SecV: LLM-based Secure Verilog Generation with Clue-Guided Exploration on Hardware-CWE Knowledge Graph

Fanghao Fan[1] , **Yingjie Xia**[1,2*] , **Li Kuang**[3*]
[1]Micro-Electronics Research Institute, Hangzhou Dianzi University
[2]College of Computer Science and Technology, Zhejiang University
[3]School of Computer Science and Engineering, Central South University
fhfan@hdu.edu.cn, xiayingjie@zju.edu.cn, kuangli@csu.edu.cn

Abstract

Verilog is specified as the primary Register Transfer Level (RTL) hardware description language, which designs the logical functions between registers for digital circuit systems. Recently, there emerges much cutting-edge research in leveraging Large Language Models (LLMs) to generate Verilog, aiming at effectively reducing errors and costs in the logic design of chips. However, these works mainly focus on logical correctness or PPA (Power, Performance, Area) measurement of the generated results, while neglecting the security problems in Verilog. In this study, we propose SecV, a novel and unified framework to generate secure Verilog by clue-guided exploration on Common Weakness Enumeration (CWE) knowledge graph (KG) for chips. First, the builder of the KG utilizes the instance-adapted chain of thought (COT) to extract entities and their relationships from raw Hardware-CWE corpora. Then, a fine-tuned BERT model is employed to verify the Hardware-CWE KG and collaborate with builder iteratively to achieve the precise KG. Based on Hardware-CWE KG, a clue-guided graph exploration paradigm is designed to facilitate collaborative inference of knowledge to generate secure Verilog by LLMs. Experiments demonstrate that SecV achieves 82.6% secure Verilog code without specified CWE in the generated functionally correct Verilog, with superior performance of a 21.7% performance improvement compared to SOTA.

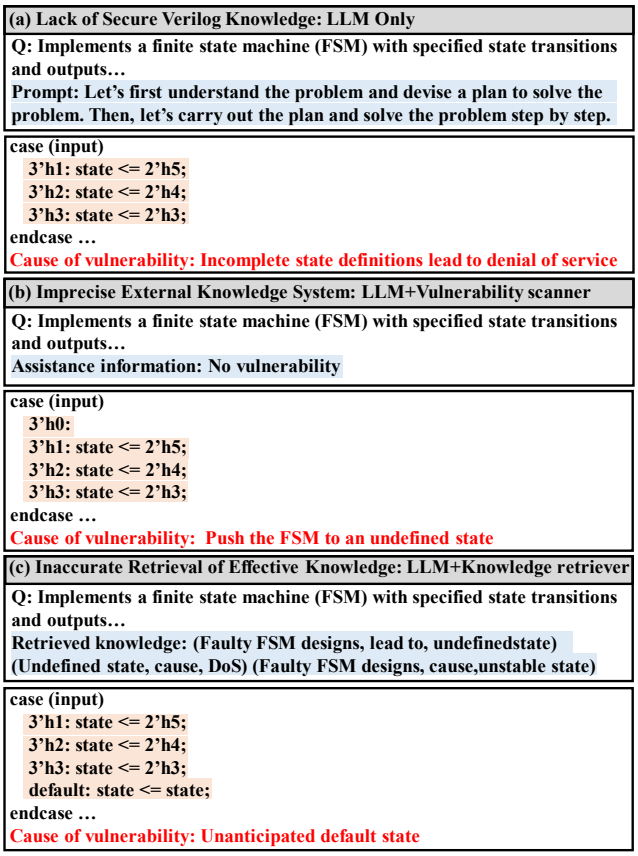

Figure 1: Existing challenges need to be resolved

1 Introduction

Modern integrated circuits (IC) design requires engineers to utilize Register Transfer Level (RTL) languages, such as Verilog, to specify hardware architectures and define logical functions between registers in digital circuit systems. As the fast development of Artificial Intelligence (AI) techniques, especially the emergence of the cutting-edge technology Large Language Models (LLMs) [Achiam *et al.*, 2023; Jiang *et al.*, 2023; Dubey *et al.*, 2024], AI-assisted Verilog development becomes a promising solution to improve efficiency and reduce code errors for chip design. Motivated by these advancements, multiple Electronic Design Automation (EDA) tools have been released to actively explore the automatic generation of Verilog code through domain-specific, fine-tuned LLMs [Liu *et al.*, 2023; Thakur *et al.*, 2024] and multi-agent collaboration approaches [Ho *et al.*, 2024; Zhao *et al.*, 2024c]. These works mainly focus on enhancing development efficiency and logical correctness to generate Verilog code, or improving the measurement of the chip design in power, performance and area (PPA), while the generated code inevitably encounters certain types of security

*Corresponding author

vulnerabilities due to the lack of secure Verilog knowledge in pre-training [Fu *et al.*, 2023].

In order to utilize LLMs to generate functionally correct and secure Verilog code, some challenges are summarized in Figure 1.

Challenge 1: Lack of Secure Verilog Knowledge in pre-trained LLMs The general LLMs are pre-trained by the datasets which are lack of secure Verilog code [Nazzal *et al.*, 2024]. By fine-tuning models [Liu *et al.*, 2024; Zhao *et al.*, 2024b] or designing prompts [Ahmad *et al.*, 2024; Lu *et al.*, 2024; Pei *et al.*, 2024], it is still difficult to construct enough Verilog coding capability of LLMs to avoid security vulnerabilities.

Challenge 2: Imprecise External Knowledge System to assist LLMs in secure Verilog generation There are some efficient techniques to enhance the capability of LLMs in generating secure Verilog, such as Retrieval-Augmented Generation (RAG) [Tsai *et al.*, 2024] and Chain of Thought (CoT) [Zhao *et al.*, 2024a]. All of these methods require external domain knowledge systems to compensate LLMs for its lack of pre-trained data. However, how to accurately collect and effectively organize domain-related data to construct a precise external knowledge system is a key challenge for the domain-specific generation of secure Verilog.

Challenge 3: Inaccurate Retrieval of Effective Knowledge for LLMs to generate secure Verilog code Although some external knowledge systems, such as the knowledge graph (KG), imported to augment LLMs are promising solutions for the issue of a lack of secure Verilog knowledge, the generated results depend on the effective retrieval of knowledge from the KG based on the task requirements [Kim *et al.*, 2023; Huang *et al.*, 2023]. Therefore, designing an accurate and cost-effective KG retrieval algorithm is a key component for the efficient utilization of both the KG and LLMs.

In this study, we propose a unified framework to generate functionally correct and secure Verilog code by LLMs, named SecV. The framework consists of three components, a hardware Common Weakness Enumeration (CWE) knowledge graph, a verifier model to iteratively optimize the precision of the KG, and a clue-guided graph exploration paradigm to retrieve accurate knowledge for LLMs. Specifically, SecV-builder firstly utilizes the instance-adapted COT to extract entities and their relationships from raw Hardware-CWE corpora to construct the domain-specific KG. Then, a fine-tuned BERT model is employed as SecV-verifier to verify the Hardware-CWE KG and collaborate with SecV-builder iteratively to improve the KG more precise. Based on the Hardware-CWE KG, SecV-retriever is designed by a clue-guided graph exploration paradigm to facilitate collaborative inference of knowledge for LLMs to generate secure Verilog.

The contributions of SecV are summarized as follows:

- SecV-builder extracts domain-specific corpora and applies instance-adapted CoT prompts to construct triples of the Hardware-CWE KG, which compensates the lack of secure Verilog knowledge for pre-trained LLMs solving the challenge 1.
- SecV-verifier utilizes the BERT model to remove error triples and iteratively collaborates with SecV-builder to optimize precision of the Hardware-CWE KG solving the challenge 2.
- SecV-retriever leverages a clue-guided graph exploration paradigm on the constructed Hardware-CWE KG to accurately retrieve effective knowledge for LLMs solving the challenge 3.

2 Related Work

Chip-Chat [Blocklove *et al.*, 2023] represents a pioneering effort to fully automate the hardware design workflow, from initial design to tape-out, by leveraging Large Language Models (LLMs) like ChatGPT [Achiam *et al.*, 2023] throughout the entire process. Since the advent of LLMs, the field of Verilog generation has experienced significant advancements. Given the stringent quality requirements in chip design, recent efforts have increasingly focused on enhancing the quality of LLM-generated Verilog. Previous approaches include training or fine-tuning code-oriented LLMs with Verilog-specific domain knowledge [Wu *et al.*, 2024; Fang *et al.*, 2024] and introducing additional stages in the code generation process [Blocklove *et al.*, 2024; Sami *et al.*, 2024b], such as planning, verification, and refinement based on simulation feedback.

Despite these approaches improving the syntactic correctness of the generated results, the process involves generating synthesizable code with frequent context switching between different subtasks. To address this, Aivril [Sami *et al.*, 2024a] implements a basic dual-agent system for code generation and review, while VerilogCoder [Ho *et al.*, 2024] employs a high-level planning agent to assign tasks to other agents with independent dialogue histories. This enables specialized task handling and greater modularity.

Although these methods have significantly improved the logical correctness and PPA (Power, Performance, Area) metrics of the generated results, they have largely overlooked the security aspects of the generated Verilog code. Some studies have explored manual template reviews [Pearce *et al.*, 2023] and AST-based scanner [Ahmad *et al.*, 2022] to identify vulnerabilities in Verilog code. However, these methods suffer from incomplete vulnerability coverage and inefficiencies, highlighting the need for a unified framework that enables LLMs to generate secure Verilog code effectively.

3 Methodology

3.1 Framework overview

We have developed a novel generation framework called SecV that leverages the Hardware-CWE knowledge graph (KG) to enhance Large Language Models (LLMs), enabling them to function as skilled designers for secure Verilog code. The framework begins with raw Hardware-CWE corpora, from which the SecV-builder uses a CWE-domain searcher to retrieve relevant content and applies instance-adaptive Chain-of-Thought (COT) to address the lack of trusted hardware triples, thereby constructing the Hardware-CWE KG. The SecV-verifier then collaborates iteratively with the SecV builder to refine and ensure the precision of the KG. Finally,

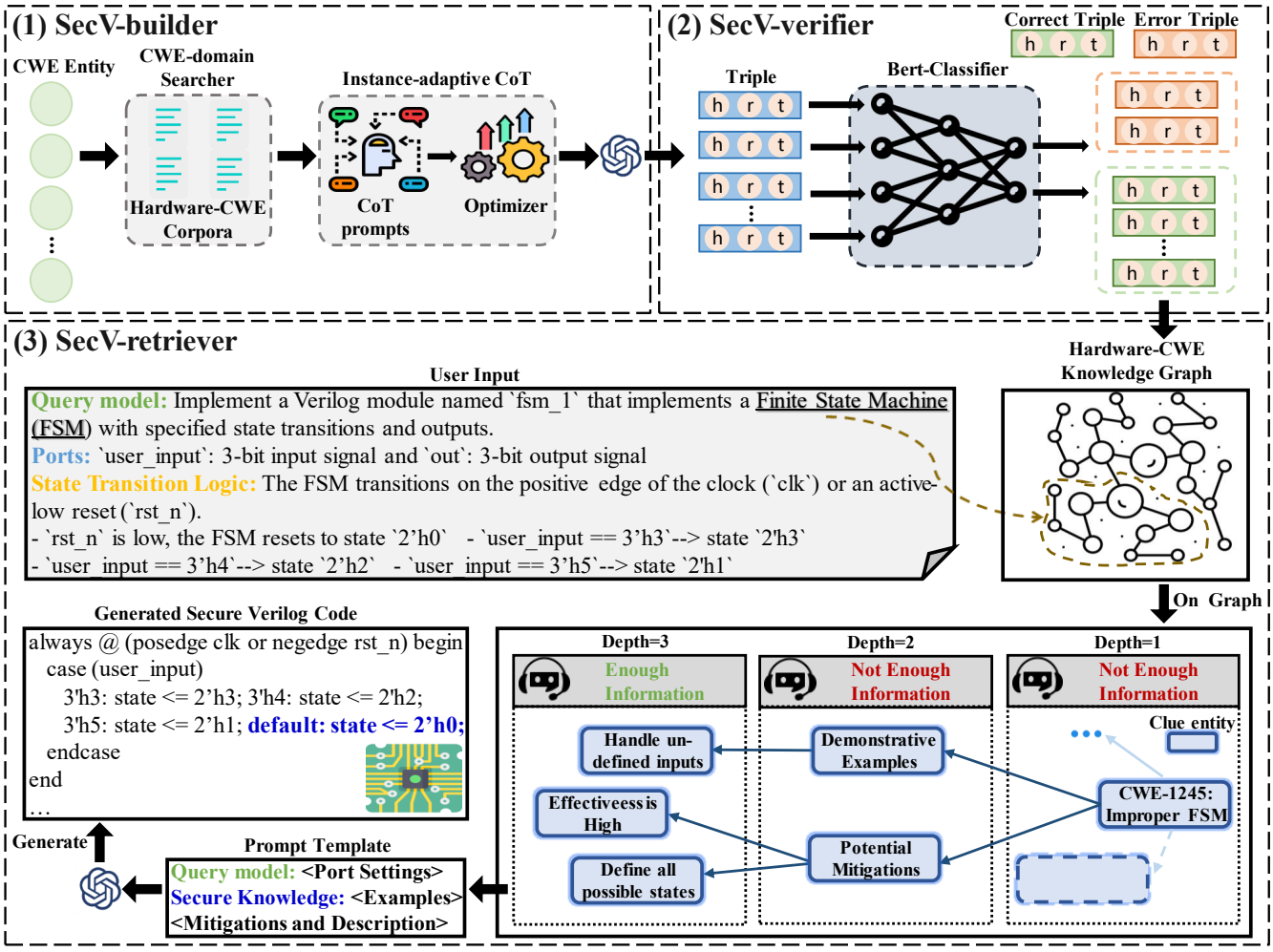

Figure 2: An overview of SecV. SecV integrates the SecV-buider, SecV-verifier, and SecV-retriever into a unified framework for generating secure Verilog.

a clue-guided graph exploration paradigm is employed on the KG to facilitate collaborative inference, enabling LLMs to generate secure and functionally correct chip modules based on the provided descriptions. SecV integrates the SecV-builder, SecV-verifier, and SecV-retriever into a unified framework for generating secure Verilog. An overview of SecV is shown in Figure 2.

3.2 SecV-builder

For CWE entities, it refers to a selected set of vulnerability terms within the chip security domain. The SecV-builder employs a CWE-domain searcher to obtain the most relevant context from raw Hardware-CWE corpora and leverages instance-adapted Chain-of-Thought to extract entities and their relationships, thereby constructing a Hardware-CWE KG.

CWE-domain searcher LLMs are frequently constrained by significant knowledge hallucinations, where the content they produce often diverges from factual knowledge. The issue impacts the precision and practical applications of the Hardware-CWE KG. To mitigate knowledge hallucinations and facilitate accurate knowledge augmentation for LLMs, we propose a CWE-domain searcher. For a given CWE entity, the searcher first segments the Hardware-CWE corpora into individual sentences. It then ranks the relevant sentences based on the frequency of occurrence of the vulnerability entity and their distributional similarity to the corpora. Finally, these sentences are concatenated into a single text to serve as input for the LLMs.

Instance-adaptive COT When the input consists solely of Hardware-CWE content and simple instructions, the output generated by LLMs is often difficult to control and may even contain incorrectly formatted triples. To address this challenge, we propose an instance-adaptive chain-of-thought (CoT) approach that selects appropriate CoT prompts in a discrete space for each CWE instance in the current reasoning task. The instance-adaptive CoT method decomposes the complex extraction task into multiple simpler steps, thereby avoiding the misleading effects of untrustworthy triples and enabling the sequential extraction of CWE entities and their relationships. For standard CoT prompting, given the task T, optimizer L, CoT specific instructions $\mathcal{P}$ like zero-shot COT

or plan-and-solve COT, we formalize this simple yet fundamental solving paradigm as:

$$L(A|P,T) = L(A|P,T,C)L(C|P,T) \tag{1}$$

, where C denotes a sampled rationale in natural language and A is the generated answer.

To generate diverse prompts, we initialize two CoT prompts, P_1 and P_2. In each iteration, we employ LLMs as optimizers and randomly select a pair of prompts P_i and P_j from the prompts pool to perform a fusion operation. Which is defined as:

$$P_f = \text{Fusion}(P_i, P_j) \tag{2}$$

Then, we enable mutation on the fusion CoT prompting P_f, which is defined as:

$$P_m = \text{Mutation}(P_f) \tag{3}$$

Ultimately, LLMs select the most appropriate CoT prompt from the generated set to extract CWE entities and their relationships, thereby constructing the Hardware-CWE KG.

3.3 SecV-verifier

Although the SecV-builder helps improve the quality of the LLM's output, errors still persist in the generated triples due to knowledge noise. To further improve the precision of the Hardware-CWE KG, we introduce the SecV-verifier, which is responsible for identifying and filtering erroneous or duplicate triples generated by the LLMs. Specifically, the SecV-verifier consists of two steps: error detection and correction.

For error detection, we propose a BERT-based binary classifier model, fine-tuned with the open source MITRE documentation [MITRE, 2023]. The classifier's input consists of triples from the Hardware-CWE KG, and its output is either "correct" or "error", indicating whether the triples need correction. To train the pruner, we gather training data from MITRE, selecting manually verified triples as the "correct" category. We also collect an equivalent subset of erroneous triples to represent the "error" category. These error triples are classified into the following types: **(i) Format Error:** The structure of the triple does not conform to the canonical format, resulting in incomplete information. **(ii) Conflict Error:** The relationship between the head and tail entities is expressed imprecisely or ambiguously. **(iii) Semantic Error:** A semantic inconsistency exists between the head entity, the tail entity, and the relationship predicate within the triple. We use "correct" and "error" triples as input, with the corresponding labels "correct" or "error" as output targets during fine-tuning. We then employ the BERT classifier to check the format, conflicts, and semantics of the triples from the Hardware-CWE KG, enabling error correction based on the results of these checks.

For error correction, the triples labeled "correct" remain unchanged, while those labeled "error" are combined with the content retrieved by the CWE-domain searcher, the triples, and the error hint message into a template. This enables us to re-prompt the SecV-builder of the KG to correct the triples. Subsequently, we obtain a precise Hardware-CWE KG through the iterative collaboration of the SecV-verifier and the SecV-builder.

3.4 SecV-retriever

After building and verifying the hardware-CWE KG, we designed a SecV-retriever that utilizes a clue-guided exploration paradigm, enabling collaborative inference by LLMs to generate secure Verilog. The paradigm allows the LLM to dynamically explore the multiple evidence subgraphs in the Hardware-CWE KG and make decisions based on this exploration. Given an input description, the SecV-retriever process starts with the identification of initial clue entities. It then uses these clue entities as starting points for exploration to construct an evidence subgraph through LLM reasoning and graph retrieval. Finally, LLMs utilize the security knowledge in the evidence subgraph to generate secure Verilog code.

Clues Recognition We first use LLM to identify key entities from the query module description Q. Specifically, we use a prompt that consists of three parts: the question to be analyzed, the instruction phrase, and the clue list. We then apply cosine similarity to match the clue entities with CWE keywords. Then, we encode all the clue entities e extracted by the LLM and all the entities G from the Hardware-CWE KG into dense embeddings V_e and V_g respectively, and compute the cosine similarity matrix between them. Finally, we obtain the clue entity sets M_q with the highest similarity scores, which we use to build the evidence subgraphs in the next step.

Clue-guided graph Exploration We define the Hardware-CWE KG by $\mathcal{G} = \{(h,r,t) \mid h \in \psi, r \in \varphi, t \in \upsilon\}$, where ψ, φ and υ represent the entity set, relation set and tail entity set, respectively. The objective of this stage is to build the evidence sub-graphs $\mathcal{S}$ based on the extracted clue entites e. An evidence sub-graph is defned by $\mathcal{S} = \{(e,r,e^*) \mid e \in \psi, r \in \varphi, e^* \in \upsilon\}$. As shown in Algorithm 1, clue-guided exploration adds more query-relevant security knowledge to evidence subgraphs by expanding each clue e in M_q by 1-hop to its neighbors e^* , and adding triples (e,r,e^*) to evidence subgraphs S in the exploration process. Specifically, before exploring the next depth layer, we update M_q by removing

Algorithm 1 Clue-guided graph exploration Algorithm

Input: Query Q, Knowledge graph $\mathcal{G}$,Clue set M_q
Parameter: Depth limit D_{max}, LLM π
Output: Evidence subgraphs $\mathcal{S}$
Initialize Neighbors set $\mathcal{N} \leftarrow \emptyset$
while $D < D_{\text{max}}$ **do**
  for each clue $e \in M_q$ **do**
    Expand e by 1-hop to its neighbors e^*
    Add triples (e,r,e^*) to $\mathcal{S}$
    Update $\mathcal{N} \leftarrow e^*$
  end for
  $M_q \leftarrow Select(\mathcal{N}, \pi)$
  $\mathcal{S} \leftarrow Prune(\mathcal{S}, \pi)$
  if (Q, $\mathcal{S}$, π) **then**
    break
  end if
  Increment D by 1.
end while
Return: $\mathcal{S}$

Design		Llama-3		Llama-3 + SecV		GPT-4		GPT-4 + SecV		Codestral		Codestral + SecV		RTLCoder		RTLCoder + SecV	
		Func.	Sec.	Func.	Sec.	Func.	Sec.	Func.	Sec.	Func.	Sec.	Func.	Sec.	Func.	Sec.	Func.	Sec.
Privilege and Access Control Issues	Foo_bar_1	1	✓	5	✓	5	✓	5	✓	5	✓	5	✓	0	-	5	✓
	Foo_bar_2	1	✗	2	✓	4	✓	3	✓	5	✓	0	-	5	✗	5	✗
	Bootrom	0	-	0	-	0	-	1	✓	0	-	5	✓	0	-	5	✓
	Mpec	0	-	0	-	1	✗	1	✓	0	-	0	-	0	-	0	-
General Circuit and Logic Design Issues	Locked_1	0	-	0	-	3	✗	4	✗	5	✗	5	✗	0	-	0	-
	Locked_2	0	-	0	-	4	✗	4	✓	5	✗	0	-	0	-	0	-
	Fsm_1	5	✓	5	✓	5	✓	5	✓	0	-	0	-	0	-	0	-
	Fsm_2	0	-	0	-	0	-	1	✗	5	✗	5	✗	5	✗	5	✗
	Shifter	5	✓	4	✓	5	✓	5	✗	5	✓	5	✓	5	✓	5	✓
Memory and Storage Issues	Regfile_1	5	✗	2	✗	4	✓	5	✓	0	-	5	✓	5	✗	5	✓
	Regfile_2	3	✗	1	✗	4	✓	4	✓	5	✓	5	✓	5	✓	5	✓
	Ase_1	1	✗	2	✓	5	✓	5	✓	0	-	0	-	0	-	0	-
	Ase_2	5	✓	5	✓	4	✓	5	✓	0	-	5	✗	5	✓	5	✓
	Register	5	✓	5	✓	5	✗	5	✓	0	-	5	✓	5	✗	5	✗
	Sha256_Register	3	✓	4	✓	4	✓	4	✓	0	-	0	-	5	✓	0	-
Power, Clock, Thermal, and Reset Issues	Jtag_1	1	✗	1	✗	3	✓	5	✓	5	✓	0	-	0	-	0	-
	Jtag_2	5	✓	5	✓	4	✓	5	✓	5	✗	5	✗	5	✓	5	✓
	Rglk	5	✓	3	✓	3	✓	3	✓	0	-	0	-	5	✓	5	✓
	Csr	5	✗	5	✓	2	✗	3	✗	5	✓	5	✓	5	✓	5	✓
Primitives and Cryptography Issues	Axi	1	✗	5	✓	2	✓	4	✓	0	-	0	-	0	-	5	✓
	Acct	2	✗	4	✗	2	✗	4	✓	0	-	5	✓	5	✗	5	✓
	Mod_exp	0	-	0	-	1	✗	3	✓	0	-	5	✓	0	-	0	-
	GlitchEx	1	✓	3	✓	4	✓	3	✓	5	✓	0	-	5	✓	5	✓
Success rate		47.0%	39.1%	53.0%	56.5%	64.3%	60.9%	75.7%	82.6%	47.8%	30.4%	56.5%	39.1%	56.5%	34.8%	65.2%	52.2%

Table 1: The Functional Correctness and Security Evaluation for Different LLMs

clues e from the previous depth layer and adding neighbors e^*, which are deemed new clues to the question by the LLM. e^* is then used as the starting node for the next depth exploration, continuing the expansion of the evidence subgraphs S. Additionally, to manage information overhead and maintain diversity, we prune the tail entities of the evidence subgraphs by leveraging the LLM. These pruning steps result in the final evidence graph S, optimizing information while preserving diversity.

Verilog generation Upon obtaining the current evidence subgraphs S through the clue-guided exploration process, we use the LLM to assess whether the evidence graph is sufficient for secure Verilog generation. If the evaluation is positive, we extract secure information, including potential mitigations and examples. We then generate secure Verilog code by prompting the LLM with a template consisting of four components: a system instruction, a module description, mitigations, and examples. Conversely, if the evaluation is negative, we repeat the clue-guided exploration steps until the evaluation becomes positive or D_{max} is reached. If the algorithm has not yet concluded, it indicates that even after reaching the maximum depth, we are still unable to explore the evidence subgraphs to resolve the question. In this case, the LLM generates Verilog code exclusively based on its inherent knowledge.

4 Experiments

4.1 Experimental Setup

Settings Unlike general-purpose code, the behavior of Verilog is typically evaluated through testbenches. We built and installed the open-source Icarus Verilog [Williams, 2023] simulator in a Python environment. Icarus Verilog is intended to compile all of the Verilog, as described in the IEEE-1364 standard. Simulation and testing are handled within the built environment, and results can be produced using just a single line of command.

Datasets We have collected five categories of vulnerable designs from the latest version (V4.16) of the Common Weakness Enumeration (CWE) dataset. These categories include privilege and access control, general circuit and logic design, memory and storage, power clock thermal and reset, and primitives and cryptography. In total, there are 23 instances of varying sizes and complexities. For each design, we provide the following information in three separate files:

- **Description (Query.txt):** This description includes an explicit indication of the module name, all input and output (I/O) signals with signal name and width.
- **Testbench (Testbench.sv):** A testbench with multiple test cases, each with input values and correct output values.
- **Secure Design (Reference_Verilog.v):** A secure Verilog code reference provided by MITRE [MITRE, 2023]. We can evaluate the security of the automatically generated Verilog.

Metrics We follow recent works in directly measuring code security rate through success rate metric [Pinckney *et al.*, 2024; Thakur *et al.*, 2023], where a problem is considered solved if any of the k samples pass the unit tests. Specifically, The success rate for functional correctness is defined as follow:

$$\text{Success rate}(Func) = \frac{1}{n \cdot k} \sum_{i=1}^{n} \mathbb{I}_{(\text{func}_i \leq k)} \tag{4}$$

, where n denotes the total number of cases and $\mathbb{I}_{(func)}$ is the indicator function that indicates whether the functional test was passed in k samples. We counted the number of designs with correct functionality generated in five trials.

In the security section, the success rate for security is defined as follow:

$$\text{Success rate}(Sec) = \frac{1}{n}\sum_{i=1}^{n}\left[\bigvee_{j=1}^{k}\mathbb{I}_{(\text{sec}_j \leq k_i)}\right] \quad (5)$$

, where k_i denotes the number of samples that pass the functional tests, $\bigvee$ is OR operation, and $\mathbb{I}_{(sec)}$ is the indicator function that indicates whether a functionally correct sample passes the security test. We evaluate the designs by examining the presence of functionally correct Verilog within k samples that also satisfy security requirements. The assessment outcomes are denoted as follows: ✓ indicates the successful identification of security-compliant code, ✗ represents the absence of secure implementations despite functional correctness, and $-$ denotes the non-existence of functionally valid code.

Compared cases To evaluate the performance of various LLMs from different perspectives, we setup temperature = 0.7 and top p = 0.9 for all LLMs. We compare SecV with four LLMs as follows in our experiments:

- LLama-3: the open-source commercial solution.
- GPT-4.0: the state-of-the-art commercial solution.
- Codestral: A recent model for code generation with 8 billion parameters, based on the Mamba architecture [Dao and Gu, 2024].
- RTLCoder: An academic model with 6.7 billion parameters, developed by fine-tuning the Deepseek-Coder [Guo *et al.*, 2024a] model using Verilog data.

4.2 Main Results

Table 1 shows the pass rates of LLM + SecV and all compared LLM methods on the specified CWE benchmarks. Overall, LLM + SecV achieved significant improvements over the compared methods while maintaining a remarkable balance between functional correctness and security rate. First, the LLM + SecV approach outperforms all other LLM methods in terms of security rates. Specifically, we observe a 17.4%-21.7% improvement over two commercial solutions, Llama3 and GPT-4, and an 8.7%-17.4% improvement over two academic models, Codestral and RTLCoder. Notably, the highest gain is achieved with the most state-of-the-art (SOTA) commercial solution, GPT-4, which improves by 21.7% and attains a 82.6% security rate in functionally correct Verilog code. These results suggest that SecV's performance improves with the reasoning power of the backbone model and that the potential of SecV-KG can be further harnessed by employing more powerful LLMs.

Additionally, with regard to functional correctness, the LLM + SecV method shows notable improvements. Specifically, we observe a 6.0%-11.4% improvement over two commercial solutions, Llama3 and GPT-4, and an 8.7% improvement over both academic models, Codestral and RTLCoder. The performance degradation of LLM + SecV in a very few test cases can be attributed to the secure knowledge provided, which may conflict with the intrinsic knowledge of the LLM, thereby undermining the model's original reasoning capability. Thus, the misalignment between the LLMs and secure knowledge poses a challenge in demonstrating the benefits of our LLM + SecV.

Backbone	Method	Func(%)	Sec(%)
GPT-4	$\text{SecV}_{\text{w/o searcher}}$	66.1	65.2
	$\text{SecV}_{\text{w/o cot}}$	71.3	73.9
	$\text{SecV}_{\text{w/o SecV-vefier}}$	72.2	78.3
	$\text{SecV}_{\text{w/o SecV-retriever}}$	67.8	69.5
	SecV	**75.7**	**82.6**
LLama-3	$\text{SecV}_{\text{w/o searcher}}$	45.2	39.1
	$\text{SecV}_{\text{w/o cot}}$	49.6	52.2
	$\text{SecV}_{\text{w/o SecV-vefier}}$	51.3	52.2
	$\text{SecV}_{\text{w/o SecV-retriever}}$	48.7	47.8
	SecV	**53.0**	**56.5**

Table 2: Ablation study for SecV in the speiced specified CWE-benchmark

4.3 Ablation Studies

To further investigate the contribution of each component within SecV to the secure Verilog generation, we performed a series of ablation experiments on the entire framework. Specifically, we denote SecV without the instance-adapted CoT as $\text{SecV}_{w/o\ cot}$, SecV without the CWE-domain searcher as $\text{SecV}_{w/o\ searcher}$, SecV without the SecV-verifier as $\text{SecV}_{w/o\ verifier}$, and SecV without the SecV-retriever as $_{w/o\ retriever}$, respectively. As shown in Table 2, the absence of any component within SecV leads to a degradation in the framework's overall performance. Notably, the SecV-retriever and the CWE-domain searcher have a more pronounced impact on SecV's performance. These two components control the source and utilization of knowledge, respectively. This underscores the importance of enhancing the precision of knowledge when using the Hardware-CWE knowledge graph to improve the ability of LLMs to generate secure Verilog code.

4.4 Discussion

We conducted extensive experiments to verify the effectiveness of the SecV-retriever and SecV-builder. Additionally, we provide a detailed analysis of how the number of CoTs and the depth of graph exploration impact SecV's performance.

Comparison for Performance We compare SecV with other methods that use LLMs to construct the Hardware-CWE knowledge graph and generate Verilog. We select GPT-4, Llama3-70B-Instruct, and Qwen-7B-Instruct [Bai *et al.*, 2023] as backbone models to evaluate the impact of LLMs with varying performance levels. The overall results are presented in Table 3. On one hand, the SecV-builder achieves the best results across all backbone networks, with a 4.3% to 8.7% improvement in security rate and a 2.6% to 7.8% improvement in functional correctness compared to other methods. On the other hand, we observe that the improvement from the SecV-retriever is more pronounced with larger LLMs that have better performance. This suggests that our

Method	Func(%)	Sec(%)
GPT-4		
[Guo *et al.*, 2024b]	67.8	69.6
[Edge *et al.*, 2024]	70.4	73.9
+ SecV	75.7	82.6
Gain	(+4.3)	(+8.7)
LLama3-70B-Instruct		
[Guo *et al.*, 2024b]	66.1	60.9
[Edge *et al.*, 2024]	67.0	73.9
+ SecV	74.8	78.2
Gain	(+7.8)	(+4.3)
Qwen-7B-Instruct		
[Guo *et al.*, 2024b]	64.3	60.9
[Edge *et al.*, 2024]	67.8	69.6
+ SecV	70.4	73.9
Gain	(+2.6)	(+4.3)

Table 3: Comparison with different Graph Construction and Retrieval Augmented Generation methods

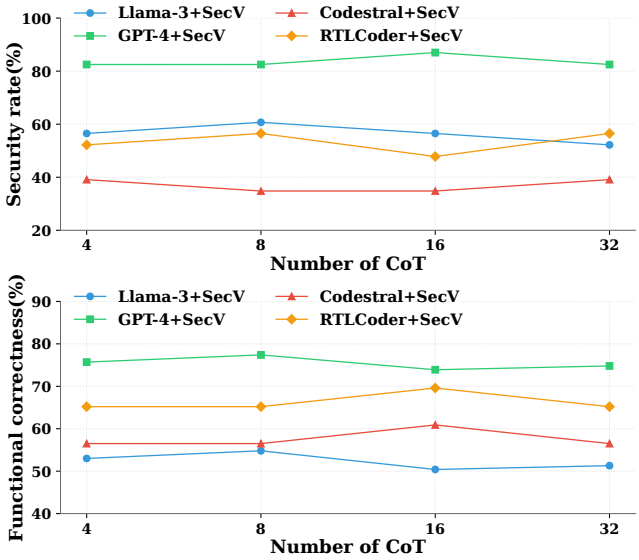

Figure 3: Effectiveness of different CoT number

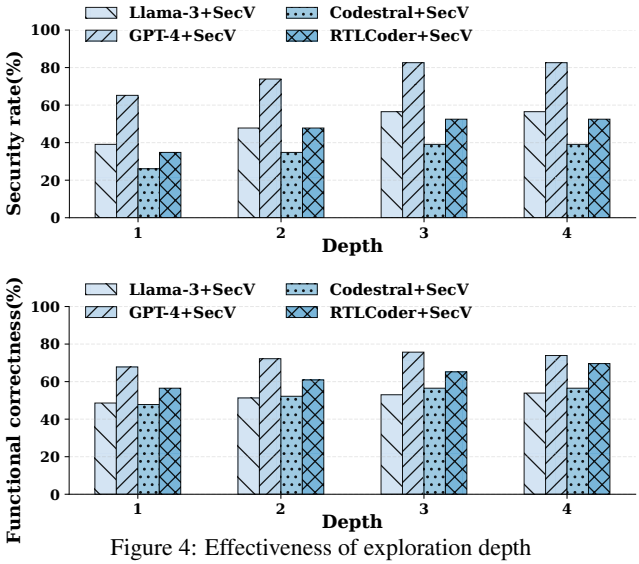

Figure 4: Effectiveness of exploration depth

method effectively leverages the powerful inference capabilities of LLMs to construct a precise Hardware-CWE knowledge graph. Notably, even for Qwen-7B-Instruct, a backbone network with relatively few parameters, our method achieves a 73.9% security rate.

Effectiveness of CoT Number In previous experiments, we strategically employed single-round fusion and mutation operations to optimize inference speed. In this section, our objective is to systematically verify the relationship between the number of CoT prompts generated during multiple rounds of fusion and mutation operations and the resulting performance of SecV. As shown in Figure 3, the results reveal a clear positive correlation: an increase in the number of CoT prompts leads to continuous improvements in SecV performance. However, the time required for inference increases proportionally with the number of generated CoT prompts. Therefore, in scenarios where inference speed is less critical, the SecV-builder can be used to significantly enhance performance. When inference time is a concern, optimal performance is typically achieved with 4 or 8 CoT prompts.

Effectiveness of exploration depth To investigate the effect of exploration depth D on SecV performance, we conducted experiments with depth settings ranging from 1 to 4. As shown in Figure 4, the performance of SecV improves as the exploration depth increases. However, due to computational cost considerations, which increase linearly with depth, we selected a depth of 3 as the default setting in previous experiments. Furthermore, performance gains diminish when the depth exceeds 3. This is mainly because only a small portion of questions related to secure Verilog generation require deeper reasoning depths, based on the number of relations in the Hardware-CWE knowledge graph.

5 Conclusions

In this work, we propose SecV, a novel and unified framework that efficiently constructs a precise Hardware-CWE knowledge graph (KG) and employs a clue-guided graph exploration paradigm to facilitate the collaborative inference of knowledge for LLMs to generate secure Verilog code. Specifically, the SecV framework consists of three components, SecV-builder, SecV-verifier, and SecV-retriever. They respectively take charge of the construction of Hardware-CWE KG to compensate the lack of secure Verilog knowledge for pre-trained LLMs, iterative removal of error triples to optimize the precision of the Hardware-CWE KG, and clue-guided accurate retrieval of effective knowledge on the KG for LLMs. Extensive experiments on CWE benchmarks demonstrate the effectiveness of SecV, achieving better performance with a security rate of 82.6% on the generated functionally correct Verilog code, compared with other SOTA. In the future, we will explore secure knowledge preference alignment for LLMs to enhance the beneficial effects of the Hardware-CWE KG, enabling LLMs to generate more secure Verilog.

Acknowledgments

This work has been supported by the National Natural Science Foundation of China under grant No.62472132 and 62472447, Key R&D Program Project of Zhejiang Province under grant No.2025C01063 and 2024C01179, Hunan Provincial Natural Science Foundation of China under grant No.2024JK2006, the Science and Technology Innovation Program of Hunan Province under grant No.2023RC1023.

References

[Achiam *et al.*, 2023] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[Ahmad *et al.*, 2022] Baleegh Ahmad, Wei-Kai Liu, Luca Collini, Hammond Pearce, Jason M Fung, Jonathan Valamehr, Mohammad Bidmeshki, Piotr Sapiecha, Steve Brown, Krishnendu Chakrabarty, et al. Don't cweat it: Toward cwe analysis techniques in early stages of hardware design. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[Ahmad *et al.*, 2024] Baleegh Ahmad, Shailja Thakur, Benjamin Tan, Ramesh Karri, and Hammond Pearce. On hardware security bug code fixes by prompting large language models. *IEEE Transactions on Information Forensics and Security*, 2024.

[Bai *et al.*, 2023] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[Blocklove *et al.*, 2023] Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. Chip-chat: Challenges and opportunities in conversational hardware design. In *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6. IEEE, 2023.

[Blocklove *et al.*, 2024] Jason Blocklove, Shailja Thakur, Benjamin Tan, Hammond Pearce, Siddharth Garg, and Ramesh Karri. Can eda tool feedback improve verilog generation by llms? *arXiv preprint arXiv:2411.11856*, 2024.

[Dao and Gu, 2024] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

[Dubey *et al.*, 2024] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[Edge *et al.*, 2024] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. April 2024.

[Fang *et al.*, 2024] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. Transferable pre-synthesis ppa estimation for rtl designs with data augmentation techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[Fu *et al.*, 2023] Weimin Fu, Kaichen Yang, Raj Gautam Dutta, Xiaolong Guo, and Gang Qu. Llm4sechw: Leveraging domain-specific large language model for hardware debugging. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6. IEEE, 2023.

[Guo *et al.*, 2024a] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

[Guo *et al.*, 2024b] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. 2024.

[Ho *et al.*, 2024] Chia-Tung Ho, Haoxing Ren, and Brucek Khailany. Verilogcoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool. *arXiv preprint arXiv:2408.08927*, 2024.

[Huang *et al.*, 2023] Yu-Xuan Huang, Zequn Sun, Guangyao Li, Xiaobin Tian, Wang-Zhou Dai, Wei Hu, Yuan Jiang, and Zhi-Hua Zhou. Enabling abductive learning to exploit knowledge graph. In *IJCAI*, pages 3839–3847, 2023.

[Jiang *et al.*, 2023] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[Kim *et al.*, 2023] Jiho Kim, Yeonsu Kwon, Yohan Jo, and Edward Choi. Kg-gpt: A general framework for reasoning on knowledge graphs using large language models. In *EMNLP (Findings)*, 2023.

[Liu *et al.*, 2023] Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. Chipnemo: Domain-adapted llms for chip design. *arXiv preprint arXiv:2311.00176*, 2023.

[Liu *et al.*, 2024] Shang Liu, Wenji Fang, Yao Lu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pages 1–5. IEEE, 2024.

[Lu *et al.*, 2024] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. Rtllm: An open-source benchmark for design rtl generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 722–727. IEEE, 2024.

[MITRE, 2023] MITRE. Common weakness enumeration, 2023. [Online; accessed 25-December-2024].

[Nazzal *et al.*, 2024] Mahmoud Nazzal, Issa Khalil, Abdallah Khreishah, and NhatHai Phan. Promsec: Prompt optimization for secure generation of functional source code with large language models (llms). In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2266–2280, 2024.

[Pearce *et al.*, 2023] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. Examining zero-shot vulnerability repair with large language models. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2339–2356. IEEE, 2023.

[Pei *et al.*, 2024] Zehua Pei, Hui-Ling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. Betterv: controlled verilog generation with discriminative guidance. In *Proceedings of the 41st International Conference on Machine Learning*, pages 40145–40153, 2024.

[Pinckney *et al.*, 2024] Nathaniel Pinckney, Christopher Batten, Mingjie Liu, Haoxing Ren, and Brucek Khailany. Revisiting verilogeval: Newer llms, in-context learning, and specification-to-rtl tasks. *arXiv preprint arXiv:2408.11053*, 2024.

[Sami *et al.*, 2024a] Humza Sami, Pierre-Emmanuel Gaillardon, Valerio Tenace, et al. Aivril: Ai-driven rtl generation with verification in-the-loop. *arXiv preprint arXiv:2409.11411*, 2024.

[Sami *et al.*, 2024b] Humza Sami, Pierre-Emmanuel Gaillardon, Valerio Tenace, et al. Eda-aware rtl generation with large language models. *arXiv preprint arXiv:2412.04485*, 2024.

[Thakur *et al.*, 2023] Shailja Thakur, Baleegh Ahmad, Zhenxing Fan, Hammond Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. Benchmarking large language models for automated verilog rtl code generation. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[Thakur *et al.*, 2024] Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

[Tsai *et al.*, 2024] YunDa Tsai, Mingjie Liu, and Haoxing Ren. Rtlfixer: Automatically fixing rtl syntax errors with large language model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

[Williams, 2023] Stephen Williams. The icarus verilog compilation system, 2023.

[Wu *et al.*, 2024] Peiyang Wu, Nan Guo, Xiao Xiao, Wenming Li, Xiaochun Ye, and Dongrui Fan. Itertl: An iterative framework for fine-tuning llms for rtl code generation. *arXiv preprint arXiv:2407.12022*, 2024.

[Zhao *et al.*, 2024a] Ruilin Zhao, Feng Zhao, Long Wang, Xianzhi Wang, and Guandong Xu. Kg-cot: Chain-of-thought prompting of large language models over knowledge graphs for knowledge-aware question answering. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*, pages 6642–6650. International Joint Conferences on Artificial Intelligence, 2024.

[Zhao *et al.*, 2024b] Yang Zhao, Di Huang, Chongxiao Li, Pengwei Jin, Ziyuan Nan, Tianyun Ma, Lei Qi, Yansong Pan, Zhenxing Zhang, Rui Zhang, et al. Codev: Empowering llms for verilog generation through multi-level summarization. *arXiv preprint arXiv:2407.10424*, 2024.

[Zhao *et al.*, 2024c] Yujie Zhao, Hejia Zhang, Hanxian Huang, Zhongming Yu, and Jishen Zhao. Mage: A multi-agent engine for automated rtl code generation. *arXiv preprint arXiv:2412.07822*, 2024.