SoftwareX 36 (2026) 103127

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

MediaWiki Code2Code Search: Neural retrieval for the semantic discovery of open-source software entities

Francesco Tosoni

Sant'Anna School of Advanced Studies, p.zza Martiri della Libertà 33, 56127 Pisa, PI, Italy

ARTICLE INFO

Keywords:
Neural information retrieval
Semantic code search
MediaWiki
Software Heritage
Vector embeddings
Large language models

ABSTRACT

Code search in large-scale ecosystems is often hindered by the lexical gap between user queries and implementation details, alongside the trade-off between the low latency of traditional Information Retrieval (IR) and the precision of Deep Learning (DL). We present MediaWiki Code2Code Search, a neural retrieval system for semantic code-to-code discovery. By indexing 1.29 million structural entities (functions, types, and templates) across 2500+ MediaWiki repositories, our system enables retrieval based on computational intent rather than surface tokens. We employ a split-build architecture, decoupling GPU-intensive offline indexing from a CPU-only serving layer; our FAISS IVF-PQ index occupies 168.6 MiB: a 96.6% reduction compared to a flat `float32` baseline, and achieves a median query latency of 1.85 s on commodity hardware, satisfying the 6 GiB RAM constraint of Wikimedia Toolforge. In a blinded evaluation graded by a three-model LLM jury (inter-judge Krippendorff's $\alpha = 0.74$), our system improves ranking quality over a BM25 baseline across a 27-query benchmark, raising the jury-mean nDCG@10 (to 0.81 from 0.63), a positive and consistent gain across all judges. Gains are most pronounced in name-obfuscated and cross-language tasks where lexical methods fail. The system is available at code2codesearch.toolforge.org under the Apache 2.0 licence and provides an open RESTful API.

Current code version

Code metadata. The tool is registered on Wikidata as item Q139251277.

Nr.	Code metadata description	Value
C1	Current code version	v2.1.0
C2	Permanent GitHub link to code/repository	https://github.com/ftosoni/mediawiki-code2code-search
C3	Legal Code License	Apache Software License 2.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python, JavaScript, SQLite, FAISS, Qwen3, Software Heritage
C6	Compilation requirements & dependencies	Python 3.11, Node.js (for UI), 6 GiB RAM (min)
C7	Link to developer documentation/manual	https://www.mediawiki.org/wiki/Code2Code_Search
C8	Support email for questions	Francesco.Tosoni@santannapisa.it

1. Motivation and significance

Wikipedia and its sister projects, attracting over 1.5 billion unique monthly visitors, are socio-technical systems underpinned by collaborative communities, open content, and open-source infrastructure [1]. Software development has been central to the Wikimedia community since its inception. The MediaWiki ecosystem (the technical backbone of most wiki projects) has evolved over two decades since 2002 [2, §1]. Continuous maintenance and expansion have transformed it from a monolithic application into a highly decentralised ecosystem spanning over 2500 code repositories across various forges (e.g., Gerrit, GitHub, GitLab). This decentralisation, while mirroring the community's diversity, complicates software discovery, maintenance, and consistent data modelling. The resulting shared codebase includes toolkits for processing Wikipedia [3] and Wikidata [4], bots for database analysis [5],

Email address: Francesco.Tosoni@santannapisa.it.

https://doi.org/10.1016/j.softx.2026.103127
Received 5 June 2026; Received in revised form 14 September 2026; Accepted 7 October 2026
Available online 13 October 2026
2352-7110/© 2026 Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

visualisation tools [6], and the core infrastructure itself: MediaWiki [2] and Wikibase [7].

For maintainers, this code abundance often leads to redundant or semantically similar features. Efficient code search (retrieving relevant snippets via natural-language or code queries) is hence essential for modern development [8,9], facilitating code completion, synthesis, traceability, and vulnerability identification.

Since 2019, the community has utilised a Codesearch tool [10] powered by Hound [11]. This tool leverages traditional information retrieval (IR) techniques, such as regular expressions and pattern matching, to index the origin/main branches of specified repositories without requiring local clones. Despite its scalability and the "2019 Coolest Tool Award," the system is limited by its reliance on surface-level terminological matching. For instance, a search for "permission validation logic" may fail if the target function is named `checkUserLoginStatus`. This lexical gap, caused by two decades of evolving and inconsistent naming conventions, hinders volunteer onboarding and reduces efficiency for core developers.

Semantic search using deep learning (DL) models [12–17] offers a robust alternative that balances accuracy and scalability. Consequently, the Wikimedia Foundation (WMF) and its affiliates have begun integrating AI-enabled retrieval into major projects, including Wikidata [18], Wikipedia [19], and Wikimedia Commons [20]. Notably, the Wikidata Embedding Project [18] uses graph embeddings for about 30 million nodes, with a two-stage scheme deploying Jina AI models and a DataStax vector database. While advanced, this reliance on proprietary models and storage contrasts with Wikimedia's commitment to free, open-source software (see, e.g., the Toolforge licensing policies[1]). Other recent initiatives include the WMF's hybrid search models (piloted in March 2026) [19] and the WISE multimodal image search engine [20,21], which provides semantic search for a curated subset of Wikimedia Commons images.

Our proposal. Despite these advancements, the underlying codebase governing Wikimedia projects still lacks a scalable, efficient code-search implementation. To address this, we present a code-to-code retrieval system that uses code embeddings to index MediaWiki-related repositories. Our primary contributions are threefold: (i) we transform snippets into vector representations, enabling intent-based nearest-neighbour retrieval by Euclidean (L2) distance over L2-normalised embeddings; (ii) we have archived the 2500+ MediaWiki repositories in Software Heritage (SWH) [22, §2.1][23–25], ensuring a unified data model across forges and utilising citable, ISO/IEC-compliant SoftWare Hash IDs (SWHIDs) [26,27]; and (iii) we provide a modern GUI built on Codex[2] localised in 30 languages at the time of writing (14 of them are Indian) and continuously expanded by volunteer translators through the translatewiki.net platform, improving accessibility and lowering barriers for non-English-speaking contributors.

Our architecture follows a build-heavy, serve-light approach: we use GPU-accelerated infrastructure to project code snippets into embeddings; conversely, the deployment environment remains resource-constrained and CPU-only. Unlike existing initiatives that rely on proprietary components, our entire system, including the chosen Qwen3 model, is released under the Apache Software License 2.0. The system is currently deployed on Toolforge (https://code2codesearch.toolforge.org/). It is accessible via both the Codex frontend (illustrated in Fig. 1) and a RESTful API with documentation served via an auto-generated Swagger UI at the `/docs` endpoint.

[1] https://wikitech.wikimedia.org/wiki/Special:PermaLink/2239062.

[2] https://doc.wikimedia.org/codex/main/.

2. Software description

The implementation of the Code2Code Search pipeline is structured into discrete modules to support maintainability, CI/CD workflows, unit testing, data integrity, and resource efficiency.

A broad range of pre-trained code models has been proposed for code understanding and retrieval, and empirical comparisons show that no single model dominates across tasks [28]. The system adopts `Qwen/Qwen3-Embedding-0.6B` [29] for its OSI-compliant licence and because a recent dedicated benchmark of deep-learning models for large-scale code-to-code recall [30] indicates it offers a compelling trade-off between inference efficiency and code-search precision across multiple languages, making it viable for a resource-constrained environment.

2.1. Software architecture

The system employs an asymmetric (build-heavy, serve-light) architecture, as illustrated in Fig. 2. Python dependencies are managed via the `requirements.txt` file at the project root.

The pipeline comprises the following stages:

1. **Repository Discovery:** We adopt the repository enumeration strategy of the legacy Codesearch tool. The script `list_repos.py` queries the `/api/v1/repos` endpoint across predefined categories (e.g., `search`, `extensions`, `pywikibot`). We identified 2568 repositories, primarily hosted on `gerrit.wikimedia.org` (1372), `gitlab.wikimedia.org` (826), and `github.com` (342). Canonical `gerrit` URLs are prioritised by replacing read-only `gerrit-replica` addresses.
2. **Cloning:** Using `download_repos.py`, we perform shallow clones (`--depth 1`) of the main branch for each repository. Consistent with existing infrastructure, Git submodules are excluded. Approximately 65 repositories are omitted due to duplication, authentication requirements, or download timeouts.
3. **Archival:** To ensure persistence, `archive_to_swh.py` submits all repositories to Software Heritage (SWH) in a single bulk request. When elevated privileges are unavailable, we provide `archive_individual_to_swh.py`, which supports sequential archiving and checkpointing for resumable execution.
4. **Structural Extraction:** `extract_structural_entities.py` uses Tree-sitter [31–33] to parse source files and extract functions, types, and templates across twelve languages. We note that 261 cloned repositories yielded no valid structural entities (as they contain only documentation, translation files, or assets in non-target languages), resulting in a final index of 2242 active repositories.
5. **Hash computation:** Subsequently, `resolve_swh_hashes_local.py` generates content-based identifiers, including `sha1` and SWHIDs, enriched with metadata (origin, filepath, lines). Using `ThreadPoolExecutor` from `concurrent.futures`, we generate a `raw_snippets.json` manifest containing all structural and repository-level metadata.
6. **Vectorisation:** `generate_embeddings.py` executes on a GPU-enabled machine. It loads the `Qwen3-Embedding-0.6B` model via `sentence-transformers` [34] to encode code snippets into normalised 1024-dimensional vectors. This process, which takes approximately 31 min on a single NVIDIA H200, produces `embeddings.npy`.
7. **Index Construction:** `build_index.py` converts `embeddings.npy` into a compressed FAISS [35] `IndexIVFPQ` index. Using product quantisation [36] with 128 sub-quantisers (8 bits/sub-vector), we significantly reduce the memory footprint to ensure compatibility with Toolforge's constraints.
8. **Metadata Migration:** `migrate_to_sqlite.py` transforms `raw_snippets.json` into an SQLite database (`snippets.db`). By

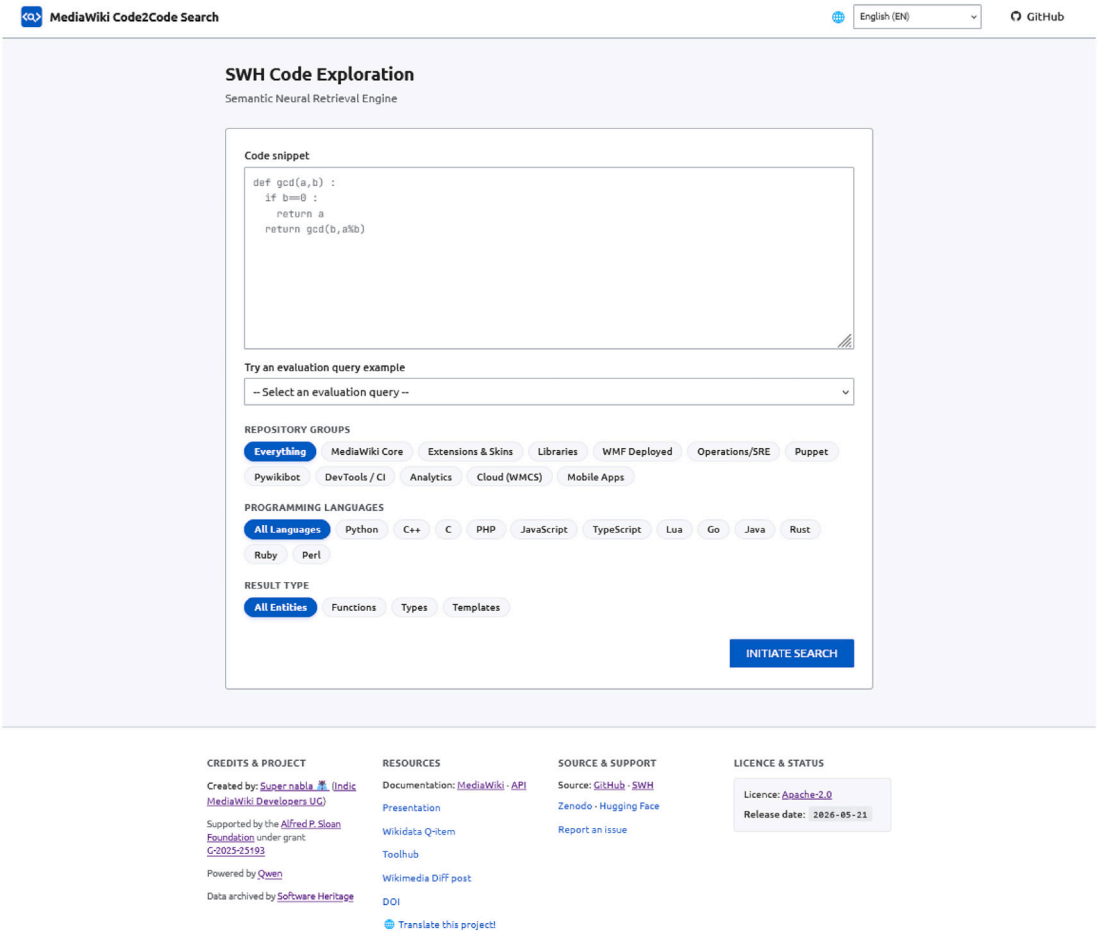

Fig. 1. The Codex-based frontend interface.

decoupling metadata storage from the FAISS index, this step runs independently and stays well within the 6 GiB RAM limit.

Table 1 summarises the indexed corpus. The pre-computed artefacts produced by this offline phase (`embeddings.npy`, the FAISS index, and `snippets.db`) are archived on Zenodo (10.5281/zenodo.20586256) and mirrored as a browsable dataset on Hugging Face (https://huggingface.co/datasets/ftosoni/mediawiki-code2code-search), so the serving layer can be reproduced without rerunning the GPU-bound stages. Following the offline build phase, the lightweight serving layer is deployed via a FastAPI backend, providing both a web interface and a RESTful API.

Latency. We measured end-to-end query latency by issuing 27 benchmark queries (see Section 3) to a local API instance and repeating the full run 7 times. We employed an ASUSTeK VivoBook X1605VA laptop (Windows 11), equipped with a 13th-generation Intel® Core™ i9-13900H processor (14 cores / 20 threads, up to 5.4 GHz boost), 32 GB of DDR4-SDRAM, and an integrated Intel® Iris® Xe Graphics GPU (128 MB dedicated VRAM). No discrete GPU was used.

Fig. 3 shows the per-query latency distributions. Averaged over the 27 queries, the per-query median latency is 1.85 s (mean 1.87 s). Latency correlates with query snippet length, ranging from 0.9 s for the shortest query (A1, a 4-line GCD) to 3.3 s for the longest (C3, a 19-line fan-out). On Toolforge (the live deployment, CPU-only shared virtual machines, 6 GiB RAM limit), end-to-end latency is substantially higher, averaging ≈31 s, ranging from 11 s for the shortest query (A1) to 111 s for the slowest (B5), because the Qwen3-Embedding-0.6B model runs on throttled shared CPU resources. GPU nodes are not currently available, but GPU deployment is the subject of an ongoing infrastructure request to the WMF to bring back inference to the order of seconds.

2.2. Software functionalities

Syntactic entity extraction. Using Tree-sitter, the system performs structural analysis across 12 languages (PHP, Python, C++, C, JavaScript, TypeScript, Lua, Go, Java, Rust, Ruby, and Perl). The extraction logic in `extract_structural_entities.py` employs a recursive scope-resolution algorithm to capture qualified identifiers (e.g., `Namespace\Class::Method`). The frontend includes multi-select filters (by repository group, language, and entity type) to refine search results.

Single-stage neural retrieval. User queries are encoded at inference time using the `Qwen3-Embedding-0.6B` model, following the bi-encoder (dual-encoder) retrieval paradigm, in which queries and code are embedded independently into a shared space and compared using vector similarity [34,37]. Corpus embeddings are stored in a FAISS `IndexIVFPQ` index, configured with 128 sub-quantisers (8 bits per sub-vector), which trades exact search for the recall/latency profile characteristic of large-scale approximate vector similarity search [38–40]. This quantisation reduces the memory footprint by 96.6% compared to a flat `float32` baseline (168.6 MiB versus approximately 4.9 GiB), ensuring the serving layer operates comfortably within Toolforge's 6 GiB RAM limit: an instance of the model and index-efficiency techniques increasingly required to deploy deep models under tight resource budgets [41].

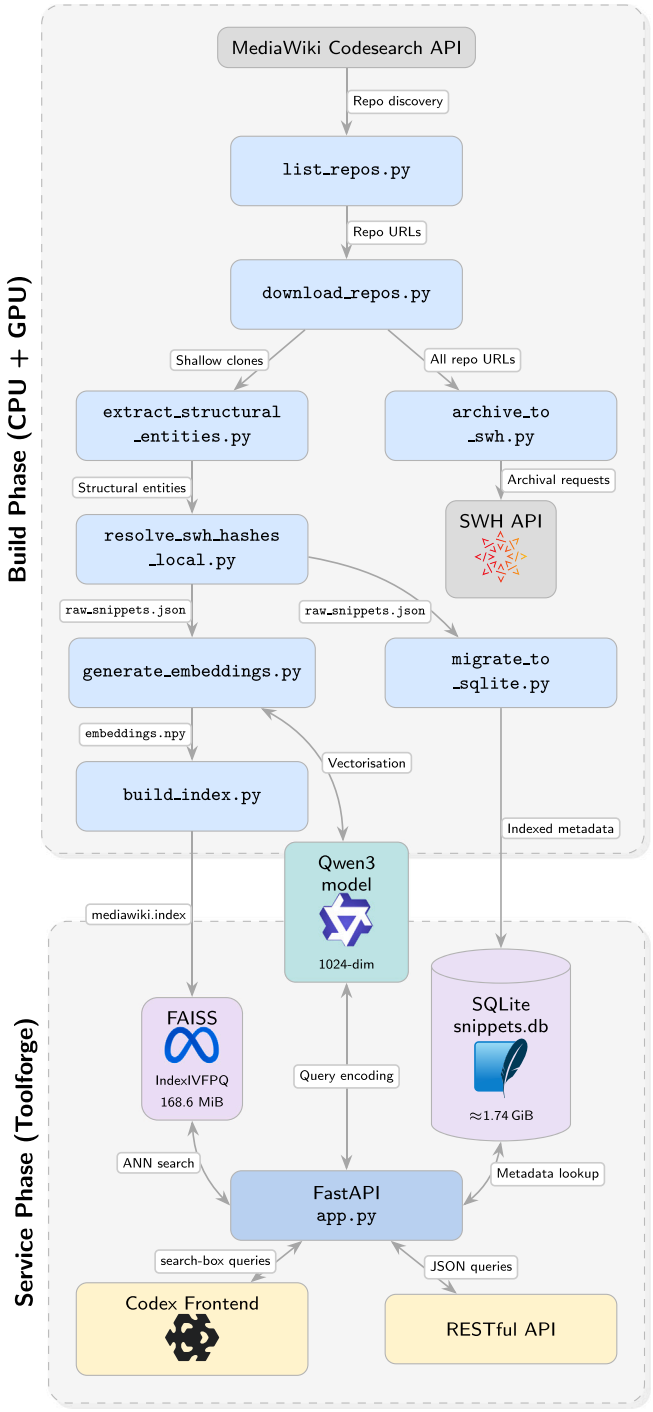

Fig. 2. System architecture of MediaWiki Code2Code Search. Node colours: build scripts; external services; data artefacts; API backend; ML model; frontend. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

2.3. Software verification and technical integrity

System robustness and semantic accuracy are validated by a suite of 34 `pytest` unit tests. Software provenance and citability are maintained via `codemeta.json` [42] and `CITATION.cff` files, which are synchronised with the release SWHID and commit hash using `update_release.py`. Further, the GitHub Actions CI pipeline (`.github/workflows/python-ci.yml`) enforces PEP 8 compliance via `flake8` and validates the parser logic on every commit; it also enables continuous deployment, as each push to the main branch triggers an automatic redeployment of the live Toolforge service: this occurs regularly, since translatewiki.net merges new and updated translations directly into the main branch.

Table 1
Indexed corpus statistics (snippets.db, May 2026).

Category	Count	Share
Functions	1,050,748	81.5%
Types (classes, structs, interfaces, enums)	237,653	18.4%
C++ Templates	1051	0.1%
Total indexed entities	**1,289,452**	
Top file extensions		
JavaScript (.js)	464,791	36.0%
PHP (.php)	312,369	24.2%
Go (.go)	305,591	23.7%
Python (.py)	112,946	8.8%
TypeScript (.ts)	36,362	2.8%
Other	57,393	4.5%

2.4. Portability and reuse

Software Heritage's content-addressed data model [23] serves as a foundational enabler, allowing repositories from disparate forges to be mapped to a unified SWHID namespace. This architecture mitigates the challenges of multi-forge diversity and demonstrates scalability across varied software ecosystems.

The system is highly modular: only the repository discovery script, `list_repos.py`, is ecosystem-specific. Retargeting the system for different forges, package registries, or domains (such as research institutions or SMEs) requires replacing only this single component. All downstream stages (Tree-sitter parsing, SWHID resolution, neural vectorisation, FAISS indexing, SQLite migration, and the FastAPI serving layer) are domain-agnostic. They depend solely on local source-file clones and the generated `raw_snippets.json` manifest.

3. Illustrative examples

The frontend exposes 27 queries as ready-to-run examples in a dropdown selector (Table 3); designed with increasing difficulty, they offer a readily available sample of the codebase's functionality.

No formal ground truth exists for code search in the MediaWiki ecosystem, unlike general-purpose benchmarks such as CodeSearchNet [43] and xCodeEval [44]; nor is a direct comparison with the Hound-based Codesearch tool meaningful, given fundamentally different query interfaces (regular expressions versus code snippets).

Methodology

Each of the 27 queries is a self-contained code snippet submitted verbatim to the API. For every query, we pool both systems' top-10 results (standard pooling practice [45]), deduplicate identical snippets *within* and *across* systems by a content hash (one shared label per snippet), and shuffle the merged pool with a per-query reproducible seed. Each candidate is graded against the relevance scale of Table 2. Crucially, the judge sees only this anonymised pool: opaque identifiers carrying no system identity, rank, score, or filename. We then compute all metrics downstream in Python.

From these labels we compute, entirely in code, our primary rank-aware metric (nDCG@10 [46, §8.4]), lenient precision P@k for $k \in \{1, 3, 5, 10\}$ (fraction of the top-k whose median label is ≥ 0.5), a strict P@10 counting only fully-relevant results (median label $= 1.0$), and MRR. Every metric is reported *per judge* and as the jury mean, each with a query-level 95% confidence interval (CI; mean ± 1.96 standard errors) for both systems and for the paired Code2Code−BM25 difference. One caveat guides interpretation: nDCG@10 is self-normalised within each system's own retrieved top-10 (its ideal ordering is taken over exactly

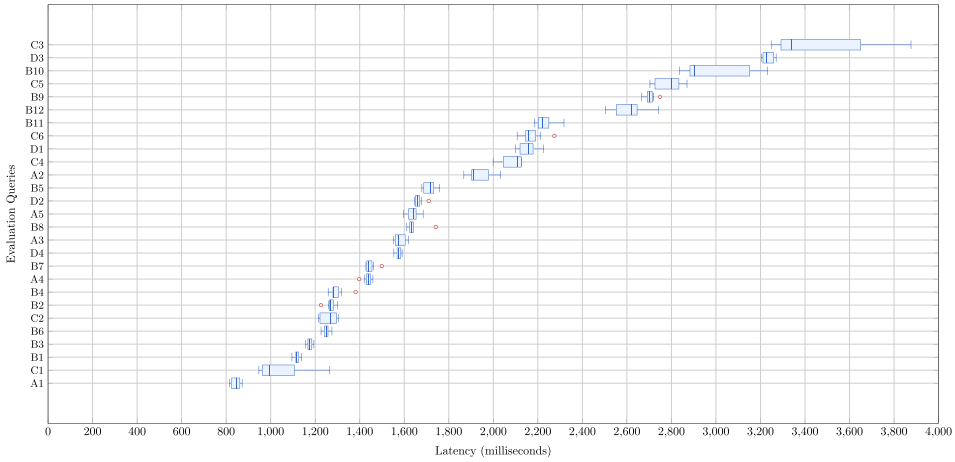

Fig. 3. Per-query end-to-end API latency distribution on the commodity laptop across the 27-query benchmark, measured on commodity CPU hardware (7 independent runs). Queries are sorted by median latency. Latency scales with query snippet length. On the current Toolforge deployment (not illustrated in the figure), the average latency is 31 s, with interquartiles of 19–36 s.

Table 2
Graded relevance scale applied by each blinded judge.

Score	Label	Criterion
1.0	Relevant	Implements the same computational task, or a correct specific solution for the query's intent. Naming, style, and *programming-language* differences do not reduce relevance: a snippet solving the same task in another language scores 1.0.
0.5	Partially relevant	Topically related but a partial or non-idiomatic realisation, or where the target logic appears only as a helper step within a larger function.
0.0	Irrelevant	Does not address the task, or overlaps only superficially (e.g., shared identifiers while doing something unrelated).
null	Indeterminable	Too truncated or context-dependent to judge; used sparingly.

Table 3
Query taxonomy (27 queries across four categories).

Cat.	Difficulty	Rationale
A	Canonical algorithms	Textbook algorithms (GCD, binary search, …). Establishes an upper bound for both systems.
B	Name-obfuscated semantics	Same computational intent as a canonical pattern, but expressed with non-standard identifiers.
C	Cross-language	Query and relevant results are often in different programming languages.
D	Domain-specific/niche	Patterns endemic to the Wikimedia ecosystem.

those items), so it captures how well each system ranks what it returned rather than coverage of the corpus; the *gap between the two systems*, not the absolute score, is therefore the meaningful signal.

Baseline. We compare against Okapi BM25 [47][46, §11.4.3][13, §3.2.1] over the same 1.29 M entities: for each query we extract its non-trivial identifiers (dropping keywords and tokens shorter than three characters) and submit them as a bag-of-words query to a BM25 index over the `code` field of `snippets.db` (the same snippet text used to build the neural embeddings).

Annotation protocol. We adopt the 'LLM-as-judge' paradigm, which approaches human–human agreement rates when strong models are used [48]. Each judge grades only the blinded, pooled, shuffled candidate set and never sees system identity, rank, or the competing ranked list. To guard against single-model idiosyncrasy, adjudication uses a *jury of three judges from different model families* (Claude Opus 4.8, Google Gemini 3.8 Flash, and Moonshot Kimi K2), each queried once, with pinned model identifiers for reproducibility; the Claude judge runs via isolated sub-agents (one per query, each reading only its own blinded prompt). This iteration deliberately uses no human annotators (a cost/time trade-off); we thus report no agreement against a human gold standard and quantify reliability through *inter-judge* agreement (Krippendorff's α of 0.74, interval). We release the blinded prompts, per-judge labels, and analysis code in `scripts/evaluation`.

Table 4
Retrieval quality per judge and as the jury mean, over the 27-query benchmark (point estimates; higher value in each BM25/C2C pair is in bold). P@10^s is strict P@10; nDCG is nDCG@10.

Judge	Sys.	P@1	P@3	P@5	P@10	P@10^s	nDCG	MRR
Claude Opus 4.8	BM25	0.59	0.54	0.50	0.46	0.20	0.71	0.69
	C2C	**0.78**	**0.73**	**0.67**	**0.60**	**0.29**	**0.87**	**0.85**
Gemini 3.8 Flash	BM25	0.48	0.41	0.36	0.28	0.11	0.52	0.55
	C2C	**0.63**	**0.57**	**0.47**	**0.39**	**0.14**	**0.72**	**0.69**
Kimi K2	BM25	0.59	0.49	0.45	0.41	0.15	0.66	0.67
	C2C	**0.78**	**0.72**	**0.64**	**0.53**	**0.19**	**0.85**	**0.86**
Jury mean	BM25	0.56	0.48	0.43	0.38	0.15	0.63	0.64
	C2C	**0.73**	**0.67**	**0.59**	**0.51**	**0.21**	**0.81**	**0.80**

Results

Table 4 reports retrieval quality per judge and the jury mean: Code2Code improves over BM25 on every metric and for every judge.

Our headline metric is the rank-aware nDCG@10, on which Code2Code raises the jury mean to 0.81 from 0.63 for BM25. A 95% CI over the 27 queries gives $[0.73, 0.89]$ for Code2Code versus $[0.51, 0.75]$ for BM25, with the interval on the paired improvement excluding zero under all three judges. Lenient precision moves the same way (jury P@5 0.59 vs 0.43, P@10 0.51 vs 0.38), and top-rank precision follows (P@1 0.73 vs 0.56, MRR 0.80 vs 0.64), though at $n = 27$ these last are high-variance, to be read as supporting evidence rather than standalone claims. Strict P@10 is low for both systems (0.21 vs 0.15), as expected from the demanding rubric.

By category (jury-mean nDCG@10), the improvement concentrates where lexical matching is weakest: Code2Code leads decisively in name-obfuscated category B (0.83 vs 0.53) and cross-language category C

(0.81 vs 0.62), modestly in canonical category A (0.90 vs 0.82), and trails slightly in niche category D (0.64 vs 0.69), where domain-specific lexical retrieval remains competitive.

Discussion

Where Code2Code excels. Code2Code's advantage is largest where the query shares no discriminative tokens with the target: the six biggest per-query gaps (jury nDCG@10 Δ from $+0.53$ to $+1.00$) are all of this kind. In C1 (SHA-256 file hashing), the frequent token `filepath` pulls BM25 toward path utilities (`parseFile`, `validateFilePath`; 0.0 for every judge), whereas Code2Code returns the actual hashers `safeFileHash/getFileHash` at ranks 1–2, fully relevant to all judges. The pattern recurs in B6 and B4 (BM25 0.0) and extends to the cross-language analogues inaccessible to a token matcher, driving its category C lead.

Where the two systems tie. On about a quarter of the queries, the two systems converge, typically when the query retains discriminative library or API tokens that both methods can exploit. Seven are effective ties ($|\Delta\text{nDCG@10}| < 0.05$): A2, A4, A5, B3, B12, C2, and C4; most sit near the ceiling (nDCG@10 ≥ 0.92), the exceptions being A4 and B12, where both are only moderately effective.

Where BM25 wins. Five queries favour the baseline, though only C5 decisively: there the Lua-specific tokens `table`, `pairs`, and `tostring` let BM25 retrieve accurate recursive dumpers (nDCG@10 of 0.89), while Code2Code drifts toward unrelated Wikibase value formatters (0.42). The other four are narrow edges ($|\Delta| \leq 0.27$): B2 (debounce) and B9 (Go worker pool), where Code2Code returns behaviourally adjacent helpers rather than the exact pattern; D2 (git/SWHID hashing), coverage-limited because the distinctive `blob \0` framing is absent from the corpus, so both fall back to generic SHA helpers; and D4 (below). These mark the limits of dense retrieval: domain vocabulary shared by adjacent but irrelevant functions can mislead embedding similarity (the textual bias documented for dense code retrievers [49]), motivating future hybrid ranking.

Judge disagreement and the value of a jury. Because the judges independently grade the same blinded pool, the design surfaces genuine ambiguity rather than hiding it. D4 (MediaWiki hook dispatcher) is the clearest example: BM25 returns the exact target `HookContainer::run` at rank 1 (relevant to all), while Code2Code returns hook-execution *test* helpers (`assertThatHookIsExcutable`); Claude and Kimi rate these partially relevant (0.5; nDCG@10 near 0.98), and Gemini rates these irrelevant (0.0). The jury thus records a BM25 win driven by a legitimate disagreement (whether a test harness counts as a partial answer), that a single judge would have silently resolved either way.

Threats to validity. Three limitations bound our claims. *(i) Benchmark size.* At $n = 27$, the sample size, not the metrics, limits statistical power and widens the top-rank CIs; we therefore lead with nDCG@10 and treat P@1/P@3/MRR as directional support. *(ii) LLM judges.* Evaluation reliability hinges on blinding, three independent model families, and inter-judge agreement ($\alpha = 0.74$), though we make no use of human-level adjudication. *(iii) Cross-language relevance.* Counting a same-task snippet in another language as fully relevant reflects the tool's intended use but structurally advantages semantic retrieval, and part of the category-C gap follows from this declared choice. Deduplication of identical snippets removes the double-counting, though near-duplicate variants remain.

4. Impact

MediaWiki Code2Code Search introduces semantic retrieval capabilities to the Wikimedia open-source ecosystem. By indexing 1.29 million code snippets, the tool enables ecosystem-scale empirical research, including analysis of code reuse and duplicate functionality [50,51], vulnerability tracing (via the SWH graph [52]), and alignment of best practices with actual implementations.

The system is deeply integrated into the Wikimedia technical landscape: it is registered on Toolhub, documented on MediaWiki.org, featured on the Wikimedia Diff blog, and mapped to structured metadata on Wikidata (`Q139251277`). For new developers, the tool mitigates the "cold start" problem; instead of navigating an unfamiliar 2500-repository corpus via keyword search, they can identify relevant functions, classes, and templates by providing a code sketch. Experienced developers benefit from granular frontend selectors that let them filter precisely by entity type, language, and repository group.

Reflecting the WMF's commitment to equitable access, the interface is localised through the community translation platform translatewiki.net.[3] Beginning with 17 languages (14 of them Indian), the catalogue is now maintained and extended by volunteer translators (30 languages at the time of writing, and growing), reaching from widely-spoken national languages to regional and minority ones (e.g., Chinese, Turkish, Korean, Irish, Galician, Luxembourgish, Santali). New and updated translations are contributed on translatewiki.net and merged automatically, redeploying without developer intervention. This reduces linguistic barriers for international contributors and broadens commitment to linguistic diversity and equity. Moreover, by adopting SWHIDs as first-class identifiers for every indexed entity, the system renders each retrieved snippet permanently citable in academic work and documentation, operationalising the UNESCO Recommendation on Open Science [53]. Crucially, it does so at the granularity of individual software entities (each function, type, and template becoming a uniquely identified, referenceable artefact), advancing the broader agenda of reusable, well-referenced research software [54,55]. These persistent identifiers further allow archived entities to be reconciled into knowledge graphs, connecting code archiving to Wikidata through emerging software-citation and code-grounding workflows [56,57].

The service has been deployed on Wikimedia Toolforge (since April 2026). Its RESTful API, documented via Swagger UI at the `/docs` endpoint, facilitates programmatic integration into IDE plugins, bots, automated code-review workflows, and retrieval-augmented code generation [58,59].

5. Conclusions

We have presented MediaWiki Code2Code Search, a neural retrieval system for the semantic discovery of structural software entities across the MediaWiki ecosystem. By leveraging Tree-sitter to index about 1.29 million functions, types, and templates, the system facilitates efficient navigation through 2500+ repositories. Our implementation encodes these entities as 1024-dimensional vectors using the `Qwen3-Embedding-0.6B` model. The resulting FAISS IVF-PQ index, occupying only 168.6 MiB, represents a 96.6% reduction in memory compared to a flat `float32` baseline, enabling the system to serve nearest-neighbour queries on commodity CPU hardware with a median latency of 1.85 s.

The architecture employs a split-build paradigm that decouples GPU-intensive vectorisation from resource-constrained deployment. This enables the use of state-of-the-art neural models within the Wikimedia Toolforge environment while maintaining strict adherence to open-source infrastructure requirements.

Beyond its technical implementation, the system reinforces three core values: (i) open reproducibility, achieved by archiving all repositories in Software Heritage and attaching ISO/IEC 18670:2025-compliant SWHIDs to every result; (ii) linguistic equity, provided through a community-maintained localised interface (30 languages at the time of writing, 14 of which are Indian) continuously expanded via translatewiki.net; and (iii) ecosystem discoverability, through integration with Toolhub, Wikidata, and MediaWiki.org. While the current

[3] https://translatewiki.net/wiki/Translating:MediaWiki_Code2Code_Search.

deployment focuses on MediaWiki, the underlying architecture is inherently modular and portable. Future research will explore cross-encoder reranking to enhance precision in critical workflows (extending the single-stage recall layer evaluated here into the recall-then-rerank pipeline [16]), implement incremental reindexing for near-real-time updates, and systematically evaluate the model's performance in supporting natural-language queries.

Funding

This study was funded by the Alfred P. Sloan Foundation with the grant #G-2025-25193 (sloan.org). It was also funded by Wikimedia CH via their micro-grant programme (https://w.wiki/SrcE).

Declaration of competing interests

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Francesco Tosoni reports a relationship with Software Heritage (Inria Paris) that includes travel reimbursement. Francesco Tosoni reports a relationship with Wikimedia Foundation that includes travel reimbursement. In July 2025, the author conducted a research visit to Inria, supported by Software Heritage (SWH) through travel reimbursement, to discuss compression and code-search algorithms. Furthermore, the author is currently engaged in ongoing research collaborations with SWH regarding the development of archival and retrieval technologies. Additionally, the author has received a travel scholarship from the Wikimedia Foundation to present research related to software preservation at Wikimania 2026. These activities are academic and collaborative in nature and are not perceived to have influenced the integrity of the reported work. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

All the offline code-vectorisation computations presented in this paper were performed using the GRICAD infrastructure (https://gricad.univ-grenoble-alpes.fr), which is supported by Grenoble research communities.

The author thanks Pierre Girard and the whole SOS Gricad team for technical support. He also thanks the Software Heritage (SWH) team in Paris and the members of CodeCommons WP3 for continuous technical feedback and discussions on this work; in particular, Erika Lena, Leonardo Venuta and Paolo Ferragina (Acube Lab, Pisa), and Gaël de Chalendar and Asma Graiess (CEA-List).

Special thanks are due to the Wikimedia Toolforge community and administrators for providing the infrastructure, and to the members of the Indic MediaWiki Developers User Group and the Italian Wikimedia technical community for their insightful discussions and feedback.

References

[1] Turki H, Taieb MAH, Aouicha MB, Rasberry L, Mietchen D. Preregistration: comparing the use of wikidata and wikipedia by open-source software programmers on GitHub repositories. In: Kaffee L, Razniewski S, Alghamdi K, Arnaout H, editors. Proceedings of the wikidata workshop 2023 co-located with 22nd international semantic web conference (ISWC 2023), Athens, Greece, November 13, 2023, Vol. 3640 of CEUR workshop proceedings. CEUR-WS.org; 2023. https://ceur-ws.org/Vol-3640/paper10.pdf

[2] Koren Y. Working with MediaWiki. 2nd ed. WikiWorks Press; 2022. https://workingwithmediawiki.com/book/, third printing.

[3] Milne D, Witten IH. An open-source toolkit for mining wikipedia. Artif Intell 2013;194:222–39, Artificial Intelligence, Wikipedia and Semi-Structured Resources. https://doi.org/10.1016/j.artint.2012.06.007. https://www.sciencedirect.com/science/article/pii/S000437021200077X

[4] Vrandečić D, Krötzsch M. Wikidata: a free collaborative knowledgebase. Commun ACM 2014;57(10):78–85. https://doi.org/10.1145/2629489

[5] Seidlmayer E, Voß J, Melnychuk T, Galke L, Tochtermann K, Schultz C, Förstner KU. ORCID for Wikidata. Data enrichment for scientometric applications. In: Kaffee L, Tifrea-Marciuska O, Simperl E, Vrandecic D, editors. Proceedings of the 1st wikidata workshop (wikidata 2020) co-located with 19th international semantic web conference(OPub 2020), virtual conference, November 2–6, 2020, CEUR Workshop Proceedings. CEUR-WS.org; 2020. https://ceur-ws.org/Vol-2773/paper-09.pdf

[6] Nielsen F, Mietchen D, Willighagen E. Scholia, scientometrics and wikidata. In: Blomqvist E, Hose K, Paulheim H, Ławrynowicz A, Ciravegna F, Hartig O, editors. The semantic web: ESWC 2017 satellite events. Cham: Springer International Publishing; 2017. p. 237–59. https://doi.org/10.1007/978-3-319-70407-4_36

[7] Rossenova L, Duchesne P, Blümel I. Wikidata and wikibase as complementary research data management services for cultural heritage data. In: Wikidata 2022: wikidata workshop 2022, proceedings of the 3rd wikidata workshop 2022 co-located with the 21st international semantic web conference (ISWC2022); 2022. p. 15. https://doi.org/10.25968/opus-2573. https://ceur-ws.org/Vol-3262/paper15.pdf

[8] Shuai J, Xu L, Liu C, Yan M, Xia X, Lei Y. Improving code search with co-attentive representation learning. In: Proceedings of the 28th international conference on program comprehension, ICPC '20. New York, NY, USA: Association for Computing Machinery; 2020. p. 196–207. https://doi.org/10.1145/3387904.3389269

[9] Luan S, Yang D, Barnaby C, Sen K, Chandra S. Aroma: code recommendation via structural code search. Proc ACM Program Lang Oct 2019;3(OOPSLA). https://doi.org/10.1145/3360578

[10] MediaWiki Codesearch. MediaWiki, Online; accessed 2026 2026. https://www.mediawiki.org/wiki/Codesearch

[11] hound-search, Norton K, Klein J. Hound: lightning fast code searching made easy, GitHub repository; 2015. https://github.com/hound-search/hound

[12] Allamanis M, Barr ET, Devanbu P, Sutton C. A survey of machine learning for big code and naturalness. ACM Comput Surv Jul 2018;51(4). https://doi.org/10.1145/3212695

[13] Grainger T, Turnbull D, Irwin M. AI-powered search. Shelter Island, NY: Manning Publications; 2024, Foreword by Grant Ingersoll.

[14] Gu X, Zhang H, Kim S. Deep code search. In: 2018 IEEE/ACM 40th international conference on software engineering (ICSE); 2018. p. 933–44. https://doi.org/10.1145/3180155.3180167

[15] Sachdev S, Li H, Luan S, Kim S, Sen K, Chandra S. Retrieval on source code: a neural code search. In: Proceedings of the 2nd ACM SIGPLAN international workshop on machine learning and programming languages, MAPL 2018. New York, NY, USA: Association for Computing Machinery; 2018. p. 31–41. https://doi.org/10.1145/3211346.3211353

[16] Hu F, Wang Y, Du L, Li X, Zhang H, Han S, Zhang D. Revisiting code search in a two-stage paradigm. In: Proceedings of the sixteenth ACM international conference on web search and data mining, WSDM '23. New York, NY, USA: Association for Computing Machinery; 2023. p. 994–1002. https://doi.org/10.1145/3539597.3570383

[17] Martinez-Gil J, Yin S. Evaluation of code similarity search strategies in large-scale codebases. Berlin, Heidelberg: Springer Berlin Heidelberg; 2025. p. 99–113. https://doi.org/10.1007/978-3-662-70140-9_4

[18] Fraine J, Pintscher L, Saadé P. WikidataSearch: retrieval augmented generation with the wikidata vector database; 2026. Web application:, https://wd-vectordb.wmcloud.org, https://github.com/wmde/WikidataSearch

[19] Wikimedia Foundation Information Retrieval Working Group. Readers/Information retrieval, MediaWiki project page; 2026. https://www.mediawiki.org/wiki/Readers/Information_Retrieval

[20] Sridhar P, Lee H, Dutta A, Zisserman A. WISE image search engine, Wiki Workshop 2023; May 2023. https://wikiworkshop.org/2023/papers/WikiWorkshop2023_paper_43.pdf, 10th edition, May 11, 2023; web application:, https://wise.wmcloud.org/

[21] Sridhar P, Lee H, Pinto DMS, Zisserman A, Dutta A. WISE: a multimodal search engine for visual scenes, audio, objects, faces, speech, and metadata. In: Proceedings of the 49th international ACM SIGIR conference on research and development in information retrieval, SIGIR '26. New York, NY, USA: Association for Computing Machinery; 2026. p. 5230–4. https://doi.org/10.1145/3805712.3808375

[22] Mens T, De Roover C, Cleve A, editors. Software ecosystems: tooling and analytics. 1st ed. Springer Cham; 2023. https://doi.org/10.1007/978-3-031-36060-2

[23] Abramatic J, Di Cosmo R, Zacchiroli S. Building the universal archive of source code. Commun ACM 2018;61(10):29–31. https://doi.org/10.1145/3183558

[24] Di Cosmo R. Archiving and referencing source code with software heritage. In: Bigatti AM, Carette J, Davenport JH, Joswig M, de Wolff T, editors. Mathematical software – ICMS 2020. Cham: Springer International Publishing; 2020. p. 362–73. https://doi.org/10.1007/978-3-030-52200-1_36

[25] Courtès L, Sample T, Zacchiroli S, Tournier S. Source code archiving to the rescue of reproducible deployment. In: Proceedings of the 2nd ACM conference on reproducibility and replicability, ACM REP '24. New York, NY, USA: Association for Computing Machinery; 2024. p. 36–45. https://doi.org/10.1145/3641525.3663622

[26] ISO/IEC 18670:2025. Information technology — SoftWare Hash IDentifier (SWHID) specification V1.2. Apr, 2025. https://www.iso.org/standard/89985.html.

[27] SWHID Working Group. SWHID.org — SoftWare hash IDentifier (SWHID), Official website of the SWHID international standard (ISO/IEC 18670:2025); 2025. https://www.swhid.org

[28] Niu C, Li C, Ng V, Chen D, Ge J, Luo B. An empirical comparison of pre-trained models of source code. In: Proceedings of the 45th international conference on software engineering, ICSE '23. IEEE Press; 2023. p. 2136–48. https://doi.org/10.1109/ICSE48619.2023.00180

[29] Zhang Y, Li M, Long D, Zhang X, Lin H, Yang B, Xie P, Yang A, Liu D, Lin J, Huang F, Zhou J. Qwen3 embedding: advancing text embedding and reranking through foundation models. arXiv:2506.05176. 2025.

[30] Venuta L, Tosoni F, Ferragina P. Candidate recall in large-scale code-to-code retrieval: a comprehensive benchmark. In: Web information systems engineering – WISE 2026. Springer; 2026, To appear.

[31] Wagner TA, Graham SL. Efficient and flexible incremental parsing. ACM Trans Program Lang Syst 1998;20(5):980–1013. https://doi.org/10.1145/293677.293678
[32] Latif A, Azam F, Anwar MW, Zafar A. Comparison of leading language parsers – ANTLR, JavaCC, SableCC, tree-sitter, yacc, bison. In: 2023 13th international conference on software technology and engineering (ICSTE); 2023. p. 7–13. https://doi.org/10.1109/ICSTE61649.2023.00009
[33] Refolli F, Sas D, Fontana FA. Lessons learned from implementing a language-agnostic dependency graph parser. In: Proceedings of the 20th international conference on evaluation of novel approaches to software engineering – ENASE. INSTICC, SciTePress; 2025. p. 484–91. https://doi.org/10.5220/0013277600003928
[34] Reimers N, Gurevych I. Sentence-BERT: sentence embeddings using Siamese BERT-networks. In: Inui K, Jiang J, Ng V, Wan X, editors. Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics; 2019. p. 3982–92. https://doi.org/10.18653/v1/D19-1410. https://aclanthology.org/D19-1410/
[35] Douze M, Guzhva A, Deng C, Johnson J, Szilvasy G, Mazaré P-E, Lomeli M, Hosseini L, Jégou H. The faiss library. IEEE Trans Big Data 2026;12(2):346–61. https://doi.org/10.1109/TBDATA.2025.3618474
[36] Jégou H, Douze M, Schmid C. Product quantization for nearest neighbor search. IEEE Trans Pattern Anal Mach Intell 2011;33(1):117–28. https://doi.org/10.1109/TPAMI.2010.57
[37] Khan MM, Yu Z. Approaching code search for Python as a translation retrieval problem with dual encoders. Empir Softw Eng 2024;30(1):12. https://doi.org/10.1007/s10664-024-10580-3
[38] Malkov YA, Yashunin DA. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans Pattern Anal Mach Intell 2020;42(4):824–36. https://doi.org/10.1109/TPAMI.2018.2889473
[39] Pan JJ, Wang J, Li G. Vector database management techniques and systems. In: Companion of the 2024 Int. Conf. on management of data, SIGMOD '24. New York, NY, USA: Assoc. for Computing Machinery; 2024. p. 597–604. https://doi.org/10.1145/3626246.3654691
[40] Chen T, Fu C, Wu J, Wu H, Fan H, Ke X, Gao Y, Ni Y, Zeng A. Reveal hidden pitfalls and navigate next generation of vector similarity search from task-centric views: [experiments & analysis]. Proc ACM Manag Data Apr 2026;4(1). https://doi.org/10.1145/3786690
[41] Menghani G. Efficient deep learning: a survey on making deep learning models smaller, faster, and better. ACM Comput Surv Mar 2023;55(12). https://doi.org/10.1145/3578938
[42] Jones MB, Boettiger C, Cabunoc Mayes A, Smith A, Gruenpeter M, Lorentz V, Morrell T, Garijo D, Slaughter P, Niemeyer K, Gil Y, Fenner M, Nowak K, Hahnel M, Coy L, Allen A, Crosas M, Sands A, Chue Hong N, Cruse P, Katz DS, Goble C, Mecum B, Gonzalez-Beltran A, Ross N. CodeMeta: an exchange schema for software metadata, Version 3.0; 2023. https://w3id.org/codemeta/3.0
[43] Husain H, Wu H-H, Gazit T, Allamanis M, Brockschmidt M. CodeSearchNet challenge: evaluating the state of semantic code search. arXiv:1909.09436. 2020.
[44] Khan MAM, Bari MS, Do XL, Wang W, Parvez MR, Joty S. XCodeEval: an execution-based large scale multilingual multitask benchmark for code understanding, generation, translation and retrieval. In: Ku L-W, Martins A, Srikumar V, editors. Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: long papers). Bangkok, Thailand: Association for Computational Linguistics; 2024. p. 6766–805. https://doi.org/10.18653/v1/2024.acl-long.367. https://aclanthology.org/2024.acl-long.367/
[45] Voorhees EM. Variations in relevance judgments and the measurement of retrieval effectiveness. In: Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '98. New York, NY, USA: Association for Computing Machinery; 1998. p. 315–23. https://doi.org/10.1145/290941.291017
[46] Manning CD, Raghavan P, Schütze H. Introduction to information retrieval. Cambridge University Press; 2008. https://doi.org/10.1017/CBO9780511809071
[47] Robertson SE, Walker S, Jones S, Hancock-Beaulieu M, Gatford M. Okapi at TREC-3. In: Harman DK, editor. Proceedings of the third text REtrieval conference, TREC 1994, Gaithersburg, Maryland, USA, November 2–4, 1994. NIST Special Publication, National Institute of Standards and Technology (NIST); 1994. p. 109–26. http://trec.nist.gov/pubs/trec3/papers/city.ps.gz
[48] Zheng L, Chiang W-L, Sheng Y, Zhuang S, Wu Z, Zhuang Y, Lin Z, Li Z, Li D, Xing EP, Zhang H, Gonzalez JE, Stoica I. Judging LLM-as-a-judge with MT-bench and chatbot arena. In: Proceedings of the 37th international conference on neural information processing systems, NIPS '23. Red Hook, NY, USA: Curran Associates Inc.; 2023.
[49] Gupta D, Lakshmy GG, Xie Y. SACL: understanding and combating textual bias in code retrieval with semantic-augmented reranking and localization. In: Christodoulopoulos C, Chakraborty T, Rose C, Peng V, editors. Findings of the Assoc. for computational linguistics. Suzhou, China: Assoc. for Computational Linguistics; 2025. p. 25052–65. https://doi.org/10.18653/v1/2025.findings-emnlp.1365. https://aclanthology.org/2025.findings-emnlp.1365/
[50] Sajnani H, Saini V, Svajlenko J, Roy CK, Lopes CV. SourcererCC: scaling code clone detection to big-code. In: Proceedings of the 38th international conference on software engineering, ICSE '16. New York, NY, USA: Association for Computing Machinery; 2016. p. 1157–68. https://doi.org/10.1145/2884781.2884877
[51] Svajlenko J, Islam JF, Keivanloo I, Roy CK, Mia MM. Towards a big data curated benchmark of inter-project code clones. In: Proceedings of the 2014 IEEE international conference on software maintenance and evolution, ICSME '14. USA: IEEE Computer Society; 2014. p. 476–80. https://doi.org/10.1109/ICSME.2014.77
[52] Pietri A, Spinellis D, Zacchiroli S. The software heritage graph dataset: large-scale analysis of public software development history. In: Proceedings of the 17th international conference on mining software repositories, MSR '20. New York, NY, USA: Association for Computing Machinery; 2020. p. 1–5. https://doi.org/10.1145/3379597.3387510
[53] UNESCO, Canadian Commission for UNESCO. An introduction to the UNESCO recommendation on open science, Programme and meeting document SC-PBS-STIP/2022/OSB/1. Licensed under CC BY-SA 3.0 IGO UNESCO; 2022. https://doi.org/10.54677/XOIR1696. https://unesdoc.unesco.org/ark:/48223/pf0000383771
[54] Di Cosmo R, Granger S, Hinsen K, Jullien N, Le Berre D, Louvet V, Maumet C, Maurice C, Monat R, Rougier NP. CODE beyond FAIR: a roadmap for reusable research software. Sci Data 2026;13(1):514. https://doi.org/10.1038/s41597-026-06705-6
[55] Di Cosmo R, Granger S, Hinsen K, Jullien N, Le Berre D, Louvet V, Maumet C, Maurice C, Monat R, Rougier NP. Stop treating code like an afterthought: record, share and value it. Nature 2025;646(8084):284–6. https://doi.org/10.1038/d41586-025-03196-0
[56] Pellizzari di San Girolamo CC, Tosoni F. From code archival to knowledge graph: bridging software heritage, COAR notify and wikidata. In: Proceedings of the wikidata workshop 2026 co-located with ISWC 2026; 2026, To appear.
[57] Pennino F, Gurioli A, Zacchiroli S, Gabbrielli M, Ferragina P. CodeGraph: open-taxonomy knowledge graph for source code with wikidata grounding. In: Proceedings of the 35th ACM international conference on information and knowledge management (CIKM '26). Rome, Italy: ACM; 2026, To appear. https://doi.org/10.1145/3799682.3840769
[58] Zhang S, Ding Y, Lian S, Song S, Li H. CodeRAG: finding relevant and necessary knowledge for retrieval-augmented repository-level code completion. In: Christodoulopoulos C, Chakraborty T, Rose C, Peng V, editors. Proceedings of the 2025 conference on empirical methods in natural language processing. Suzhou, China: Association for Computational Linguistics; 2025. p. 23278–88. https://doi.org/10.18653/v1/2025.emnlp-main.1187. https://aclanthology.org/2025.emnlp-main.1187/
[59] Wang ZZ, Asai A, Yu XV, Xu FF, Xie Y, Neubig G, Fried D. CodeRAG-Bench: can retrieval augment code generation? In: Chiruzzo L, Ritter A, Wang L, editors. Findings of the association for computational linguistics: NAACL 2025. Albuquerque, New Mexico: Association for Computational Linguistics; 2025. p. 3199–214. https://doi.org/10.18653/v1/2025.findings-naacl.176. https://aclanthology.org/2025.findings-naacl.176/