arXiv:2603.17208v1 [cs.CL] 17 Mar 2026

SYMDIREC：一种用于增强寄存器传输级综合与摘要生成的神经符号分解-检索-征服框架

Prashanth Vijayaraghavan、Apoorva Nitsure、Luyao Shi、Charles Mackin、Ashutosh Jadhav、David Beymer、Ehsan Degan、Vandana Mukherjee

IBM Research，圣何塞，加利福尼亚州，美国
{prashanthv,apoorva.nitsure,luyao.shi,charles.mackin}@ibm.com
{ashutosh,beymer,edehgha,vandana}@us.ibm.com

摘要

寄存器传输级（RTL）综合与摘要生成是硬件设计自动化的核心，但由于 HDL 语法严格、监督信号有限，以及与自然语言的对齐较弱，大语言模型（LLM）在处理这类任务时仍面临挑战。现有的提示和检索增强生成（RAG）方法尚未纳入符号规划，因此限制了其结构化精确性。我们提出 SYMDIREC[1]，，一种神经符号框架，将RTL任务分解为符号子目标，通过微调后的检索器检索相关代码，并借助大语言模型推理组装经过验证的输出。SYMDIREC 无需对大语言模型进行微调，同时支持 Verilog 和 VHDL；与提示和 RAG 基线方法相比，该方法在综合任务上的 Pass@1 比率提升 ~20%，在摘要生成任务上的 ROUGE-L 提升 15–20%，展示了符号引导在 RTL任务中的优势。

1 引言

寄存器传输级（RTL）综合与摘要生成是电子设计自动化（EDA）中的核心任务。RTL 综合将高级自然语言规范转换为可综合的硬件模块，而 RTL 摘要则为现有硬件代码生成简洁的自然语言说明。例如，给定自然语言（NL）规范“构建一个 8 位波纹进位加法器”，系统必须生成正确的 Verilog/VHDL 模块，由全加器构成并传递进位；反之，给定这样的模块，系统应说明其功能结构（例如，最低有效位/最高有效位加法器和进位逻辑）。虽然这个例子相对简单，但现实中的 RTL 设计通常涉及多级流水线、控制逻辑，以及具有复杂时序和数据依赖的层次化模块。这些特征使单体式生成变得脆弱且容易出错，因此需要进行模块化分解、定向检索可复用的 RTL 组件，并进行明确的集成与验证。如图 1所示，综合和摘要生成都要求严格遵循 HDL 语法、模块化结构和精确的功能语义，因此有别于通用代码生成与摘要生成。

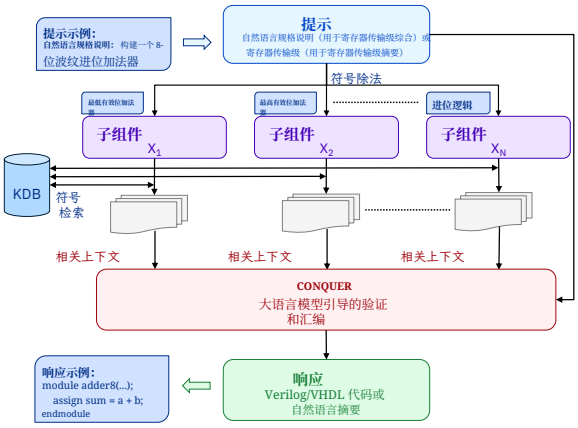


图 1：SYMDIREC 寄存器传输级综合与摘要生成框架概览。

尽管大语言模型（LLM）在代码生成方面展现出越来越大的潜力，但由于语法严谨、标注数据稀疏，且语义与自然语言存在差异，它们在 Verilog 和 VHDL 等硬件描述语言上的表现仍然有限（Vijayaraghavan 等，2024b；Zhao 等，2025）。近期的提示策略，如思维链（CoT）（Wei 等，2022）、CoDes（Vijayaraghavan 等，2024a）和 ReAct（Yao 等，2023），增强了逐步推理能力，但由于领域不匹配且缺乏结构先验，在寄存器传输级特定任务上仍面临困难。检索增强生成（RAG）方法（Lewis 等，2020；Petroni 等，2021；Ho 等，2025；Ping 等，2025；Yao 等，2024）能够减少幻觉，并

[1]神经符号化**策略**的简称：**分**解–**检**索–**治**服

通过为大语言模型提供外部上下文来提高事实准确性。然而，这些方法往往依赖成本高昂的指令微调，只针对 Verilog，并且将综合或摘要生成作为孤立任务处理。此外，提示和检索增强生成流水线通常都未纳入符号支架，而符号支架能够明确表征硬件意图。

符号规划和神经符号方法已在增强生成任务的可解释性和结构感知方面展现出显著优势（Zhou 等，2022年；Pan 等）。Logic-LM（Pan 等）和 Code-as-Symbolic-Planner（Chen 等，2025年）等模型利用符号支架引导生成，但这些技术尚未扩展到寄存器传输级工作流中。我们提出 **SymDiRec**，一种面向 Verilog 和 VHDL 中寄存器传输级综合与摘要生成的神经符号分治–检索–征服 框架。SYMDIREC 由三个由符号推理驱动的阶段组成：（a）通过符号分解进行**分治**，大语言模型将高层寄存器传输级任务拆分为模块化子组件，并生成自然语言和符号表示（例如布尔逻辑或数据流逻辑）；（b）使用在 RTL-IR 数据集上进行微调的领域适配符号检索器进行**检索**，该检索器结合符号和文本线索，以获取语义相关的寄存器传输级片段；（c）通过大语言模型引导的验证与组装进行**征服**，将检索候选项与符号意图对齐，并组装成最终代码块或摘要。通过将符号逻辑融入每个阶段，SYMDIREC 提升了检索精确率、输出一致性和验证效果，且无需对大语言模型进行完整微调。实证结果表明，与强大的检索增强生成和提示基线相比，SYMDIREC 在综合任务中的 Pass@1 准确率提高了约 20%，在摘要生成任务中的 ROUGE-L 提升了 15–20%。这些结果凸显了符号推理在弥合自然语言、逻辑与 RTL语义之间鸿沟方面的价值。我们的主要贡献如下：

SymDiRec 框架： 我们提出了一种新颖的神经符号分治–检索–征服流水线，用于寄存器传输级任务，将符号分解、检索和验证整合到统一的架构中。**用于检索与验证的符号指导：** 我们表明，符号逻辑能够实现更精确的检索和结构上更一致的输出，优于自然语言方法。

跨语言 RTL 评估： 我们评估在 Verilog 和 VHDL 基准测试上评估 SYMDIREC 的综合和摘要生成效果，证明其可推广至不同的寄存器传输级领域。

轻量且模块化的推理： 我们的方法无需对大语言模型进行完整微调，只需调整检索器，同时保持相较于若干强基线方法的竞争力。

2 相关工作

Megatron-LM（Shoeybi 等人，2019年）、StarCoder（Li 等人，2023a）、CodeGen（Nijkamp 等人，2022年）、CodeLlama（Roziere 等人，2023年）和 Granite（Mishra 等人，2024年）等基于 Transformer 的代码模型，推动了多语言生成和代码推理领域的大部分进展。包括思维链（CoT）（Wei 等人，2022年）、CoDes（Vijayaraghavan 等，2024a）以及带自我验证的 CoT（Ping等人，2025年）在内的提示技术，为硬件相关任务提供了结构化推理。ReAct 提示（Yao 等，2023）通过加入迭代式的优化与行动循环，进一步提高正确性。检索增强生成（RAG）利用外部上下文为大语言模型的输出提供依据（Lewis 等人，2020年；Petroni 等人，2021年）。近期的扩展方法，例如自反思式和纠正式检索增强生成（Asai 等人，2023年；Yan 等人，2024年），以及文档级提示（Zhou 等人，2022年），能够减少幻觉并简化领域适配。REDCODER（Parvez 等人，2021年）等框架已将检索增强生成应用于通用语言的代码摘要和生成，展现了结合双重检索与生成的优势。

在硬件设计领域，模块化检索增强生成和推理策略正逐渐兴起。VerilogCoder采用任务与电路关系图以及基于AST的调试，在Verilog基准测试中实现了超过90%的通过率（Ho等人，2025年）。HDLCoRe和HDLde-bugger加入了硬件感知提示分解和证据筛选（Ping等人，2025年；Yao等人，2024年）。ComplexVCoder采用两阶段方法，结合中间表示和领域专用检索增强生成（Zuo等人，2025年）。CodeV等多级摘要生成模型（Zhao等人，2025年）通过指令调优改进了Verilog生成。所有这些方法都依赖成本高昂的模型调优或微调，且通常仅关注Verilog或单一任务，即综合或摘要生成。

专门针对 VHDL 的研究有限。VHDL-Eval 基准测试（Vijayaraghavan 等人，2024b）和 CoDes for VHDL（Vijayaraghavan

等人，2024a）凸显了大语言模型在 VHDL 方面持续存在的不足，表明亟需能够兼顾综合与摘要生成的方法。我们提出的 SYMDIREC 独特地弥补了这些不足。它将符号逻辑用作结构化中间表示，从而提升分解与检索能力。不同于仅依赖图的检索增强生成框架，符号逻辑能够捕捉功能意图，实现更精准的检索与验证。SYMDIREC 是少数同时在 Verilog 和 V HDL 上进行评估的系统之一，也是首个同时针对这两种语言开展综合与摘要生成的系统。该神经符号组合由符号分解、针对寄存器传输级语义的检索器调优以及大语言模型引导的验证构成，在保持可解释性与模块化推理能力的同时，取得了优于现有方法的性能。

3 神经符号分治-检索-征服（SY MDIREC）框架

3.1 概述

我们介绍**SymDiReC**，这是一个统一的神经符号框架，专为寄存器传输级（RTL）设计中的两项互补任务而设计：(i)综合，即将自然语言（NL）问题陈述或规范转换为相应的 RTL代码；以及 (ii)摘要生成，即将 RTL 模块转换为易于理解的自然语言解释，并辅以符号逻辑。通过将符号推理融入流水线，SymDiReC 确保分解、检索和验证在语义上均有意义。符号表示充当中间支架，可提升检索精确率和输出正确性，尤其是在涉及更具挑战性的 RTL语义时。SymDiReC 采用三阶段架构（见图2），综合和摘要生成均采用这一架构：**分治（符号分解）**：输入可以是自然语言规范或 RTL代码，它会被分解为更小且在语义上有意义的子组件。每个子组件均附有简短的文本描述和符号逻辑表示，以捕捉其功能行为。

检索（符号查询）：对于每个子组件，使用相应的符号逻辑，从结构化知识库中检索相关的 RTL代码片段或符号摘要。检索器经过训练，能够理解符号逻辑与寄存器传输级结构之间的对齐关系。

征服（大语言模型验证与组装）：大语言模型根据检索候选项与符号逻辑及描述的对齐情况进行评估，为每个子组件选出最佳候选项，并将它们组装成连贯的寄存器传输级实现或摘要。

3.2 分治：符号分解

分治阶段将输入分解为一组可解释的子组件，每个子组件都由一个符号逻辑表达式以及相应的文本或结构单元表示。此分解步骤为任务提供结构和粒度，使下游检索和验证能够以更细的粒度进行。形式化地，对于输入 X，分解函数定义如下：

$$f_{\text{DIV}}(X) = \{(x_1, \phi_1), \ldots, (x_N, \phi_N)\}$$

其中，x_i 是 i^{th} 子单元，ϕ_i 是其对应的符号逻辑表示。子单元的数量 N 是动态的，可能随输入的复杂度而变化；它由提示策略或语法驱动的划分机制决定。对于综合任务，输入 X 是自然语言规范。我们使用预训练大语言模型将该规范分解为一系列简短的自然语言描述 $\{x_i\}$，每条描述表示一个不同的功能子任务（例如计数器、比较器、多路复用器）。对于每个 x_i，我们提示同一个大语言模型生成相应的符号逻辑表达式ϕ_i，以基于逻辑的形式（例如时序、布尔或数据流表达式）捕捉预期行为。我们还可以提供一些上下文示例，以促使模型生成格式结构化且一致、符合符号硬件语义的输出。对于摘要任务，输入 X 是寄存器传输级代码。我们首先将代码解析为抽象语法树（AST），并将其划分为功能块 $\{x_i\}$，例如 ‘always‘ 块、模块或组合逻辑片段。对于每个块 x_i，我们提示大语言模型将其功能意图抽象为简洁的符号表示 ϕ_i，通常涵盖控制逻辑、状态转换或数据转换。我们使用基于提示的模板来引导这些符号形式符合寄存器传输级语义，从而确保结构一致性。

:

大语言模型引导与符号抽象：在两个方向上，符号抽象都依赖于提示预训练大语言模型，使其推理硬件功能并生成可解释的符号表达式。虽然大语言模型未必总能

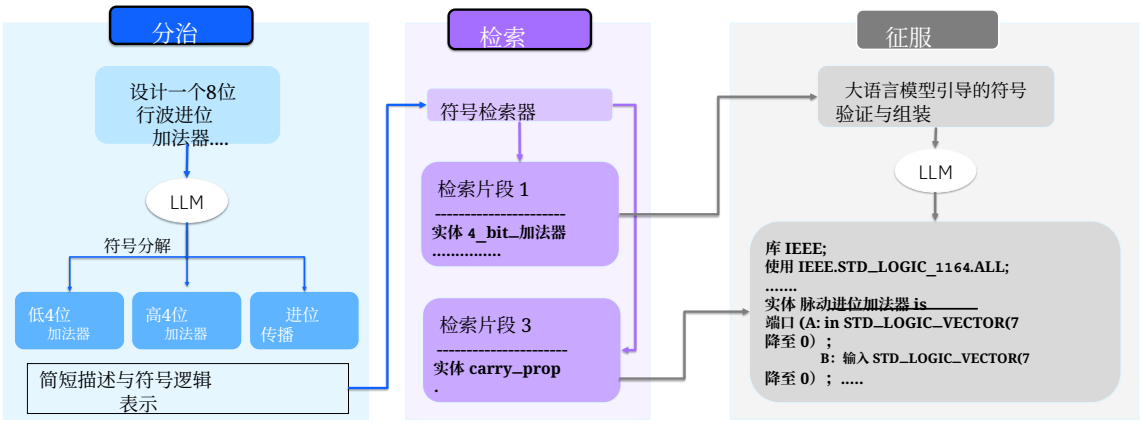


图2：我们面向 8位脉动进位加法器的 SYMDIREC 框架示意图。

生成完整的形式逻辑，但其输出通常能提供高质量近似，捕捉底层子组件的关键语义元素。这些符号草图为后续检索和对齐提供锚点。此分解过程确保每个子单元在语义上都有意义，并在结构上与寄存器传输级设计原则保持对齐，从而支持后续阶段的定向检索和稳健组合。

3.3 检索：符号检索器

检索 阶段会为每个子组件（x_i，ϕ_i）补充相关的寄存器传输级代码片段或符号/自然语言摘要。为同时利用文本描述 x_i 及其形式化符号逻辑 ϕ_i，我们设计了一种联合嵌入检索器，用于将成对查询（x_i，ϕ_i）与结构化知识库 $\mathcal{K}$ 进行匹配。我们将它们表示为：

$$f_{\text{RET}}(x_i, \phi_i) = R_i \;=\; \text{TopK}_{y \in \mathcal{K}}\, \text{score}(x_i, \phi_i; y).$$

3.3.1 知识库

我们构建了一个包含 S 索引条目的存储库：$\mathcal{K} = \{(y_j,\ d_j,\ \phi_j)\}_{j=1}^{S}$，每个条目包含：(a) y_j：一个寄存器传输级代码片段（VHDL/Verilog）或自然语言摘要；(b) d_j：对 y_j的简短自然语言解释；以及 (c) ϕ_j：符号逻辑表示。

3.3.2 JointRetriever架构

我们采用三个 Transformer 编码器构建双编码器：$e_x : \mathcal{X} \to \mathbb{R}^D$，$e_\phi : \Phi \to \mathbb{R}^D$，$e_y : \mathcal{Y} \to \mathbb{R}^D$，</style}其中 e_x 对自然语言片段 x进行编码，e_ϕ 对符号逻辑 ϕ进行编码，而 e_y 对候选条目 y进行编码。为构建联合查询表示，我们将它们拼接并投影：

$$q_i = W_q\big[e_x(x_i) \,\|\, e_\phi(\phi_i)\big] \;\in\; \mathbb{R}^D,$$

使用学习得到的投影矩阵 $W_q \in \mathbb{R}^{D \times 2D}$。随后，检索根据余弦相似度对每个条目 y_j 进行排序：

$$\text{score}(x_i, \phi_i;\, y_j) = \cos\big(q_i,\ e_y(y_j)\big).$$

3.3.3 训练目标

我们在由对齐三元组 构成的 RTL- IR 数据集上微调全部三个编码器 $\{(x_p,\ \phi_p,\ y_p)\}_{p=1}^{S}$。对于批次大小为 B的每个批次，正样本（x_p，ϕ_p，y_p）会与批内负样本$\{y_q\}_{q \neq p}$进行对比。我们通过最小化多负样本排序损失进行训练，并针对 e_x 和e_ϕ探索稠密（连续嵌入）和稀疏（加权项）两种变体；实现和超参数的详细信息见附录 B，其中还包含数据集统计信息和消融实验。

3.3.4 推理：检索子组件

在推理阶段，每个分解后的查询（x_i，ϕ_i）都会被编码为 q_i，随后我们计算查询与候选 $y_j \in \mathcal{K}$ 之间的余弦相似度分数。检索器返回排名前 k 的候选项，如下所示：

$$R_i = f_{\text{RET}}(x_i, \phi_i) = \text{TopK}_{j \in [1,S]}\, \text{score}(x_i, \phi_i; y_j),$$

从而得到 $R_i = \{r_{i,m}\}_{m=1}^{k}$。每个 $r_{i,\ j}$ 要么是 RTL 代码片段（用于综合），要么是自然语言摘要（用于摘要生成）。表 4 展示了一个 8 位波纹进位加法器的定性示例，其中包括每个子模块的布尔/逻辑表达式符号分解，以及与这些子模块对应的检索到的 Verilog 和 VHDL 代码片段。

3.4 大语言模型引导的验证与组装

CONQUER **阶段**将检索候选项整合为最终输出。给定原始输入 X

方法	LLM	Pass@1		ROUGE-L	
		Verilog	VHDL	Verilog	VHDL
原始提示	GPT-4o	0.543	0.285	43.1	39.3
	Llama-3	0.385	0.226	40.2	34.1
CoDes	GPT-4o	0.602	0.348	46.9	43.2
	Llama-3	0.435	0.274	43.5	39.5
ReAct 提示	GPT-4o	0.616	0.353	46.1	43.0
	Llama-3	0.437	0.291	42.9	38.8
VRAG-CodeBERT	GPT-4o	0.688	0.487	53.2	50.3
	Llama-3	0.527	0.396	47.4	44.5
VRAG-FT	GPT-4o	0.719	0.531	57.0	52.8
	Llama-3	0.569	0.439	50.5	48.1
RTLCoder（开源）	Mistral	0.625*	-	-	-
	GPT-4o/Llama-3	-	-	-	-
CodeV（指令微调）	CodeQwen	0.532*	-	-	-
	GPT-4o/Llama-3	-	-	-	-
SYMDIREC（我们的方法）	GPT-4o	$\mathbf{0.805}_{\pm 0.020}$	$\mathbf{0.634}_{\pm 0.022}$	$\mathbf{62.5}_{\pm 0.015}$	$\mathbf{56.6}_{\pm 0.018}$
	Llama-3	$\mathbf{0.652}_{\pm 0.022}$	$\mathbf{0.545}_{\pm 0.020}$	$\mathbf{56.1}_{\pm 0.018}$	$\mathbf{50.8}_{\pm 0.015}$
SYMDIREC-GT（预言机）	GPT-4o	**0.902**	**0.842**	**70.2**	**64.7**
	Llama-3	**0.807**	**0.721**	**63.3**	**57.9**

表 1：不同方法和大语言模型在 RTL 综合（Pass@1）和摘要生成（ROUGE-L）方面的表现。对于我们的 SYMDIREC，报告五次独立运行的均值 ± 标准差。与最强基线方法相比，结果具有统计显著性（配对 t 检验，$p < 0.01$）。 * 表示结果引自原论文。

以及检索得到的集合 $\{R_i\}_{i=1}^{N}$，最终产物 $\hat{}Y$ 由以下过程生成：$\hat{}Y = f_{\text{CONQ}}\big(X,\ \{R_i\}_{i=1}^{N}\big)$，其中$\hat{}Y$是合成的寄存器传输级模块或生成的自然语言描述。给定 $R_i = \{r_{i,m}\}_{m=1}^{k}$是每个子组件的前$k$ 个候选项i，根据第 3.3 节，我们提示生成器大语言模型基于子组件 x_i 及其关联的符号逻辑表示 ϕ_i $\forall m \in \{1,k\}$，为候选项分配对齐分数：$\hat{}\alpha_{i,\ m}$ = 验证_得分$\big(r_{i,\ m},\ x_i,\ \phi_i\big) \in [0,1]$；这些分数既反映功能正确性，也反映检索器结果与符号规范的吻合程度。我们为每个子组件选取得分最高的候选项。由此得到一组经过验证的子模块或摘要 $\{\hat{}r_i\}_{i=1}^{N}$。最终组装阶段以 X 和经过验证的候选项为条件调用大语言模型。此步骤生成连贯的 RTL 设计或全面的自然语言摘要，确保命名一致、模块连接性良好且逻辑完整。

4 实验

本节概述我们的实验设置，包括寄存器传输级基准测试（涵盖 VHDL 和 Verilog）、多样化的基线模型套件，以及用于评估所提出方法的有效性、泛化能力和效率的指标。

SYMDIREC 框架。完整的实现和数据集详情请参阅附录 A。我们的评估围绕以下研究问题展开：**（RQ1）SYMDIREC 的有效性：**与现有基线方法相比，SYMDIREC 框架在寄存器传输级综合和摘要生成方面的效果如何？（**RQ2）符号逻辑的影响：**符号逻辑表示在多大程度上提升了检索质量，以及下游寄存器传输级生成或摘要生成的效果？**（RQ3）超参数敏感性：**关键超参数会如何影响 SYMDIREC 的性能？

4.1 基准测试与基线方法

我们在两个标准寄存器传输级基准测试上评估 SYMDIREC：**Verilog-Eval**包含来自 HDLBits 的 156 个功能性 Verilog 任务（Liu等人，2023年），并配有测试平台；以及**VHDL-Eval**，包含 202 个由 Verilog-Eval 或公开教程翻译而来的 VHDL 任务（Vijayaraghavan等人，2024b），并采用类似的验证流程。我们的比较方法包括零样本提示、中间计划式 CoDes（Vijayaraghavan等人，2024a）、迭代式 ReAct（Yao等人，2023年）、原始检索增强生成（VRAG-CodeBERT），以及在**RTL-IR**数据集上微调的检索增强生成（VRAG-FT）。我们还纳入了近期的领域专用模型 RTLCoder（Liu等人，2024年）和 CodeV（Zhao等人，2025年）。最初

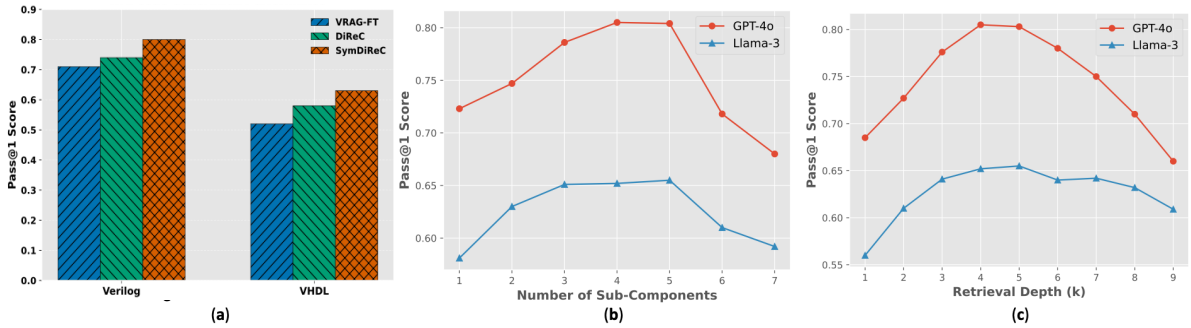


图3: 消融结果：不同（a）子组件数量和（b）分块策略下的性能 .

RTL-IR 数据集统计信息	
# 总规模	~ 50.5k
# 文本到代码对	~ 8k
# FEC 对	~ 13.5k
# 代码到摘要对	~ 6.5k
# 部分代码到完整代码对	~ 22.5k

表2：用于模型微调和检索增强的 RTL-IR 数据集统计信息。

后，我们使用真实片段评估 SYMYMDIREC 及其预言机变体（SYMYMDIREC-GT）。RTL-IR 数据集是一个经过筛选的 RTL代码和注释集合，用于微调检索增强生成模型，包含文本转代码、功能等效代码（FEC）、代码转摘要以及由部分代码补全为完整代码的配对数据。表2 总结了数据集统计信息。数据集的详细描述、基准测试和其他示例见附录A、C。基线方法采用 GPT-4o 和 Llama-3（70B），分别使用 Pass@1 评估代码合成，使用 ROUGE-L 评估摘要生成。结果报告 5 次运行的均值 ± 标准差，并采用配对 t 检验。

4.2 评估指标

对于 RTL 综合，我们采用 Pass@1 指标，该指标表示首次生成的 RTL 设计成功通过 Verilog-Eval 和 VHDL-Eval 基准测试所提供的自检测试平台的比例。对于代码摘要生成，我们采用 ROUGE-L，用于衡量生成摘要与参考摘要之间的最长公共子序列（LCS），并计算 F 值以评估流畅性和连贯性（Lin，2004年）。

5 结果与讨论

5.1 SYMDIREC 的总体性能

表1 比较了 SYMDIREC 与多种强基线方法，包括仅使用提示的方法和检索增强生成（RAG）策略。我们的神经符号流水线始终优于所有其他方案，展现了符号分解、领域适配检索器和大语言模型引导的验证的有效性。

5.2 SYMDIREC 的有效性（RQ1）

原始提示作为各模型的零样本下限，在综合和摘要任务中均表现最弱。相比之下，SYMDIREC-GT 代表了基于检索增强生成的方法的近似上限，因为它始终将真实标注包含在排名前 k 的检索候选项中。尽管上下文中存在正确解，但 SYMDIREC-GT 在特定实验案例中的失败表明，大语言模型仍可能难以从检索候选项中筛除干扰项，或缺乏足够的 RTL 专用推理能力。描述链（CoDes）与 Re-Act 提示的性能相当。相较于描述链，Re-Act 在 Pass@1 上展现出小幅优势（~5-8% 的相对提升），这可能是因为其迭代式推理/行动循环提供了简单描述链所缺乏的纠错路径。在检索增强生成策略中，采用微调检索器的 VRAG-FT 始终优于 VRAG-CodeBERT，在 Pass@1 上实现了 ~10%-15% 的相对提升，在 ROUGE-L 上实现了 ~5%-8% 的相对提升。这表明，尽管通用代码检索器有益于 RTL任务，但通过将自然语言查询更有效地与 RTL语义对齐，任务特定的调优能够进一步提升检索质量。SYMDIREC 显著优于所有基线方法：相较于最佳的基于提示的方法，其 Pass@1 的相对提升最高可达 ~80%；相较于仅使用检索增强生成的基线方法，最高可达 ~20%。对于摘要任务，SYMDIREC 在 ROUGE-L 上相较于提示方法最高提升 ~35%，相较于检索增强生成最高提升 ~10%，凸显了符号规划和 RTL感知检索的优势；其与 SYMDIREC-GT 之间剩余的 ~10–15%

差距表明，检索器精确率和大语言模型对齐仍有改进空间。

5.3 符号逻辑的影响（RQ2）

为单独评估符号逻辑带来的益处，我们比较了三种流水线变体：VRAG-FT，它在自然语言查询上使用微调后的检索器；DiReC，它采用相同的分解–检索–征服结构，但仅使用自然语言查询进行检索；以及完整的 SymDiReC，它结合了符号分解、模块化检索和大语言模型引导的验证。结果表明，SymDi-ReC 的 ∼10 Pass@1 相对提升了 15%（参见图 3(a)），且 ROUGE-L 得分高于 VRAG-FT 和 DiReC。这些发现表明，符号表示能够发挥强大的支撑作用，使检索更加精准、减少噪声，从而提升综合和摘要生成的效果。

5.4 超参数敏感性（RQ3）

我们通过消融实验探究关键超参数——子组件数量（N）和检索深度（k）——如何影响 SymDiReC 的性能。图 3(b) 和 (c) 总结了实验结果。将 N 从 2 增至 6，可持续提升 Pass@1；但当 N 超过 6 时，性能便开始下降，这可能是因为过度分割导致子单元的语义关联减弱。值得注意的是，N 的最优值取决于任务。较简单的组合逻辑任务（如加法器、多路复用器）适合较小规模的分解（$N \approx 3$–4）；而多阶段时序设计（如计数器、有限状态机）采用稍大的 N ($N \approx 4$–5）时效果更好，这反映了它们更高的结构复杂度。对于检索深度，随着 k 增至 5，性能有所提升；但当 k 过大时，性能会趋于平稳并最终下降。这一下降可能是由于相关性较低的检索结果引入了额外噪声。在各项基准测试中，采用 $N = 4$ 和 $k = 5$作为默认配置，能够在综合准确率、摘要质量和计算效率之间取得良好平衡，同时也允许在适当情况下针对具体任务进行调优。

6 错误分析

我们对 SYMDIREC 进行详细的错误分析，以了解其在寄存器传输级综合和摘要生成方面的局限性。我们的调查聚焦于三个主要错误来源：(a) **符号分解错误：** 大约 8-10% 的子组件存在不完整或不一致的符号表达式，尤其是在多位比较器或时序元件中。这些错误与较低的检索精确率相关，使受影响设计的 Pass@1 性能降低多达 5-7%；(b)**检索不匹配：** 即使有符号支架，约 12-15% 的检索候选项也只能部分匹配预期行为，或包含干扰项。这会导致 Pass@1 准确率下降 3-6%，摘要生成的 ROUGE-L 分数下降 2-4 分；以及 (c)**大语言模型汇编与验证失败：** 检索候选项正确时，大语言模型偶尔仍无法正确整合它们（信号错位、连接缺失或进位传播问题），约有 6-8% 的子组件出现此类情况。这些失败导致 SYMDIREC 与真实值 SYMDIREC-GT 在综合方面仍存在 ∼10-15% 的差距，在摘要生成方面仍存在 6–8 个 ROUGE-L 分数点的差距。定性检查显示，大多数失败发生在层次化设计或不常见的模块模式中。SYMDIREC-GT 的结果表明，提高检索精确率和符号推理能力，或可弥合相当一部分性能差距。

7 结论

我们提出了 SYMDIREC，这是一种面向 Verilog 和 VHDL 中寄存器传输级综合与摘要生成的神经符号 DiReC（分解–检索–CONQUER）框架。通过整合符号分解、领域自适应检索和大语言模型引导的验证，SYMDIREC 有效弥合了硬件形式语义与大语言模型生成之间的鸿沟。不同于严重依赖指令微调或忽视符号意图的既有方法，我们的方法引入了结构化中间推理，从而同时提升了检索相关性和生成正确性。实证结果表明，与基于提示和基于检索增强生成的基线方法相比，我们的方法持续取得更好的表现，在综合准确率和摘要质量方面均有所提升。这项工作凸显了符号规划在程序综合中的实用价值，并为硬件设计自动化及其他领域中可解释、模块化的神经符号系统开辟了新的方向。

局限性

尽管 SYMDIREC 在寄存器传输级综合和摘要生成方面表现出色，但仍存在若干局限。符号分解

依赖大语言模型生成格式良好的符号表达式的能力。规模较小或能力较弱的模型可能生成不完整或不一致的分解，从而削弱符号脚手架的优势，使性能与纯自然语言查询相近。CONQUER 阶段的大语言模型引导的验证也可能无法使检索候选项与预期逻辑对齐，尤其是在 Top-k 检索结果包含干扰项或仅部分匹配的片段时。分解粒度十分关键：过细的分段会将输入切分得过于零散，产生质量较差的子单元；而过粗的分段则可能降低检索精度。当前流水线仅适用于单文件寄存器传输级设计，不支持层次化或多文件项目，而这类项目通常存在跨模块依赖。要扩展至此类设计，可能需要更先进的抽象语法树处理、块级和函数级分块，以及多层摘要生成策略。即使检索到了正确答案，大语言模型也可能无法选中它，这凸显了噪声过滤、候选项排序以及在检索结果不完善时进行推理所面临的挑战。要将 SYMDIREC 扩展至复杂的现实硬件设计场景，解决这些问题至关重要：提升符号推理能力、检索对齐能力和层次化抽象能力。

伦理声明

我们使用了公开的数据集（例如 Verilog-Eval）以及自行整理的 RTL-IR 数据集，后者来源于采用宽松许可证（MIT 许可证、BSD 许可证、Apache-2.0 许可证）的 GitHub 代码库；许可证元数据见补充材料。我们未使用任何私人或敏感数据；输出仅用于研究和辅助开发。潜在风险包括：若未经验证便投入使用，生成的硬件设计可能存在错误或安全隐患。本系统旨在作为辅助开发工具，所有输出在实际使用前都应通过标准测试平台和人工审查进行验证。

参考文献

Akari Asai、Zeqiu Wu、Yizhong Wang、Avirup Sil 和 Hannaneh Hajishirzi。2023年。Self-rag：通过自我反思学习检索、生成和评析。载于第十二届国际学习表征会议。

Yongchao Chen、Yilun Hao、Yang Zhang 和 Chuchu Fan。2025年。Code-as-symbolic-planner：基础基于模型的机器人规划，通过符号代码生成。*arXiv* 预印本 *arXiv:2503.01700*。

Elastic。2024年。机器学习：使用 Elastic 进行自然语言处理。https://www.elastic.co/guide/en/machine-learning/current/ml-nlp-elser.html。访问日期：2025年3月21日。

Thibault Formal、Carlos Lassance、Benjamin Piwowarski 和 Stéphane Clinchant。2021年。SPLADE v2：用于信息检索的稀疏词汇扩展模型。*arXiv* 预印本 *arXiv:2109.10086*。

Kelvin Guu、Kenton Lee、Zora Tung、Panupong Pasupat 和 Mingwei Chang。2020年。检索增强语言模型预训练。收录于机器学习国际会议，第 3929–3938 页。机器学习研究论文集。

Chia-Tung Ho、Haoxing Ren 和 Brucek Khailany。2025年。VerilogCoder：采用基于图的规划和基于抽象语法树（AST）的波形追踪工具的自主Verilog编码智能体。收录于*AAAI*人工智能会议论文集，第 39 卷，第 300–307 页。

PatrickLewis、Ethan Perez、Aleksandra Piktus、Fabio Petroni、Vladimir Karpukhin、Naman Goyal、Hein- rich Küttler、Mike Lewis、Wen-tau Yih、Tim Rock- täschel 及其他 1 位作者。2020年。面向知识密集型自然语言处理任务的检索增强生成。神经信息处理系统进展，33:9459–9474。

Raymond Li、Yangtian Zi、Niklas Muennighoff、Denis Kocetkov、Chenghao Mou、Marc Marone、Christopher Akiki、Jia LI、Jenny Chim、Qian Liu等。2023a。Starcoder：愿源代码与你同在！机器学习研究汇刊。

Zehan Li、Xin Zhang、Yanzhao Zhang、Dingkun Long、Pengjun Xie和Meishan Zhang。2023b。通过多阶段对比学习实现通用文本嵌入。*arXiv*预印本 *arXiv:2308.03281*。

Chin-Yew Lin。2004年。ROUGE：用于摘要自动评估的软件包。载于文本摘要拓展研讨会。

Mingjie Liu、Nathaniel Ross Pinckney、Brucek Khailany和Haoxing Ren。2023年。VerilogEval：评估大语言模型的Verilog代码生成能力。载于*ICCAD*。

Shang Liu、Wenji Fang、Yao Lu、Jing Wang、Qijun Zhang、Hongce Zhang 和 Zhiyao Xie。2024年。RTL Coder：一种完全开源且高效的大语言模型辅助寄存器传输级代码生成技术。*IEEE*集成电路与系统计算机辅助设计汇刊。

Mayank Mishra、Matt Stallone、Gaoyuan Zhang、Yikang Shen、Aditya Prasad、Adriana Meza Soria、Michele Merler、Parameswaran Selvam、Saptha Surendran、Shivdeep Singh等。2024年。Granite代码模型：面向代码智能的一系列开放基础模型。*CoRR*。

Niklas Muennighoff、Nouamane Tazi、Loic Magne和Nils Reimers。2023年。Mteb：大规模文本嵌入基准。载于第*17*届欧洲计算语言学协会分会会议论文集。计算语言学协会。

Erik Nijkamp、Bo Pang、Hiroaki Hayashi、Lifu Tu、Huan Wang、Yingbo Zhou、Silvio Savarese 和 Caiming Xiong。2022年。Codegen：一种面向代码、支持多轮程序合成的开放式大语言模型。载于第十一届国际学习表征会议。

Liangming Pan、Alon Albalak、Xinyi Wang 和 William Yang Wang。Logic-lm：借助符号求解器增强大语言模型，实现忠实的逻辑推理。载于*2023*年自然语言处理经验方法会议。

Md Rizwan Parvez、Wasi Uddin Ahmad、Saikat Chakraborty、Baishakhi Ray 和 Kai-Wei Chang。2021年。检索增强代码生成与摘要。载于*EMNLP-Findings*。

F Petroni、A Piktus、A Fan、PSH Lewis、M Yazdani、ND Cao、J Thorne、Y Jernite、V Karpukhin、J Mail-lard 等。2021年。Kilt：知识密集型语言任务基准。载于*NAACL-HLT*，第2523–2544页。计算语言学协会。

Heng Ping、Shixuan Li、Peiyu Zhang、Anzhe Cheng、Shukai Duan、Nikos Kanakaris、Xiongye Xiao、Wei Yang、Shahin Nazarian、Andrei Irimia 及其他 1 位作者。2025年。HDLCoRe：用于缓解大语言模型生成的硬件描述语言中的幻觉的免训练框架。*arXiv* 预印本 *arXiv:2503.16528*。

Nils Reimers 和 Iryna Gurevych。2019年。Sentence-bert：使用孪生BERT网络的句子嵌入。发表于*2019*年自然语言处理实证方法会议和第九届国际自然语言处理联合会议（*EMNLP-IJCNLP*）论文集，第 3982–3992 页。

Stephen Robertson、Hugo Zaragoza 及其他 1 位作者。2009年。概率相关性框架：BM25 及其后续发展。《信息检索基础与趋势》，3(4):333–389。

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin，以及其他 1 位作者。2023年。Code Llama：用于代码的开放基础模型。*arXiv*预印本 *arXiv:2308.12950*。

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper 和 Bryan Catan-zaro。2019年。Megatron-LM：利用模型并行训练数十亿参数语言模型。*arXiv*预印本 *arXiv:1909.08053*。

Aivin V. Solatorio。2024年。GistEmbed：用于文本嵌入微调的训练负例引导式样本内选择。*arXiv*预印本 *arXiv:2402.16829*。

Prashanth Vijayaraghavan、Apoorva Nitsure、Charles Mackin、Luyao Shi、Stefano Ambrogio、Arvind Haran、Viresh Paruthi、Ali Elzein、Dan Coops、David Beymer及另外1位作者。2024a。《描述链：提升面向VHDL代码生成与摘要生成的代码大语言模型》。载于*2024*年*ACM/IEEE*计算机辅助设计机器学习国际研讨会论文集，第1–10页。

Prashanth Vijayaraghavan, Luyao Shi, Stefano Am-brogio, Charles Mackin, Apoorva Nitsure, David Beymer 和 Ehsan Degan。2024b。VHDL-Eval：用于评估 VHDL 代码生成中大语言模型的框架。发表于*2024 IEEE*大语言模型辅助设计研讨会 *(LAD)*，第 1–6 页。IEEE。

Yue Wang、Weishi Wang、Shafiq Joty 和 Steven CH Hoi。2021年。Codet5：面向代码理解与生成的标识符感知统一预训练编码器—解码器模型。载于*2021*年自然语言处理实证方法会议论文集，第8696–8708页。

Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Fei Xia、Ed Chi、Quoc V Le、Denny Zhou 等。2022年。思维链提示能够在大语言模型中引发推理。神经信息处理系统进展，35：24824–24837。

Shi-Qi Yan、Jia-ChenGu、Yun Zhu 和 Zhen-HuaLing。2024年。纠错式检索增强生成。*arXiv* 预印本 *arXiv:2401.15884*。

Shunyu Yao、Jeffrey Zhao、Dian Yu、Nan Du、Izhak Shafran、Karthik Narasimhan 和 Yuan Cao。2023年。ReAct：在语言模型中协同推理与行动。载于国际学习表征会议（*ICLR*）。

Xufeng Yao、Haoyang Li、Tsz Ho Chan、Wenyi Xiao、Mingxuan Yuan、Yu Huang、Lei Chen 和 Bei Yu。2024年。HDLDebugger：利用大语言模型简化HDL调试。*ACM*电子系统设计自动化汇刊。

Dun Zhang。2023年。stella_en_400m_v5。https://huggingface.co/dunzhang/stella_en_400M_v5。Hugging Face 模型卡片，访问日期为2025年3月21日。

Yang Zhao、Di Huang、Chongxiao Li、Pengwei Jin、Muxin Song、Yinan Xu、Ziyuan Nan、Mingju Gao、Tianyun Ma、Lei Qi 和另外1位作者。2025年。CodeV：通过多级摘要赋能大语言模型进行硬件描述语言生成。*IEEE*计算机辅助设计与集成电路系统汇刊。

Shuyan Zhou、Uri Alon、Frank F Xu、Zhengbao Jiang 和 Graham Neubig。2022年。Docprompting：通过检索文档生成代码。载于《第十一届国际学习表征会议》。

Jian Zuo、Junzhe Liu、Xianyong Wang、Yicheng Liu、Navya Goli、Tong Xu、Hao Zhang、Umamah-eswara Rao Tida、Zhenge Jia 和 Mengying Zhao。2025年。ComplexVCoder：一个由大语言模型驱动的复杂 Verilog 代码系统化生成框架。*arXiv* 预印本 *arXiv:2504.20653*。

A 数据集

A.1 RTL-IR

A.1.1 数据收集与预处理 g

该数据集整理自 GitHub 上公开的 VHDL/Verilog 代码库。我们根据宽松许可筛选代码库，并选取了注释和 README 描述有实际意义的 VHDL/Verilog 项目。预处理流程包括：

- 提取注释和 README 文档。
- 使用 Granite-13b-Instruct 进行上下文学习（ICL），以完善问题陈述和代码摘要。
- 应用多种变换来生成在功能上等价的代码对。

文本转代码（TC）对 为构建 TC 对，我们从 VHDL/Verilog 文件中的注释以及 README 文档的相关部分提取自然语言描述。由于原始注释可能缺乏结构，我们使用搭配 Granite-13b-Instruct 的 ICL 来生成结构化问题陈述，并对这些陈述进行验证，以确保其清晰且与对应的 VHDL/Verilog 代码相关。

代码转摘要（CS）对 我们通过将 VHDL/Verilog 代码映射为文本摘要来创建 CS 对。优先选取结构注释完善的代码文件，并使用 ICL 将详细注释转换为简洁摘要。为评估摘要质量，我们对 100 个示例进行了人工标注，并将其分为以下类别：

- 良好 （清晰、准确且信息丰富）。
- 可接受 （提供部分信息，但仍有用）。
- 较差 （不完整或具有误导性）。

总体而言，84% 的摘要被评为良好或可接受，而 16% 被评为较差。后者被视为“困难负样本”。

功能等价代码（FEC）对我们通过对功能相同的 VHDL/Verilog 代码应用不同的转换策略来生成 FEC 对，形成不同的变体。这些转换包括：

• **类型-2：** 在保持功能等价性的同时重命名标识符。我们采用基于大语言模型的重命名策略，提取并重命名实体、架构、进程和端口名称。单字符标识符被替换为大语言模型建议的替代名称，而复杂标识符则通过缩写、排列或转换来保持可读性。

• **类型-3：** 调整语句顺序并引入功能上无影响的代码，在保留功能的同时确保代码有所变化。重新排列声明并重构条件逻辑，进一步增加了多样性。

• **类型-4：** 使用 GHDL 和 ICARUS Iverilog 在 VHDL 与 Verilog 之间进行回译。此过程改变了变量名称并引入中间信号，在最大限度降低词法相似性的同时，呈现功能等效的结构。

图 4 通过示例展示了这些变换类型。

部分到完整代码（PC）代码对 PC代码对是通过从较大的代码库中提取部分 VHDL/Verilog 片段，并将其与对应的完整版本配对而创建的。为确保表示在词法上有所差异，我们对部分完整版本应用了类型-2变换。片段提取仅限于包含少于 1024 个词元的代码段，其中包括函数声明和实体定义，以及相关的上下文注释。

A.1.2 质量控制措施

为确保数据集完整性和实用性，我们采取了以下质量控制措施：

• **编译验证：** 所有经过功能转换的代码均进行了编译测试，以确保其正确性。

• **测试平台执行：** 我们运行了 GitHub 上可用的测试平台，以验证功能等价性。

• **人工审查：** 我们对代码摘要进行了人工审查，并将质量较低的摘要标记为“困难负例”。

这些措施提高了数据集的可靠性，确保其可作为 VHDL代码生成和摘要任务的有力基准。

B 检索器的训练与评估

检索 我们使用 RTL-IR 数据集，对来自三类的模型进行微调：稀疏检索器、稠密检索器和混合检索器。稀疏检索器为 BM25（Robertson 等人，2009年）。对于稠密模型，我们从 MTEB 排行榜（Muennighoff 等人，2023年）中选取表现最佳的模型进行微调，包括 GTE-Qwen-1.5b（Li 等人，2023b）、Stella-400m（Zhang，2023）、GIST-Large（Solatorio，2024年），以及用于代码嵌入的 CodeT5+（Wang 等人，2021 年）和句子变换器（ST）（Reimers 和 Gurevych，2019年）。混合方法包括 SPLADE（Formal 等人，2021年）和 ELSER（Elastic，2024年）。除 ELSER 外，所有模型都在 RTL-IR 训练集上进行微调。我们在留出的测试集上进行评估。

B.1 评估指标

检索： 我们采用归一化折损累积增益（NDCG）评估排序质量，对列表中排名靠前的高度相关结果给予更高奖励。NDCG@1 衡量排名第一的结果的相关性，而 NDCG@10 则评估前 10 个位置的排序效果，并为排名靠前的项目赋予更高权重。

B.2 检索器性能

表3展示了多种检索方法在 RTL-IR 测试集上的表现，采用 NDCG@1 和 NDCG@10 作为评估指标。结果表明，稠密检索方法始终优于混合方法和稀疏方法，因为 RTL-IR 需要识别超越表层词汇重叠的语义相关匹配。在所评估的模型中，CodeT5+ 表现最佳，NDCG@1 达到 0.657，NDCG@10 达到 0.872，展现出其检索相关 VHDL/Verilog 代码片段的强大能力。这一性能优势可归因于 CodeT5+在针对 RTL-IR 微调之前，先使用 VHDL/Verilog 代码进行预训练。

Stella-400m（归一化折损累积增益@1 = 0.656）和 GTE-Qwen-1.5b（归一化折损累积增益@1 = 0.644）等文本嵌入模型紧随其后，尽管它们仅在 RTL-IR 数据集上进行微调。它们的有效性与较大的参数规模和嵌入维度有关（例如，Stella-400m 的嵌入维度为 8192），这使其能够更好地泛化。然而，CodeT5+的代码专项训练似乎弥补了其较小的嵌入维度，从而带来更优异的检索性能。

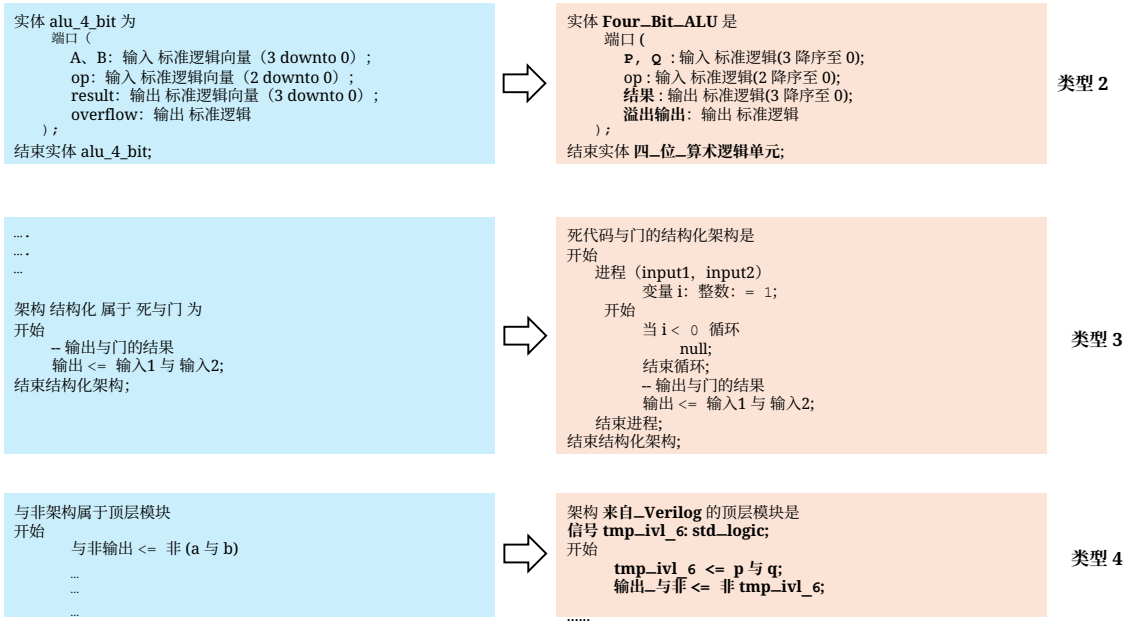


图4：采用三类代码克隆——Type2、Type 3 和 Type4——应用不同转换策略后的 VHDL 示例。

方法	NDCG@1	NDCG@10
BM25	0.434	0.570
ELSER	0.485	0.664
SPLADE	0.577	0.688
GTE-Qwen-1.5b	0.644	0.864
Stella-400m	0.656	0.866
GIST-Large	0.616	0.802
CodeT5+	**0.657**	**0.872**
ST	0.556	0.665

表3：在 RTL-IR 测试集上对稀疏、混合和稠密检索器的评估。

C 基准测试与基线方法

C.1 基准测试

Verilog-Eval： 来自 HDLBits 的 156 个功能性 Verilog 任务（ Liu 等人，2023 年），每项任务均配有可自行验证的测试平台。这些任务涵盖多种寄存器传输级结构，包括组合逻辑、时序模块、计数器、比较器和简单的有限状态机。

VHDL-Eval： 202 个 VHDL 任务，译自 Verilog-Eval 问题或取自公开的 VHDL 教程（Vijayaraghavan等人，2024b），并配有用于功能验证的测试平台。该套件在功能多样性方面与 Verilog-Eval 相近。

C.2 基线方法

我们将 SYMDIREC 与多种基线方法进行比较，其中包括近期的领域专用模型：

原始提示（ZS）： 零样本自然语言到寄存器传输级代码/摘要生成。

描述链（CoDes）（Vijayaraghavan等人，2024a）：使用中间文本计划（描述）来指导大语言模型综合。

ReAct 提示（Yao等人，2023年）：通过迭代的推理与行动循环逐步生成并优化。

原始检索增强生成（VRAG-CodeBERT）： 使用未经寄存器传输级专项微调的通用 CodeBERT 检索器。

VRAG-FT： 使用在我们的寄存器传输级 RTL-IR 数据集上微调的检索器进行检索增强生成，该数据集包含对齐的（自然语言、符号逻辑、寄存器传输级）三元组。

RTLCoder（Liu等人，2024年）：一款专为寄存器传输级生成训练的开源大语言模型，旨在实现高效运行并支持本地部署。

CodeV（Zhao等人，2025年）：采用多层级摘要策略的指令微调 Verilog 生成大语言模型，研究表明，该策略先进行摘要再生成，可提升代码生成效果。

SymDiRec:我们完整的神经符号流水线，结合了符号分解、检索和大语言模型引导的验证。

SymDiRec-GT(预言机)： 一种上界变体，使用真实标注的符号片段进行检索，以评估理想检索条件。

C.3 大语言模型设置与评估

我们使用两个大语言模型运行所有方法：一个专有模型（GPT-4o）和一个开源模型（Llama-3, 70B）。我们使用自验证测试平台上的 Pass @1 衡量综合正确性，并使用 ROUGE-L 与参考摘要进行对比，以评估摘要质量。为确保统计稳健性，我们对每种设置进行五次独立运行，并报告均值和标准差；配对 t 检验（例如，比较 SYMDIREC 与 VRAG-FT 或其他基线方法）也会进行计算，并在主要结果表中报告。

D 实现细节与计算成本

我们的系统使用 Python 和 PyTorch 框架实现，支持灵活的模型开发和高效训练。检索器微调在两块 NVIDIA V100 GPU 上进行，从而能够高效处理我们的 RTL-IR 数据集。我们使用 LangChain 编排大语言模型查询，并借助 Milvus（一种高性能向量数据库）为高维向量表示建立索引；Milvus 同时支持内存内和 GPU 加速的相似性搜索。相比之下，Elasticsearch（Elastic，2024年）和 Elastic 的学习式稀疏编码器表示（ELSER）（Elastic，2024年）等传统搜索解决方案则作为基线方法；尽管 Elasticsearch 擅长全文搜索，但与 Milvus 和 ELSER 相比，其语义检索能力有限。

我们的 SymDiReC 流水线可并行处理每项任务的多个提示词，平均周转时间约为 5–10 秒。这种效率表明，我们的方法在 RTL 代码综合和摘要任务中具有实际可行性。这些实现选择与近期关于检索增强生成（Lewis 等人，2020年；Guu 等人，2020年）和领域专用微调策略的研究相一致。通过 Milvus 实现高级索引，并使用 LangChain 进行查询编排，不仅优于传统检索方法，还显著提升了系统的整体性能。

Task	符号分解	检索上下文
8位行波进位加法器	**最低有效位半加器：** $S_0 = A_0 \oplus B_0$, $C_1 = A_0 \wedge B_0$ **位 1–7 全加器：** $S_i = A_i \oplus B_i \oplus C_i$, $C_{i+1} = (A_i \wedge B_i) \vee (B_i \wedge C_i) \vee (A_i \wedge C_i)$	**半加器（LSB）** Verilog: 模块 half_adder(input a, b, o utput sum, carry); 赋值 sum = a ^ b; 赋值 carry = a & b; 结束模块 VHDL: 实体 half_adder 是 端口(a, b：输入 std_逻辑; sum，进位：输出 std_逻辑）; 结束实体; 半加器的寄存器传输级架构 _adder 是 开始 和 <= 异或 b; 进位 <=与 b; 结束架构; **全加器（位 1–7）** Verilog: 模块 full_adder（输入 a, b，cin, 输出 sum, cout)^;b ^ 赋值 sum = a cin; 赋值 cout =(a & b) |(b & cin)	(a & cin); 结束模块 VHDL: 实体 full_adder is 端口(a, b，cin：输入 std_logic；sum, cout：输出 std_logic）; 结束实体; full 的寄存器传输级架构 _加法器为 开始 sum <= a 异或 b 异或 cin; cout <= (a 与 b) 或 (b 与 cin) 或 (a 与 cin); 结束架构;

表4：8位行波进位加法器的定性示例。符号分解展示了每个子模块的布尔/逻辑表达式。检索上下文包含与这些子模块对应的模块化Verilog和VHDL代码片段。所有子模块均通过了仿真/测试平台验证。