RTLSCOPE: FINE-GRAINED UNCERTAINTY-AWARE RTL RETRIEVAL WITH PROBABILISTIC EMBEDDINGS

Anonymous authors
Paper under double-blind review

ABSTRACT

There is increasing deployment of LLM and agentic applications in the chip design industry, and fine-grained natural-language-to-RTL retrieval becomes essential as it grounds queries in the RTL code snippets that hardware-design workflows require. Such queries span abstraction levels and vary in detail, from local RTL operations to behavioral intent, and a single description can align with multiple snippets; existing benchmarks, however, focus on RTL generation and leave this retrieval capability unmeasured. The limited fault tolerance of chip design makes an uncertainty score useful for decision-making and human review. We introduce RTLSCOPE, a repository-scale benchmark for fine-grained RTL retrieval built from practical large-scale open-source designs; two complementary construction pipelines are built to obtain multi-level queries and one-to-many natural language–RTL alignments. We further propose a probabilistic embedding method that learns alignment with a multiplicity-aware loss and estimates uncertainty. With the proposed method, we achieve 15.0% and 15.2% increases on MRR and MAP over the strongest fine-tuned baseline. Together, the benchmark and method enable reliable RTL retrieval for design, agentic debugging, verification assistance, design comprehension, and other retrieval-augmented hardware-design workflows.

1 INTRODUCTION

Large language models and agents are increasingly adopted across hardware design, from code generation and debugging to verification Wang et al. (2024b); Xu et al. (2024); Wang et al. (2025b); Fu et al. (2026). These workflows receive natural-language (NL) intent in the form of design specifications, verification plans, bug reports, and agent instructions Wang et al. (2025a); Xing et al. (2026b). Grounding this intent in the code snippets that implement the relevant behavior is an essential first step for design comprehension, code review, debugging, verification, and retrieval-augmented generation Wang et al. (2025c); Yao et al. (2025); Wang et al. (2024a). Chip design primarily uses Verilog/SystemVerilog at the register-transfer level (RTL), which describes a circuit as registers, combinational logic, and the data transfers between them. A design engineer may need to pinpoint the logic that implements a protocol, while a debugging agent needs to retrieve the relevant code before proposing a localized repair. All these require fine-grained NL–RTL alignment.

Existing RTL benchmarks have established important evaluations for natural-language-to-RTL generation (Yang et al., 2025; Wan et al., 2025; Li et al., 2025). VerilogEval evaluates generated implementations for instructional, self-contained Verilog tasks (Liu et al., 2023), primarily covering relatively simple, single-functionality modules, such as instruction decoders, ALUs, and FIFOs. RTLLM and related work broaden generation assessment beyond syntax alone (Lu et al., 2024). Repository-scale benchmarks utilize multi-file context for generating or completing RTL in practical projects (Allam & Shalan, 2024; Wu et al., 2025). These benchmarks evaluate whether a model generates a correct implementation, leaving fine-grained NL–RTL retrieval unmeasured.

General-purpose code retrieval offers a useful foundation for matching natural language and code, but RTL presents distinct challenges: its corpus is severely limited, and its semantics combine concurrency, timing, hierarchy, and data/control dependencies that differ from those of programming languages. Degraded performance has been observed when applying general-purpose models to hardware-design scenarios (Wan et al., 2026). Therefore, a fine-grained NL–RTL retrieval benchmark is needed to cover complex, large-scale practical designs and realistic query challenges. Given

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

a natural-language description and a repository-scale RTL corpus, the task is to retrieve the fine-grained RTL snippets that jointly implement the described behavior.

Beyond the scarcity of RTL data and its semantic differences from programming languages, fine-grained NL–RTL retrieval presents three connected challenges. First, descriptions span abstraction levels and vary in detail, from specific local RTL operations to broader behavioral intent. Second, the task exhibits multiplicity: a description is often implemented by multiple snippets distributed across control logic, state updates, datapath operations, interface handling, and module boundaries. Third, chip design flow has little fault tolerance; retrieval results need to be auditable, and retrieval systems need to estimate query uncertainty.

Since correctness is central to chip design, it is desirable that retrieval can provide more than a list of candidates. Downstream users, including design engineers and hardware-design agents, need decision-relevant information about whether results can be trusted for design, debugging, verification, or the next design action. This motivates uncertainty quantification for retrieval. Probabilistic embeddings represent inputs as distributions rather than deterministic points, capturing representation uncertainty while yielding interpretable uncertainty estimates (Chun et al., 2021; Oh et al., 2019; Tang et al., 2025). Building on this principle, we represent queries and RTL candidates in a probabilistic embedding space, which supports uncertainty-aware retrieval. The proposed multiplicity-aware loss accommodates the one-to-many relations in NL–RTL alignments.

In this work, our contributions are summarized as follows:

- We introduce RTLSCOPE, a repository-scale benchmark for fine-grained NL–RTL retrieval over large-scale designs. The benchmark captures natural-language queries across abstraction levels and degrees of detail, with one-to-many NL–RTL alignments.
- We develop two complementary construction pipelines, revision-grounded query generation and round-trip-verified query synthesis, to provide natural-language queries with varied detail and abstraction levels. We construct graph-assisted hard negatives for contrastive NL–RTL alignment training.
- We propose a probabilistic embedding method and a multiplicity-aware loss that train on NL–RTL pairs with multiplicity and provide variance signals for uncertainty-aware RTL retrieval.
- We conduct quantitative and qualitative evaluations of the benchmark and method. The results show that the variance produced by the probabilistic embedding reflects uncertainty and that our method outperforms the strongest fine-tuned baselines on all metrics (15.0% on MRR, 15.2% on MAP, 21.5% on Recall@1, and 22.0% on Hit@1).

2 PROBABILISTIC EMBEDDING FOR UNCERTAINTY-AWARE RETRIEVAL

As discussed above, correctness is critical in hardware design, and retrieval systems should provide uncertainty estimates for downstream decision-making. Deterministic embeddings represent each query as a single point and rank query–candidate pairs using a point similarity score. Such a representation does not explicitly characterize uncertainty. We therefore propose a probabilistic embedding that represents each input as a distribution, capturing uncertainty and enabling more nuanced retrieval (Tang et al., 2025), as illustrated in Figure 1.

Following the principle of Chun et al. (2021), we represent each natural-language query q and RTL candidate c as a diagonal Gaussian:

$$p(\mathbf{z} \mid x) = \mathcal{N}\big(\boldsymbol{\mu}_x, \mathrm{diag}(\boldsymbol{\sigma}_x^2)\big), \qquad x \in \{q, c\}. \tag{1}$$

We use CodeRankEmbed (Suresh et al., 2025) as the base encoder and attach a mean head and a variance head that map each input to a diagonal Gaussian with mean $\boldsymbol{\mu}_x$ and standard deviation $\boldsymbol{\sigma}_x$. The mean $\boldsymbol{\mu}_x$ specifies a retrieval location, and the variance $\boldsymbol{\sigma}_x^2$ characterizes uncertainty. We score a query–candidate pair by the negative diagonal-Gaussian mutual likelihood,

$$s(q, c) = -\frac{\gamma}{2} \sum_{d=1}^{D} \left[\frac{(\mu_{q,d} - \mu_{c,d})^2}{\sigma_{q,d}^2 + \sigma_{c,d}^2} + \log\left(\sigma_{q,d}^2 + \sigma_{c,d}^2\right) \right], \tag{2}$$

where $\gamma > 0$ is a learned scale parameter. The first term measures the squared distance between the query and candidate means relative to their combined variance, while the second term penalizes excessive variance.

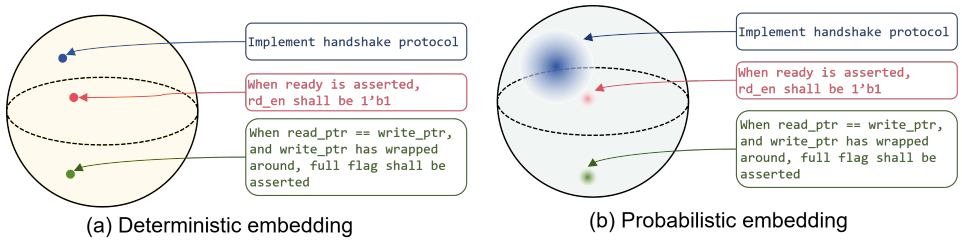


Figure 1: A deterministic embedding maps each query to a single point, whereas the probabilistic embedding represents it by a distribution with input-dependent spread.

Listwise multiplicity-aware learning. In realistic retrieval settings, a query can align with multiple RTL snippets that jointly implement the same behavior. We refer to this property as alignment multiplicity, and the benchmark construction intentionally preserves this structure for embedding model training. In the contrastive learning formulation below, the natural-language item serves as the anchor q and the RTL snippet as the candidate c. The training objective combines three terms: a listwise multiplicity-aware retrieval loss, an auxiliary pair loss, and a KL regularizer.

Let $\mathcal{C}_g = \mathcal{P}_g \cup \mathcal{N}_g$ be the explicit candidate set for query group g, where $\mathcal{P}_g$ contains all known implementation snippets and $\mathcal{N}_g$ contains explicit negatives. We optimize the listwise multiplicity-aware objective

$$\mathcal{L}_{\text{list}}(g) = \log \sum_{c \in \mathcal{C}_g} \exp s(q_g, c) - \sum_{p \in \mathcal{P}_g} w_{g,p}\, s(q_g, p), \qquad \sum_{p \in \mathcal{P}_g} w_{g,p} = 1. \tag{3}$$

where $w_{g,p} = 1/|\mathcal{P}_g|$. All snippets that jointly implement the behavior reinforce the same query. Code with no explicit positive or negative relation is excluded from the loss calculation.

We also include an auxiliary Monte Carlo pair loss that increases the sampled match probability of known query–positive pairs and decreases that of explicitly labeled query–negative pairs. We draw M reparameterized samples $\mathbf{z}_q^{(a)} \sim p(\mathbf{z} \mid q)$ and $\mathbf{z}_c^{(b)} \sim p(\mathbf{z} \mid c)$ from the query and candidate Gaussians, respectively, and define

$$p_{\text{MC}}(q, c) = \frac{1}{M^2} \sum_{a=1}^{M} \sum_{b=1}^{M} \text{sigmoid}\left(\beta - \gamma \left\| \mathbf{z}_q^{(a)} - \mathbf{z}_c^{(b)} \right\|_2^2 \right), \tag{4}$$

where β is a learned bias parameter and γ is the scale parameter in Equation 2. Given explicitly labeled positive pairs $\mathcal{E}^+$ and negative pairs $\mathcal{E}^-$, the pair loss is

$$\mathcal{L}_{\text{pair}} = \frac{1}{2} \left[-\frac{1}{|\mathcal{E}^+|} \sum_{(q,p) \in \mathcal{E}^+} \log p_{\text{MC}}(q, p) - \frac{1}{|\mathcal{E}^-|} \sum_{(q,n) \in \mathcal{E}^-} \log \left(1 - p_{\text{MC}}(q, n)\right) \right]. \tag{5}$$

Following probabilistic embedding methods (Chun et al., 2021; Oh et al., 2019), we include a KL regularizer $\mathcal{L}_{\text{KL}}$ that draws the embeddings toward a standard normal prior $\mathcal{N}(0, I)$. The complete training objective is $\mathcal{L} = \mathcal{L}_{\text{list}} + \mathcal{L}_{\text{pair}} + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}$, where λ_{KL} controls the strength.

Retrieval and uncertainty calculation. At inference, we rank RTL candidates by the mutual-likelihood score in Equation 2. For a query q, we summarize representation uncertainty from the predicted diagonal variance using mean variance,

$$U(q) = \frac{1}{D} \sum_{d=1}^{D} \sigma_{q,d}^2. \tag{6}$$

where D is the embedding dimension. We use this quantity as an uncertainty signal for retrieval and downstream decision-making. Qualitative and quantitative evaluations of the predicted variance are covered in Section 4.2.

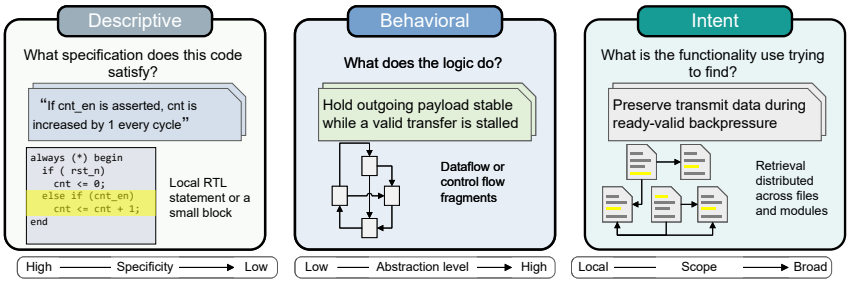


Figure 2: The three query styles, ordered by decreasing specificity and increasing abstraction and scope.

3 Benchmark Construction

We construct RTLScope with two complementary pipelines that produce fine-grained NL–RTL alignments at multiple abstraction levels and preserve the multiplicity of query–RTL alignments. Revision-grounded query generation transforms revision history to natural language queries. Round-trip-verified synthesis (Section 3.3) generates query–RTL pairs from complete designs. We further use the design graph to mine hard negatives for contrastive learning.

3.1 Multi-level Query Semantics

In the benchmark, we define three levels of natural-language queries that vary in abstraction level, detail, and specificity to reflect real-world usage. They are defined as follows:

- **Descriptive: What specification does this code satisfy?**

  Query semantics: A declarative hardware specification implemented by the code. Preserves signal, module, register names and numeric details; describes only code-supported behavior.
  Expected retrieval: Often local and identifier-rich; closely aligns with a specific snippet.
  Example: "When `tx_valid` is asserted and `tx_ready` is deasserted, `tx_data_reg` retains its previous value until the transfer completes."
- **Behavioral: What does this logic do?**

  Query semantics: A functional description of what the code does, expressed through inputs, conditions, state transitions, and effects. Removes identifiers and structural detail.
  Expected retrieval: Retrieve local implementation without relying on lexical name overlap.
  Example: "Hold outgoing payload stable while a valid transfer is stalled."
- **User: What capability is the user trying to find?**

  Query semantics: A concise, user-style query expressing the desired operation, important condition, mechanism, interface, or effect. Omits implementation details and all implementation-specific identifiers.
  Expected retrieval: May retrieve distributed code snippets that jointly implement a high-level functionality.
  Example: "preserve transmit data during ready-valid backpressure."

The construction framework presented in Sections 3.2 and 3.3 can accommodate additional styles. We retain these three as canonical because they cover the main range of specificity and abstraction while keeping query generation and evaluation practical.

3.2 Revision-Grounded Query Generation

Fine-grained NL–RTL alignment data are limited. Revision histories provide a valuable source of supervision because they associate a natural-language commit message with one or more coherent RTL regions changed during the implementation, repair, or refinement. We use these revision records as evidence for generating natural-language queries and RTL snippet pairs.

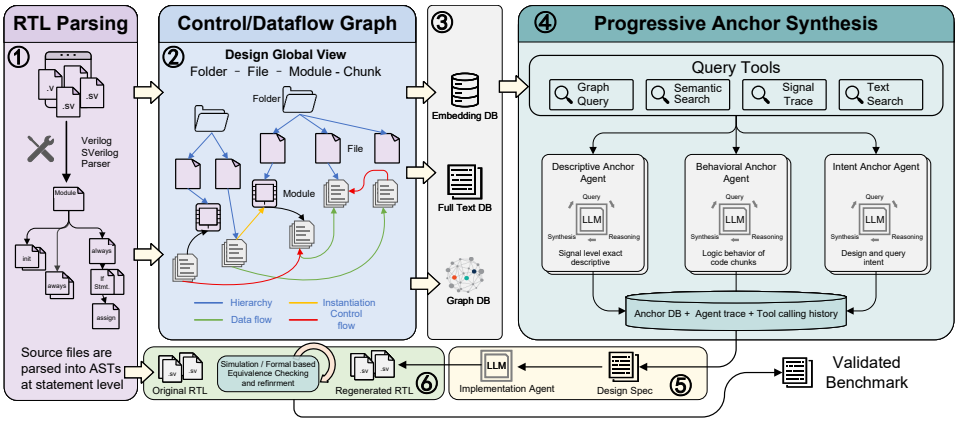


Figure 3: The round-trip-verified query synthesis pipeline. (1) Design parsing, elaboration and graph construction. (2) Graph enhancement through static analysis. (3) Indexing and database. (4) Query synthesis. (5) Implementation plan. (6) Round-trip verification.

Commit messages are unconstrained. Some are vague (e.g., "bug fix"), administrative (e.g., "add copyright"), or contain boilerplate (e.g., "Signed-off-by"), while others may be too short or too long to provide a useful query. We therefore develop a pipeline that filters and processes revision records, generates multi-level queries, and constructs single- and multi-snippet positives.

The pipeline begins with (1) revision extraction, which collects commit messages, changed RTL files, and diff chunks from each repository. In (2) revision quality control, we use an LLM-as-a-judge to determine whether a revision can be retained. We retain messages that describe what the code change does and are specific enough to serve as retrieval anchors.

In (3), statement-level RTL connectivity graph construction, eligible Verilog/SystemVerilog files are parsed into directed graphs where each node represents a syntactically complete RTL statement. Edges represent relationships between these nodes, including data-flow relations, such as a register written by one statement and read by another, and control-flow relations, such as a signal being used by a conditional branch. The graph is used to identify positive code snippets and to mine hard negatives.

In (4) statement-level RTL grounding, we map the changed diff ranges to the statement-level snippets whose source ranges overlap with changed lines. These snippets provide syntactically complete RTL units with the surrounding context needed to interpret their behavior, whereas a raw diff range may contain only a few changed lines from a larger statement. The resulting statement-level snippets are used for query generation and positive construction.

In (5), multi-level query generation, we use LLMs to produce descriptive, behavioral, and user queries from the retained revision message and RTL snippets. In (6), positive construction, each query is paired with either a single snippet or a group of snippets, preserving the one-to-many structure that Section 2 exploits for probabilistic embedding training. The connectivity graph is reused in (7), graph-assisted hard-negative mining, to produce hard negatives; details of this procedure are described in Section 3.4.

3.3 Round-Trip-Verified Query Synthesis

Revision-grounded query generation pairs NL descriptions with RTL snippets, but only for code changed in a revision. The second pipeline complements this coverage by traversing a complete design, providing comprehensive design-wide coverage. Because synthesized queries can be inaccurate, we introduce round-trip verification. The overall flow is shown in Figure 3.

In (1) parsing and elaboration, we construct a statement-level AST from all source files of a repository as a single design graph. In (2) graph enhancement, we add cross-reference, data-flow, and control-flow edges by static analysis. In (3) indexing, we store the graph together with node embeddings and node full text, giving agents graph, semantic, and lexical access to the design.

In (4), agentic query synthesis, three agents operate in parallel to synthesize queries at the three levels defined in Section 3.1. Each agent is prompted for its level and equipped with tools to query the graph, search embeddings, trace signal flow, and look up cross-references. An agent gathers information across tool calls and incrementally submits one or two queries at a time to central storage. Every tool call is recorded with the submission, so each query can be traced to the code snippets inspected during its synthesis.

In (5), the submitted queries are organized into an implementation plan containing natural-language descriptions of the code snippets. In (6), round-trip verification, an agent reimplements a complete RTL design from the plan alone. The plan is accepted if the regenerated design behaves equivalently to the original under the same stimulus, passes the same testbenches, or passes formal equivalence checking. Otherwise, the agent revises the plan until the regenerated design passes verification.

3.4 Graph-Assisted Hard Negative Mining

For effective contrastive learning, negative examples should be closely related to the positives while remaining incorrect matches for the query. We use two strategies. First, we use embedding similarity to identify hard-negative candidates. Both the revision-grounded and round-trip-verified pipelines provide positive query–RTL alignments. RTL snippets that are not in the positive set but have similarity scores above a threshold are selected as hard-negative candidates.

Second, we exploit the structure of the connectivity graphs constructed in Sections 3.2 and 3.3. These graphs provide a natural basis for mining structurally and functionally coupled code snippets. We treat the graph as undirected and let $\mathbf{L}$ be its Laplacian. The effective resistance between positive node p and candidate node v is

$$R_{\text{eff}}(p, v) = (\mathbf{e}_p - \mathbf{e}_v)^\top \mathbf{L}^\dagger (\mathbf{e}_p - \mathbf{e}_v) \tag{7}$$

where $\mathbf{L}^\dagger$ is the Moore–Penrose pseudoinverse and $\mathbf{e}_i$ is the indicator vector for node i. We select candidates whose resistance to a positive node is below a threshold. This measure has an intuitive circuit interpretation: a candidate with low resistance is tightly coupled to the positive context while remaining outside the positive set, which makes it a graph-based hard negative.

4 Evaluation

We report the benchmark characteristics, retrieval performance, and qualitative and quantitative analyses of uncertainty estimation. Besides the queries of the three styles, we keep the sanitized raw commit messages as a noisy-query baseline. RTLScope is still evolving; at the time of this study, it builds upon 15 actively maintained open-source projects, such as OpenTitan, ibex, and PULP, and consists of 13K commits, 134K query–RTL pairs, and 3.5M lines of code. The probabilistic embedding is implemented in PyTorch. The Verilog/SystemVerilog parser is built on Slang, and the construction pipelines use Python, Neo4j, Milvus, and an in-house agent framework. Deepseek-V4-Flash is used for agent and query generation.

4.1 Benchmark Characteristics

RTLScope is designed to represent the variation in specificity, vagueness, and abstraction encountered in realistic queries. We characterize this variation with the specificity score of Ko et al. (2019), where larger values indicate more specific language. We additionally report the mean variance predicted by our probabilistic embedding model, which describes how broadly the model represents a query in the probabilistic embedding space.

Figure 4 reports these distributions for 2,500 sampled queries from each query style. The specificity distributions establish the intended semantic progression. Raw revision messages are concentrated at low specificity, with 62.2% of samples below 0.4. Descriptive queries show higher specificity due to implementation detail, their mean specificity is 0.715. Behavioral queries occupy a nearby but less identifier-dependent regime (mean 0.658), whereas intent queries concentrate at low-to-intermediate specificity (mean 0.47), with 41.9% below 0.4. These separated distributions show that the query

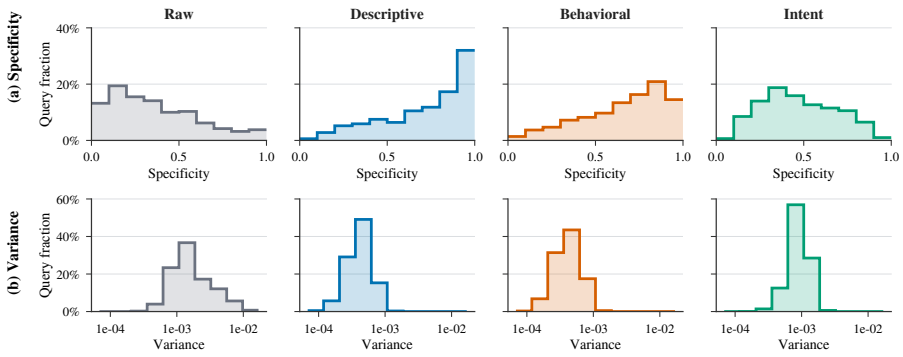


Figure 4: Benchmark query characteristics across raw, descriptive, behavioral, and user styles. Top: specificity-score distributions. Bottom: predicted mean-variance distributions from the probabilistic embedding model.

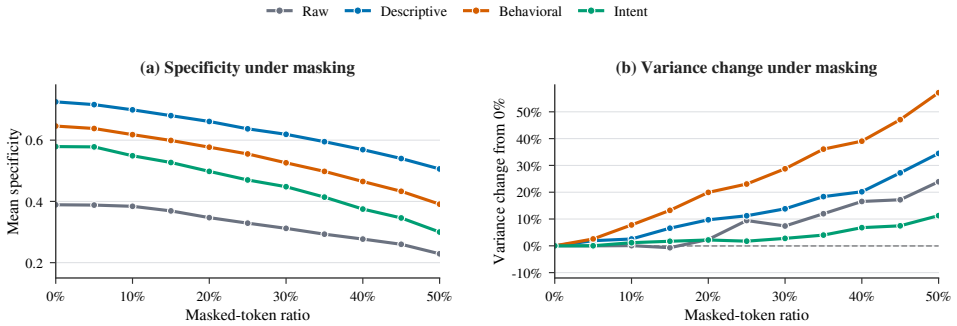


Figure 5: Progressive query-token masking. (a) Mean specificity. (b) Change in mean predicted variance relative to the unmasked (0%) baseline of each style.

styles exhibit intended variation in description granularity. Since the raw style is collected from various sources without control, it is more diverse and has a broader distribution.

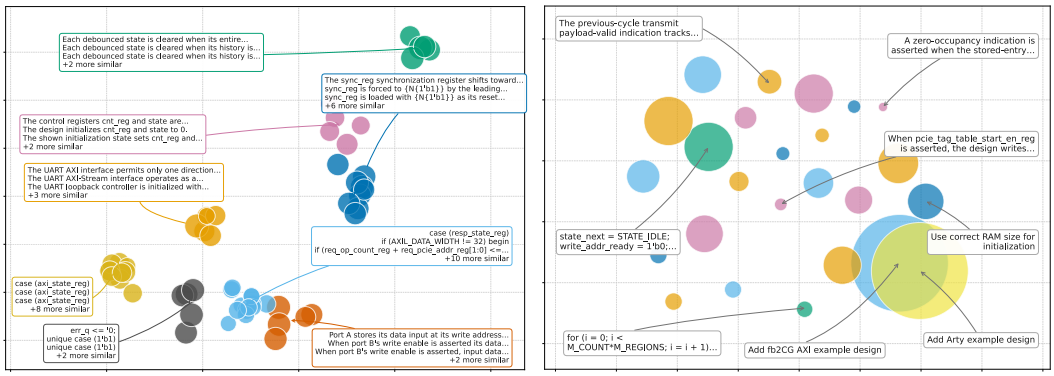
The variance distributions also differ across query styles. Descriptive and behavioral queries concentrate at low-variance distributions. Intent queries shift to a higher variance range, consistent with their broader capability-level phrasing. Raw revision messages have the broadest, highest-variance profile, as they are collected from diverse sources and authored without a unified guideline.

4.2 Uncertainty Quantification

The previous section shows qualitatively that the variance follows the intended trends across the four query styles. We now evaluate quantitatively whether the variance reflects uncertainty. To introduce controlled ambiguity, we apply a masking procedure to the benchmark queries (Chun et al., 2021; Kuhn et al., 2023). For each query, we randomly mask a fraction of its tokens and replace them with a special `[MASK]` token. We increase the masking rate from 0% to 50% with a step size of 5% and compute both the specificity score and the predicted variance for each masked query.

Figure 5 reports specificity and the relative change in mean predicted variance with respect to the unmasked baseline. Specificity decreases monotonically for every style, confirming that masking removes information as intended. For the three generated styles, predicted variance rises monotonically with the masking rate, indicating that the model reflects uncertainty caused by missing information. This inverse relationship between specificity and predicted variance is consistent with the qualitative analysis of Vilnis & McCallum (2015), where more specific words tend to have lower variance.

Figure 6 provides a 2D visualization of the learned embedding space for qualitative analysis. Each colored bubble represents a probability distribution that a query maps to. The size of the bubble



(a) Clustered embeddings with representative queries.

(b) Annotated projection of queries with different variances.

Figure 6: Qualitative 2D visualization of the probabilistic embedding.

Table 1: Retrieval performance on the benchmark. The final row reports the relative improvement of this work over the best method for each metric.

Model	Type	Ranking		Recall (%)			Hit (%)		
		MRR	MAP	R@1	R@5	R@10	H@1	H@5	H@10
BM25	Lexical	0.329	0.288	17.85	36.74	44.53	24.06	42.51	49.70
all-MiniLM-L6-v2	General	0.426	0.392	24.64	47.33	55.08	32.66	53.10	60.68
CodeBERT	Code	0.042	0.037	1.74	4.38	6.50	2.04	5.25	7.88
GraphCodeBERT	Code	0.110	0.096	5.69	12.04	15.85	6.83	14.23	18.69
UniXcoder	Code	0.501	0.463	33.47	55.76	63.32	39.44	62.57	69.69
Frozen CodeRankEmbed	Code	0.597	0.544	41.34	63.45	70.50	49.72	71.22	77.55
Fine-tuned CodeRankEmbed	Fine-tuned	0.617	0.571	39.48	68.93	79.03	48.33	77.83	86.09
This work	**Probabilistic embedding**	**0.709**	**0.658**	**50.23**	**75.12**	**82.28**	**60.68**	**83.61**	**89.18**
Relative gain vs. second best		*15.0%*	*15.2%*	*21.5%*	*9.0%*	*4.1%*	*22.0%*	*7.4%*	*3.6%*

is proportional to the predicted variance. The left panel shows that queries with similar semantics cluster together; for example, several descriptions related to synchronization registers are clustered closely. The right panel illustrates queries with varying uncertainty: "Add Arty example design" corresponds to a large bubble as it is a generic query that covers a wide range of code, while "When `pcie_tag_table_start_en_reg` is asserted..." yields a small bubble because it describes a specific behavior. "`state_next = STATE_IDLE`" contains a specific identifier, however this is a common state transition pattern that can be found in many designs, so it corresponds to a relatively large bubble.

4.3 Retrieval Evaluation

We compare our probabilistic retrieval model with lexical retrieval (BM25), a general sentence encoder (all-MiniLM-L6-v2), and code-specialized embeddings (CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2021), UniXcoder (Guo et al., 2022), and CodeRankEmbed (Suresh et al., 2025)). We also include a fine-tuned CodeRankEmbed baseline. We report mean reciprocal rank (MRR), mean average precision (MAP), Recall@K, and Hit@K, and evaluate retrieval with progressively masked query tokens using the masking protocol in Section 4.2.

Table 1 reports the results. General-purpose code encoders transfer poorly to fine-grained RTL retrieval. CodeBERT and GraphCodeBERT fall below lexical matching. Our probabilistic model achieves the highest score on every metric, improving MRR from 0.617 to 0.709 and MAP from 0.571 to 0.658 over the strongest baseline.

Figure 7 shows retrieval performance as the fraction of masked query tokens increases. Retrieval quality declines for every method, yet our probabilistic model retains the highest absolute perfor-

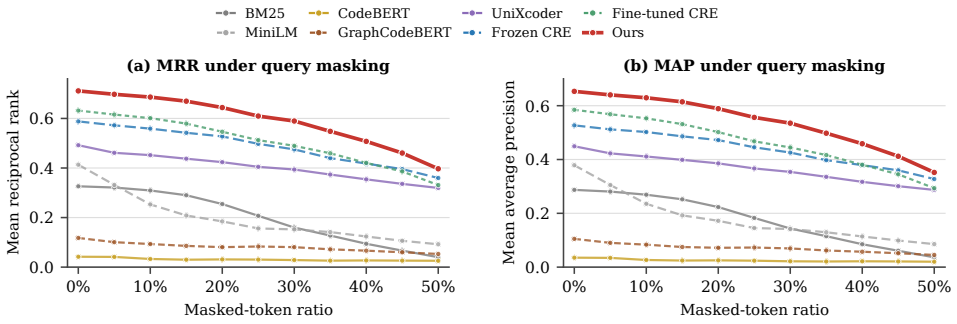


Figure 7: Retrieval under progressive random query-token masking.

mance. This shows that the learned probabilistic embedding continues to rank relevant RTL under substantially incomplete natural-language queries.

5 Related Work

Existing RTL benchmarks primarily focus on specification-to-code generation. VerilogEval provides an executable evaluation suite built from HDLBits tasks (Liu et al., 2023); RTLLM, RTL-Repo, and VeriBench extend the design space toward quality-aware, repository-scale, and multi-dimensional evaluation (Lu et al., 2024; Allam & Shalan, 2024; Agarwal et al., 2025; Abdelatty et al., 2025). RTLCoder demonstrates the value of large instruction–RTL corpora for adapting RTL models (Liu et al., 2025a). RTLRepoCoder focuses on code completion (Wu et al., 2025). RTLocating introduces ΔSpec-to-RTL localization over git commits from OpenTitan (Xing et al., 2026a). These studies focus on RTL generation or completion. The closest work to ours is DeepRTL (Liu et al., 2025b;c), which generates Verilog annotations at line, block, and module granularity. Its dataset varies annotation detail across code units, but does not organize queries by semantic level, specificity, abstraction, or retrieval intent, nor does it consider NL–RTL multiplicity. This work addresses fine-grained NL–RTL retrieval across queries with varied abstraction and specificity, together with uncertainty estimation.

Semantic code search evaluates the alignment of natural-language queries and code. CodeSearchNet introduced expert relevance annotations for this setting at large scale (Husain et al., 2019), and CoSQA provides human-annotated web-query/code pairs (Huang et al., 2021). RTL retrieval also involves hardware-specific semantics such as concurrency, timing, and hierarchy, while available hardware-design retrieval data remain limited.

Probabilistic embeddings represent each item as a distribution rather than a deterministic point, so the spread captures representation uncertainty. Gaussian word embeddings introduced this idea for words (Vilnis & McCallum, 2015), and Oh et al. (2019) proposed hedged instance embeddings to quantify per-example uncertainty. PCME applied probabilistic embeddings to cross-modal retrieval by mapping both modalities to Gaussians and scoring matches by their probabilistic similarity (Chun et al., 2021); later work proposed an InfoNCE-based contrastive learning approach for probabilistic embeddings (Kirchhof et al., 2023) and extended probabilistic embeddings to composed retrieval (Tang et al., 2025) and binary embeddings (Cheng et al., 2025).

6 Conclusion

We introduced RTLScope, a repository-scale benchmark for fine-grained natural-language-to-RTL retrieval with multi-level queries and one-to-many alignments. Its probabilistic embedding method combines Gaussian representations with multiplicity-aware supervision and achieves the best result on every retrieval metric. The predicted variance also increases under query masking, providing an uncertainty signal for downstream RTL retrieval workflows.

REFERENCES

Manar Abdelatty, Jingxiao Ma, and Sherief Reda. Metrex: A benchmark for verilog code metric reasoning using llms. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pp. 995–1001, 2025.

Mihir Agarwal, Zaqi Momin, Kailash Prasad, and Joycee Mekie. Veribench: Benchmarking large language models for verilog code generation and design synthesis. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2025. doi: 10.1109/ISCAS56072.2025.11044004. URL https://doi.org/10.1109/ISCAS56072.2025.11044004.

Ahmed Allam and Mohamed Shalan. Rtl-repo: A benchmark for evaluating llms on large-scale rtl design projects. In *IEEE LLM Aided Design Workshop (LAD)*, 2024. doi: 10.1109/LAD62341.2024.10691810. URL https://doi.org/10.1109/LAD62341.2024.10691810.

Kun Cheng, Qibing Qin, Wenfeng Zhang, Lei Huang, and Jie Nie. Deep probabilistic binary embedding via learning reliable uncertainty for cross-modal retrieval. In *Proceedings of the 33rd ACM international conference on multimedia*, pp. 6393–6402, 2025.

Sanghyuk Chun, Seong Joon Oh, Rafael Sampaio De Rezende, Yannis Kalantidis, and Diane Larlus. Probabilistic embeddings for cross-modal retrieval. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8411–8420. IEEE, 2021.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. In *Findings of the association for computational linguistics: EMNLP 2020*, pp. 1536–1547, 2020.

Lik Tung Fu, Jie Zhou, Shaokai Ren, Mengli Zhang, Jia Xiong, Hugo Jiang, Nan Guan, Xi Wang, and Jun Yang. Chatsva: Bridging sva generation for hardware verification via task-specific llms. *arXiv preprint arXiv:2604.02811*, 2026.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. Graphcodebert: Pre-training code representations with data flow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=jLoC4ez43PZ.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7212–7225, 2022.

Junjie Huang, Duyu Tang, Linjun Shou, Ming Gong, Ke Xu, Daxin Jiang, Ming Zhou, and Nan Duan. Cosqa: 20,000+ web queries for code search and question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 5690–5700, 2021.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

Michael Kirchhof, Enkelejda Kasneci, and Seong Joon Oh. Probabilistic contrastive learning recovers the correct aleatoric uncertainty of ambiguous inputs. In *International Conference on Machine Learning*, pp. 17085–17104. PMLR, 2023.

Wei-Jen Ko, Greg Durrett, and Junyi Jessy Li. Domain agnostic real-valued specificity prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 6610–6617, 2019.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=VD-AYtP0dve.

Zeju Li, Changran Xu, Zhengyuan Shi, Zedong Peng, Yi Liu, Yunhao Zhou, Lingfeng Zhou, Chengyu Ma, Jianyuan Zhong, Xi Wang, et al. Deepcircuitx: A comprehensive repository-level dataset for rtl code understanding, generation, and ppa analysis. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 204–211. IEEE, 2025.

Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. Verilogeval: Evaluating large language models for verilog code generation. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023. doi: 10.1109/ICCAD57390.2023.10323812. URL https://doi.org/10.1109/ICCAD57390.2023.10323812.

Shang Liu, Wenji Fang, Yao Lu, Jing Wang, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025a. doi: 10.1109/TCAD.2024.3483089. URL https://doi.org/10.1109/TCAD.2024.3483089.

Yi Liu, Changran Xu, Yunhao Zhou, Zeju Li, and Qiang Xu. Deeprtl: Bridging verilog understanding and generation with a unified representation model. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025b. URL https://openreview.net/forum?id=2hcfoCHKoB.

Yi Liu, Hongji Zhang, Yunhao Zhou, Zhengyuan Shi, Changran Xu, and Qiang Xu. Deeprtl2: A versatile model for rtl-related tasks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6485–6500, 2025c.

Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. Rtllm: An open-source benchmark for design rtl generation with large language model. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024. doi: 10.1109/ASP-DAC58780.2024.10473904. URL https://doi.org/10.1109/ASP-DAC58780.2024.10473904.

Seong Joon Oh, Kevin P. Murphy, Jiyan Pan, Joseph Roth, Florian Schroff, and Andrew C. Gallagher. Modeling uncertainty with hedged instance embeddings. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=r1xQQhAqKX.

Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. Cornstack: High-quality contrastive data for better code retrieval and reranking. In *International Conference on Learning Representations*, volume 2025, pp. 15800–15817, 2025.

Haomiao Tang, Jinpeng Wang, Yuang Peng, GuangHao Meng, Ruisheng Luo, Bin Chen, Long Chen, Yaowei Wang, and Shu-Tao Xia. Modeling uncertainty in composed image retrieval via probabilistic embeddings. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1210–1222, 2025.

Luke Vilnis and Andrew McCallum. Word representations via gaussian embedding. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6623.

Gwok-Waa Wan, SamZaak Wong, Mengnv Xing, Nan Guan, Ning Xu, Qiang Xu, Xi Wang, et al. Genben: A genarative benchmark for llm-aided design. 2025.

Gwok-Waa Wan, SamZaak Wong, Shengchu Su, Chenxu Niu, Ning Wang, Xinlai Wan, Qixiang Chen, Mengnv Xing, Jingyi Zhang, Jianmin Ye, et al. Fixme: Towards end-to-end benchmarking of llm-aided design verification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 1087–1095, 2026.

Ning Wang, Bingkun Yao, Jie Zhou, Xi Wang, Zhe Jiang, and Nan Guan. Large language model for verilog generation with golden code feedback. *arXiv preprint arXiv:2407.18271*, 2024a.

Ning Wang, Bingkun Yao, Jie Zhou, Yuchen Hu, Xi Wang, Nan Guan, and Zhe Jiang. Insights from verification: Training a verilog generation llm with reinforcement learning with testbench feedback. *arXiv preprint arXiv:2504.15804*, 2025a.

Ning Wang, Bingkun Yao, Jie Zhou, Yuchen Hu, Xi Wang, Zhe Jiang, and Nan Guan. Large language model for verilog generation with code-structure-guided reinforcement learning. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 164–170. IEEE, 2025b.

Ning Wang, Bingkun Yao, Jie Zhou, Yuchen Hu, Xi Wang, Zhe Jiang, and Nan Guan. Veridebug: A unified llm for verilog debugging via contrastive embedding and guided correction. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 61–67. IEEE, 2025c.

Xi Wang, Gwok-Waa Wan, Sam-Zaak Wong, Layton Zhang, Tianyang Liu, Qi Tian, and Jianmin Ye. Chatcpu: An agile cpu design and verification platform with llm. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024b.

Peiyang Wu, Nan Guo, Junliang Lv, Xiao Xiao, and Xiaochun Ye. Rtlrepocoder: Repository-level rtl code completion through the combination of fine-tuning and retrieval augmentation. *arXiv preprint arXiv:2504.08862*, 2025. URL https://arxiv.org/abs/2504.08862.

Changwen Xing, Yanfeng Lu, Lei Qi, Chenxu Niu, Jie Li, Xi Wang, Yong Chen, and Jun Yang. Rtlocating: Intent-aware rtl localization for hardware design iteration, 2026a. URL https://arxiv.org/abs/2603.00434v1.

Changwen Xing, SamZaak Wong, Xinlai Wan, Yanfeng Lu, Mengli Zhang, Zebin Ma, Lei Qi, Zhengxiong Li, Nan Guan, Zhe Jiang, et al. Chipmind: Retrieval-augmented reasoning for long-context circuit design specifications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 1337–1345, 2026b.

Ke Xu, Jialin Sun, Yuchen Hu, Xinwei Fang, Weiwei Shan, Xi Wang, and Zhe Jiang. Meic: Re-thinking rtl debug automation using llms. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2024.

Guang Yang, Wei Zheng, Dong Liang, Peng Hu, Yukui Yang, Shaohua Peng, Zhenghan Li, Jiahui Feng, Xiao Wei, Kexin Sun, et al. Large language model for verilog code generation: Literature review and the road ahead. *ACM Computing Surveys*, 2025.

Bingkun Yao, Ning Wang, Jie Zhou, Xi Wang, Hong Gao, Zhe Jiang, and Nan Guan. Location is key: Leveraging llm for functional bug localization in verilog design. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7. IEEE, 2025.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic data sets, support qualitative and thematic data analysis, clean and reformat datasets, and implement methods. We have not used generative AI tools to develop theoretical models or conceptual frameworks or to formulate mathematical claims. The remaining required disclosure tasks are not applicable to this work.

Additionally, we used generative AI tools to edit a research paper for readability, create artifacts, propose a title and keywords. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sample dataset is uploaded in supplementary material for qualitative review. Full dataset will be uploaded to HuggingFace when this work is accepted.

ETHICS STATEMENT

This work introduces a benchmark and a probabilistic embedding method for hardware-design automation, constructed entirely from publicly available open-source RTL repositories and their commit histories. The benchmark involves no human subjects, personally identifiable information, or

proprietary design data. The primary risk we identify is that LLM-generated queries may contain inaccuracies; making this uncertainty explicit is a central objective of our method. We foresee no other direct negative societal impact, and all open-source assets are used in accordance with their licenses.