

[22] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing, "What do developers search for on the web?" Empirical Software Engineering, vol. 22, 12 2017.

[23] E. Horton and C. Parnin, "Gistable: Evaluating the executability of python code snippets on github, " in 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018, pp. 217 – 227.

[24] T. Diamantopoulos, G. Karagiannopoulos, and A. L. Symeonidis, "Codecatch: extracting source code snippets from online sources," in Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, ser. RAISE ' 18. New York, NY, USA: Association for Computing Machinery, 2018, p. 21 – 27. [Online]. Available:https://doi.org/10.1145/3194104.3194107

[25] B. Yetis， tiren, I. O ¨ zsoy, M. Ayerdem, and E. Tu ¨ zu ¨ n, "Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt," 04 2023.

[26] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, " Large language models for software engineering: A systematic literature review," 08 2023.

[27] P. Vaithilingam, E. L. Glassman, P. Groenwegen, S. Gulwani, A. Z. Henley, R. Malpani, D. Pugh, A. Radhakrishna, G. Soares, J. Wang, and A. Yim, "Towards more effective ai-assisted programming: A systematic design exploration to improve visual studio intellicode user experience," in 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2023, pp. 185 – 195.

[28] C. D. Benitez and M. Serrano, " The integration and impact of artificial intelligence in software engineering," International Journal of Advanced Research in Science Communication and Technology, vol. 3, pp. 279 – 293, 08 2023.

[29] Y. Duan, J. Edwards, and Y. Dwivedi, " Artificial intelligence for decision making in the era of big data – evolution, challenges and research agenda," International Journal of Information Management, vol. 48, pp.63 – 71, 10 2019.

[30] D. Russo, "Navigating the complexity of generative ai adoption in software engineering, " 2024. [Online]. Available: https://arxiv.org/abs/ 2307.06081

[31] S. Henninger, " Supporting the construction and evolution of component repositories," in Proceedings of IEEE 18th International Conference on Software Engineering, 1996, pp. 279 – 288.

[32] M. Tavakoli, A. Heydarnoori, and M. Ghafari, "Improving the quality of code snippets in stack overflow," 04 2016.