VeriAgent：一种采用演化记忆的工具集成多智能体系统，用于 PPA 感知的 RTL 代码生成

Yaoxiang Wang[1] , **Qi Shi**[2*]、**Shangzhan Li**[3] ,**Qingguo Hu**[1],**Xinyu Yin**[1],**Bo Guo**[2], **Xu Han**[2], **Maosong Sun**[2],**Jinsong Su**[1*]厦门大学 [2]清华大学 [3]哈尔滨工业大学

arXiv:2603.17613v1 [cs.CL] 18 Mar 2026

摘要

近年来，大语言模型在自动 RTL 代码生成方面展现出强大的能力，在语法正确性和功能正确性方面均取得了较高水平。然而，大多数方法侧重于功能正确性，却忽视了功耗、性能和面积等关键物理设计目标。本文提出一种面向高质量 Verilog 代码生成的 PPA 感知工具集成多智能体框架。该框架将电子设计自动化工具明确纳入闭环工作流，其中包含程序员智能体、正确性智能体和*PPA* 智能体，从而实现对功能正确性和物理指标的联合优化。为在无需模型重新训练的情况下持续改进，我们提出演化记忆机制，将优化经验外化为结构化记忆节点。专用的记忆管理器动态维护记忆池，并使系统能够依据历史执行轨迹优化策略。大量实验表明，我们的方法在实现较高功能正确性的同时，显著改善了 PPA 指标。通过将工具驱动的反馈与结构化、可演化的记忆相结合，该框架将 RTL 生成从一次性推理转变为持续的反馈驱动优化过程，为大语言模型在真实硬件设计流程中的应用提供了可扩展的途径。

1 引言

随着集成电路的规模和复杂度不断增长，现代芯片设计流程正面临前所未有的挑战（Chang 等人，2023年）。因此，手动编写和优化寄存器传输级（RTL）代码的成本和耗时日益增加。在此背景下，自动化 RTL 代码生成（Liu 等人，2024年）已成为缩短设计周期、减少工程投入并提升整体设计质量的重要方向。作为占主导地位的硬件描述语言，Verilog 在数字电路设计自动化中发挥着核心作用。近年来，大型语言模型（Achiam 等人，2023年；Team 等人，2023年；Guo 等人，2025年；Yang 等人，2025a）在代码生成方面展现出卓越能力，并已成功应用于 RTL 代码生成任务，在语法正确性和功能正确性方面取得了显著提升。

尽管取得了这些进展，但基于大语言模型的 RTL 代码生成距离在工业环境中部署仍有相当大的差距（Yang等人，2025b）。首先，现有方法大多主要关注功能正确性，却忽视了真实工作流程中至关重要的物理设计指标，例如功耗、性能和面积（PPA）（Thorat 等人，2025年）。其次，尽管 RTL 设计本质上依赖仿真和综合等电子设计自动化（EDA）工具链，但当前方法（Zhao等人，2025b；Tasnia等人，2025年）仅以浅层方式利用环境反馈，未能建立持续的优化循环。更重要的是，现有系统通常缺乏长期优化记忆机制。它们无法在多轮设计迭代中积累和优化知识，导致优化过程严重依赖一次性推理，而非持续改进。

现有基于大语言模型的 RTL 代码生成方法大致可分为两类：基于训练的方法和免训练方法。基于训练的方法（Pei 等人，2024年；Liu 等人，2024年；Zhang 等人，2025年；Zhao 等人，2025a）通常依赖大规模合成数据进行监督微调或强化学习，以提升模型在 RTL 任务上的性能。然而，这些方法往往依赖自动生成的样本，而这些样本的分布存在显著偏差（例如简单模块和模板化模式），限制了它们在复杂设计场景中的泛化能力。此外，训练会产生高昂的计算成本；在实践中，与直接使用最先进的闭源商用大语言模型相比，训练所得模型的表现往往较差。

*通讯作者。

生成的样本分布往往存在显著偏差（例如简单模块和模板化模式），限制了模型在复杂设计场景中的泛化能力。此外，训练需要付出高昂的计算成本；在实际应用中，训练得到的模型往往不如直接使用最先进的闭源商用大语言模型。

免训练方法旨在不修改模型参数的情况下提升生成质量。多智能体框架已被用于将RTL 代码生成拆解为协作阶段。例如，Zhao等人(2024年) 和Wu等人(2024年) 通过将规划、生成和验证组织为协调一致的阶段，提升了功能正确性。然而，这些系统主要关注语法和功能验证，缺乏对 PPA 相关信号的系统建模。电子设计自动化工具生成的丰富时序、面积和功耗信息并未被提炼为可复用知识，而是仅作为单轮反馈存在。一些研究将 PPA 约束纳入 Verilog 代码生成。例如，Tasnia等人(2025年) 和 Thorat 等人(2025年) 利用上下文学习进行 PPA 感知优化，但其效果很大程度上取决于提示工程，且缺乏自动化的经验积累机制。

为解决这些局限，我们提出了一种工具集成多智能体框架，用于生成高质量的 RTL 代码，并配备了可演化结构化记忆机制，以实现持续改进。首先，我们设计了 PPA感知工具集成多智能体工作流，将电子设计自动化工具明确纳入闭环生成流程。该系统由三个协同工作的智能体组成：负责代码生成的程序员智能体 、负责运行测试平台的正确性智能体 ，以及负责调用综合工具并分析PPA指标的*PPA* 智能体 。这一设计实现了对功能正确性和物理设计目标的联合优化。此外，我们引入了演化记忆机制，用于在迭代设计周期中积累并优化优化经验。我们将记忆划分为三种互补类型：规则记忆，用于捕获通用设计约束和领域最佳实践；结构记忆，用于从代码结构中提炼可复用的优化模式；以及*EDA*信号记忆，用于编码从PPA评估过程中获得的优化知识。为了系统地管理这些经验，我们设计了专用记忆管理器，从多智能体工作流的执行轨迹中提取记忆节点。记忆管理器通过丢弃、插入和更新 操作维护记忆池，使系统能够保留有效策略、纳入新发现，并优化欠佳的指导。每个记忆节点都以明确的触发条件和可执行指导进行结构化定义，从而能够在后续迭代中精准激活。通过以这种结构化方式外化并演化优化经验，我们的框架将RTL代码生成从孤立的一次性推理转变为持续的反馈驱动优化过程。

我们在 VerilogEval 和 RTLLM 基准测试上进行了全面评估。实验结果表明，我们的方法在保持较高功能正确性的同时，显著改善了 PPA 指标，验证了工具集成和记忆演化的有效性。总体而言，这项工作为 PPA 感知的 RTL 代码生成提出了一种新范式，具备持续优化能力，为将大语言模型融入真实世界的硬件设计工作流提供了切实可行的途径。

我们的贡献总结如下：

•我们提出了一种面向功耗、性能、面积且与工具集成的多智能体自动化 Verilog 生成框架，通过协调程序员智能体、正确性智能体和 PPA 智能体与 EDA 工具链协同工作，建立闭环优化流程，共同优化功能正确性和物理设计指标。

• 我们引入了一种演进式结构化记忆机制，用于跨迭代积累和优化经验。该机制由 RuleMemory、StructureMemory 和 EDA信号记忆组成，使系统能够将一次性 RTL 代码生成转变为持续的反馈驱动优化流程。

• 我们在 VerilogEval 和 RTLLM 基准测试上进行了大量实验，结果表明，我们的方法在保持较强功能正确性的同时，显著改善了 PPA指标，验证了工具集成优化和记忆演化的有效性。

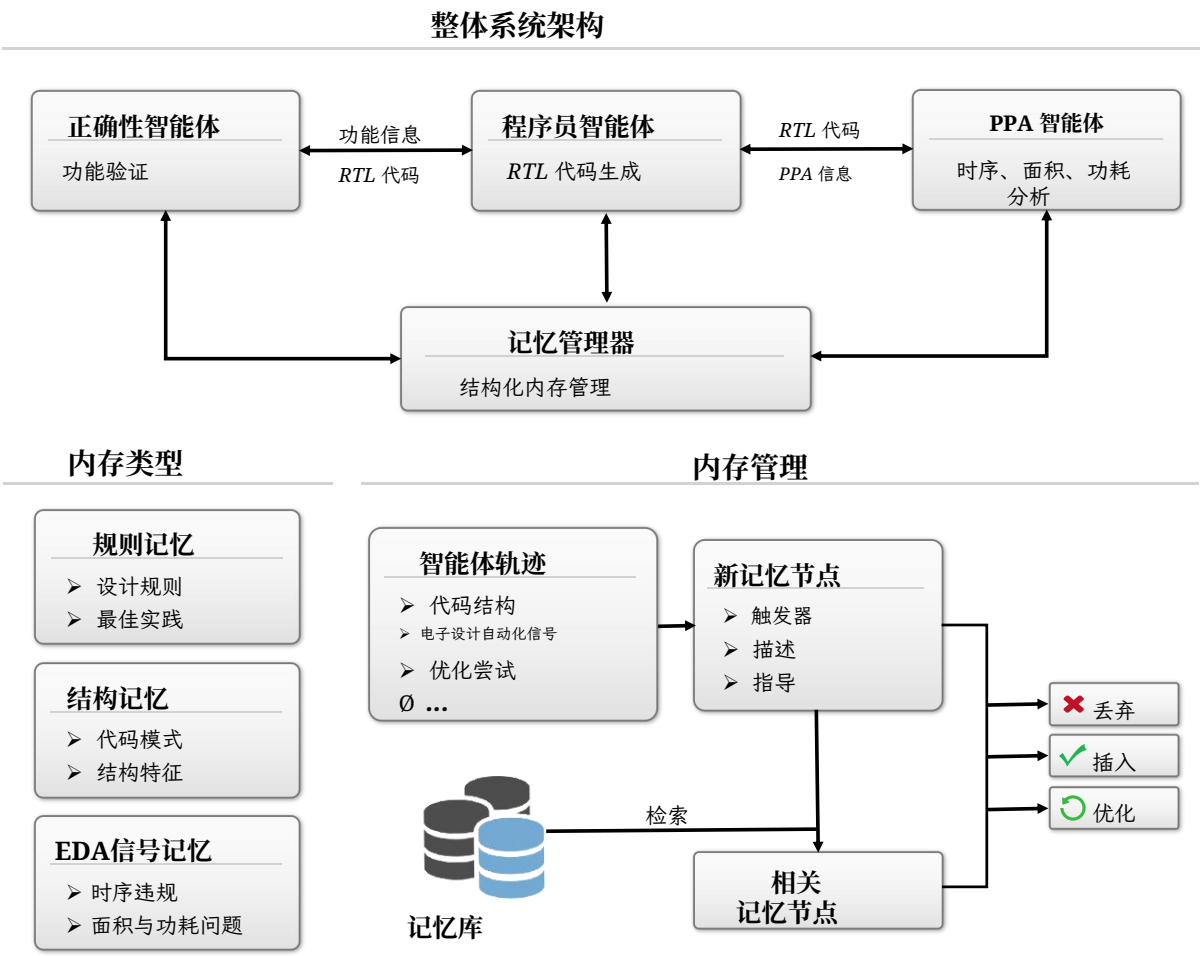


图1：所提出的 PPA感知的多智能体RTL代码生成框架概览。

2 方法论

2.1 总体框架

如图所示1，我们的系统由四个紧密耦合的组件构成：程序员智能体、正确性智能体、PPA 智能体和记忆管理器。程序员智能体根据设计规范生成 Verilog实现。正确性智能体通过仿真和测试平台执行功能验证。满足功能正确性后，PPA 智能体会调用综合工具来评估时序、面积和功耗指标。记忆管理器监控整个执行轨迹，提取可复用的优化经验，并维护结构化记忆池。

这些组件构成了一个闭环工作流。在每次迭代中，程序员智能体在已激活记忆的指导下生成或优化 Verilog代码。随后，系统将验证和 PPA分析的反馈提炼为结构化经验并加以存储，从而影响后续迭代。通过这一机制，系统逐步提升正确性和物理质量，同时无需修改模型参数。

2.2 多智能体协作

程序员智能体。程序员智能体负责 Verilog代码生成。与以往的单轮提示方法不同，我们的程序员智能体会根据从先前迭代中提取的结构化记忆指导来生成代码。这样，智能体便能将积累的优化知识（例如结构优化策略或面向PPA的设计模式）融入当前的生成步骤。因此，代码生成会逐步吸收历史优化经验。

正确性智能体。 正确性智能体通过运行测试平台并分析仿真结果，确保功能有效性。该智能体从原始错误日志中提取关键信息，并将其概括为结构化语义信号，

例如接口不匹配、状态转换不一致或算术逻辑错误。这些结构化信号不仅能指导即时调试，还可作为记忆提取过程的输入，使系统能够将反复出现的功能问题归纳为可复用知识。

PPA 智能体。 PPA 智能体将综合工具集成到优化循环中。功能验证通过后，该智能体会执行综合，并从日志和报告中提取关键指标，包括关键路径延迟、单元面积和估算功耗。报告还可归纳为更高层次的语义信号，例如组合逻辑深度过大和面积异常增长。这种语义抽象连接了底层工具报告与高层推理，从而能够系统地建模物理反馈。

2.3 演进式结构化记忆

2.3.1 记忆结构

我们的框架的一项关键贡献，是引入了一种演进式结构化记忆机制，将优化知识从迭代执行轨迹中提取并外化。每个记忆单元都表示为一个结构化记忆节点，由触发条件、指导和元数据组成。触发条件用于指定记忆变得相关时所满足的语义条件，这些条件可能源自代码结构或电子设计自动化反馈。指导以自然语言描述可执行的优化策略。元数据记录优化意图（例如时序、面积、功耗、正确性）以及适用的智能体角色。我们将记忆分为三种互为补充的类型。

- 规则记忆。规则记忆记录了适用于各种任务的一般设计约束和领域最佳实践。这些规则编码了稳定的设计原则，例如避免过深的组合逻辑路径或确保正确的复位处理。规则记忆提供全局指导，并在广泛条件下保持激活状态。

- 结构记忆。结构记忆由从 RTL 代码中提取的结构特征触发。通过静态分析或模式检测，系统可识别宽数据通路、深层嵌套的条件逻辑或重复的算术结构等特征。这些结构模式与相应的优化策略相关联使系统能够学习代码形式与改进操作之间可复用的映射关系。

- EDA信号记忆。EDA信号记忆由从综合及时序分析中提取的语义信号触发。例如，负时序裕量可能激活流水线策略，而过大的面积可能触发资源共享或逻辑重构建议。通过编码此类关系，该框架将暂时性的工具反馈转化为持久的优化知识。

2.3.2 记忆触发

在每个生成步骤之前，记忆管理器会根据当前任务上下文确定应激活哪些记忆节点。该上下文由设计规范、当前RTL代码和最新的执行反馈组成。记忆管理器会在此上下文与存储于记忆池中的记忆节点的触发字段之间执行语义匹配。具体而言，系统会将上下文描述与所有候选触发条件一并提供给大语言模型，由其充当语义选择器，识别最相关的触发条件和记忆节点。随后，选中的记忆节点将被激活，其指导字段会注入智能体的提示词中，以指导下一步生成。

2.3.3 记忆演化

除了在生成过程中激活记忆之外，记忆管理器还会在每次任务执行后持续演化记忆池。此过程旨在将执行轨迹转化为可复用的优化知识，并将其整合到结构化记忆库中。

具体而言，执行轨迹 T 包含当前任务的完整交互历史，包括设计规范、生成的 RTL 代码、修改步骤、仿真反馈和功耗、性能、面积评估结果。基于该轨迹，记忆管理器会提取潜在的优化见解，并将其转化为候选节点。每个候选节点都包含触发条件、优化指导和相关元数据。

详细的演化过程总结于算法 1 中。首先，管理器分析轨迹，并通过函数生成一组候选记忆节点 N

算法 1： 记忆演化

输入： 执行轨迹 T，记忆库 M , 检索数量 K

输出： 已更新的记忆库 M

$N \leftarrow$ 生成代码节点(T)

对 $n \in N$ **执行循环**

  $R \leftarrow$ 检索(M, $n.trigger$, K); // top-K 相似节点

  (act, m_id, $content$) $\leftarrow$ 决定(n, R) ; // 执行 = 插入/优化/丢弃

  如果 act =插入 **则**

    $M \leftarrow M \cup \{n\}$

  否则如果 act =优化**则**

    $M[m_id] \leftarrow content$

  else

    继续

返回 M

GENERATENODES(T)。对于每个候选节点n，系统根据新节点的触发条件和现有触发条件提示大语言模型，从记忆库中检索最相关的前 K 个记忆节点。这些检索到的节点提供上下文参考，以判断候选节点代表的是新知识还是对现有经验的优化。

接下来，记忆管理器会调用一个基于大语言模型的决策模块 DECIDE(n, R)，该模块会评估候选节点 n 以及检索到的记忆集合 R，并输出一项操作决策。该决策包含三种可能的操作：插入、优化或丢弃。如果候选节点捕获了此前未见过的优化模式，就会将其直接插入记忆库。如果该节点对应一条现有记忆条目，但提供了改进或更精确的指导，则会更新相应的记忆节点。否则，该候选节点会被视为冗余并予以丢弃。

通过反复执行循环，该机制逐步积累针对 RTL 代码生成和面向 PPA 的设计优化的结构化优化经验。通过将优化知识移至模型参数之外，系统将 RTL 代码生成从孤立的一次性推理转变为持续的反馈驱动优化过程。

3 实验

3.1 设置

基准测试。我们在两个具有代表性的 RTL 代码生成基准测试上评估了我们的框架：VerilogEval 和 RTLLM。

VerilogEval 旨在衡量大语言模型根据自然语言规范生成可综合 Verilog 实现的能力。它包含两条互补赛道。人工赛道由硬件专家撰写的设计描述组成，反映了真实且多样的规范风格。相比之下，机器赛道提供由模型生成的规范，这些规范通常更加标准化且歧义更少。

RTLLM通过精选的一系列复杂度各异的 RTL 设计任务，评估 Verilog代码生成质量。该基准测试包含 50 个问题，涵盖组合电路和时序电路、算术模块以及控制逻辑。以往的研究广泛采用 RTLLM v1.1 进行功能正确性评估，而 RTLLM v2.0 则用于系统性地评估功耗、性能、面积。

评估指标。对于功能正确性，我们采用 pass@k 指标。针对每项任务，模型会生成 k 个候选实现；如果至少有一个候选实现通过所有给定的测试平台，则认为该任务已成功解决。

为评估物理质量，我们遵循（Chen 等人，2025年）使用 RTLLM v2.0 进行基于综合的分析。我们报告的标准物理指标包括关键路径延迟、面积和功耗。评估流程通过 NanGate45 工艺技术仿真使用 Yosys 和 OpenROAD。

实现细节。 我们使用*GPT-4o* 和 *Gemini-3-Pro-Preview*作为骨干模型进行实验，采样温度设为 0.8。程序员智能体分别与正确性智能体和 PPA 智能体交互，最多进行两轮。记忆检索和触发通过大语言模型提示进行语义匹配，从记忆库中选取最相关的 3 个记忆节点。为降低记忆检索和触发等高频但简单操作的计算成本，我们采用较小的模型（*GPT-4o-mini*）。

表 1：在 VerilogEval 和 RTLLM v1.1 上的功能对比。每列的最佳结果以粗体突出显示。

方法	基础模型	VerilogEval-Machine (%)			VerilogEval-Human (%)			RTLLM v1.1 (%)	
		一次通过率	五次通过率	十次通过率	一次通过率	五次通过率	十次通过率	五次综合率	Func@5
基于训练									
RTLCoder	Mistral 7B	62.5	72.2	76.6	36.7	45.5	49.2	73.7	37.3
	DeepSeek-Coder 7B	61.2	76.5	81.8	41.6	50.1	53.4	83.9	40.3
BetterV	CodeLlama 7B	64.2	75.4	79.1	40.9	50.0	53.3	–	–
	DeepSeek-Coder 6.7B	67.8	79.1	84.0	45.9	53.3	57.6	–	–
	CodeQwen 7B	68.1	79.4	84.5	46.1	53.7	58.2	–	–
CodeV	CodeLlama 7B	78.1	86.0	88.5	45.2	59.5	63.8	89.2	50.3
	DeepSeek-Coder 6.7B	77.9	88.6	90.7	52.7	62.5	67.3	87.4	51.5
	CodeQwen 7B	77.6	88.2	90.7	53.2	65.1	68.5	89.5	53.3
OriGen	DeepSeek-Coder 6.7B	74.1	82.4	85.7	54.4	60.1	64.2	–	65.5
CraftRTL	CodeLlama 7B	78.1	85.5	87.8	63.1	67.8	69.7	93.9	52.9
	DeepSeek-Coder 6.7B	77.8	85.5	88.1	65.4	70.0	72.1	92.9	58.8
	Starcoder2 15B	81.9	86.9	88.1	68.0	72.4	74.6	93.9	65.8
ChipSeek-R1	Qwen2.5-Coder 7B	84.1	90.6	92.3	62.2	73.7	76.9	96.6	82.8
无需训练									
直接	GPT-4o	65.9	71.4	72.7	57.1	63.9	66.7	93.9	65.5
	Gemini3-pro-preview	88.8	92.3	92.3	91.7	94.2	94.9	100.0	–
VeriAgent	GPT-4o	86.7	90.2	90.2	78.2	85.9	89.1	100.0	72.4
	Gemini3-pro-preview	97.9	99.3	99.3	98.1	98.7	98.7	100.0	89.7

3.2 主要结果

功能正确性。 表 1 报告了 VerilogEval 和 RTLLM v1.1 上的功能正确性结果。近期基于训练的方法通过训练小型开源模型取得了出色的表现。相比之下，我们的框架直接利用现代闭源基础模型，实现了很高的正确性。这些观察结果与近期文献一致；例如，Zhao 等人 (2025b) 报告称，使用闭源模型在 VerilogEval-Human 上的一次通过率达到 94.8%。总体而言，这些结果表明，在现有基准测试上，当前基准测试的功能正确性已接近饱和。

PPA 得分 我们进一步使用 RTLLM v2.0 评估生成的寄存器传输级实现的物理质量。遵循 Chen 等人 (2025年)，表 2 报告了使用 NanGate45 工艺库获得的 PPA 结果。遵循 Chen 等人(2025年)，我们采用一种称为相对 *PPA* 分数 的综合指标来概括整体硬件效率。PPA 分数定义如下：

$$\text{PPA Score} = \frac{1}{\text{Delay} \times \text{Area} \times \text{Power}}, \quad (1)$$

该指标数值越高，表示综合硬件效率越好。相对 PPA 分数的计算方法是：将每个设计相对于基准测试提供的原始参考实现进行归一化。为提高透明度，表 2 还报告了各设计的原始延迟、面积和功耗值。总体而言，VeriAgent 在大多数单项指标上均取得最佳结果，且平均相对 PPA 分数最高。

PPA 获胜率 为进一步了解我们的优化框架的有效性，我们对 VeriAgent 与先前的优化系统进行了成对比较。VeriPPA（Thorat 等人，2025年）和 VeriOpt（Tasnia等人，2025年）均采用上下文学习（Wu等人，2023b）进行功耗、性能、面积优化。由于 VeriOpt 使用的评估方案和数据集与我们不同，且未公开详细的实现信息，因此我们将 VeriPPA 原论文报告的结果作为对比。具体而

表2：RTLLM v2.0 上的功耗、性能、面积对比。每个条目依次列出（延迟，单位为 ns / 面积，单位为 μm^2 / 功耗，单位为 W ）。不适用表示没有功能正确且可综合的设计。粗体表示最佳 PPA 分数。

设计	原始 (纳秒 / μm^2 / 瓦)	ChipSeek-R1 (纳秒 / μm^2 / 瓦)	VeriAgent（GPT-4o） (纳秒 / μm^2 / 瓦)	VeriAgent（Gemini 3） (纳秒 / μm^2 / 瓦)
accu	0.47 / 150.822 / 1.23e-05	0.46 / 210.672 / 1.83e-05	0.53 / 212.002 / 1.78e-05	**0.49 / 143.64 / 1.14e-05**
加法器_16 位	0.84 / 89.376 / 6.49e-05	0.07 / 93.632 / 4.44e-05	**0.57 / 97.888 / 2.26e-06**	**0.57 / 97.888 / 2.26e-06**
加法器_32位	0.76 / 472.15 / 0.000325	1.13 / 191.786 / 0.00012	1.08 / 195.776 / 4.52e-06	**0.51 / 240.464 / 5.6e-06**
加法器_8bit	0.35 / 51.072 / 3.14e-05	0.07 / 46.816 / 2.22e-05	**0.3 / 49.21 / 1.12e-06**	**0.3 / 49.21 / 1.12e-06**
加法器_bcd	0.34 / 46.018 / 3.85e-05	0.34 / 46.018 / 3.84e-05	**0.31 / 42.56 / 9.36e-07**	0.31 / 42.826 / 9.38e-07
加法器_流水线_64位	0.75 / 2534.182 / 0.000235	N/A	**0.1 / 756.77 / 4.43e-05**	0.14 / 894.824 / 4.99e-05
alu	1.92 / 1573.39 / 0.000751	1.75 / 1286.908 / 0.000433	N/A	**1.49 / 1202.852 / 2.73e-05**
异步_先进先出	**0.72 / 1397.032 / 7.67e-05**	N/A	N/A	N/A
桶形 _移位器	0.17 / 44.688 / 1.41e-05	0.17 / 39.368 / 1.41e-05	0.17 / 44.688 / 8.62e-07	**0.17 / 39.368 / 8.13e-07**
日历	0.44 / 164.92 / 1.44e-05	0.44 / 164.92 / 1.44e-05	0.54 / 169.708 / 1.4e-05	**0.4 / 157.738 / 1.3e-05**
比较器_3位	0.1 / 11.704 / 5.26e-06	0.11 / 11.704 / 5.25e-06	**0.1 / 11.704 / 3.03e-07**	**0.1 / 11.704 / 3.03e-07**
比较器_4位	0.16 / 18.886 / 8.91e-06	0.13 / 17.29 / 7.6e-06	0.13 / 17.29 / 4.33e-07	**0.12 / 16.492 / 3.99e-07**
计数器_12	0.25 / 36.176 / 3.1e-06	0.25 / 36.176 / 3.1e-06	0.25 / 36.176 / 2.99e-06	**0.19 / 31.92 / 2.76e-06**
div_16 位	5.18 / 760.228 / 0.027	5.57 / 745.332 / 0.0234	5.35 / 755.706 / 1.71e-05	**5.38 / 725.382 / 1.67e-05**
边沿_检测	0.12 / 18.354 / 1.79e-06	0.1 / 17.822 / 1.75e-06	0.12 / 18.354 / 1.73e-06	**0.13 / 9.31 / 7.64e-07**
fixed_point_加法器	1.69 / 606.214 / 0.000565	**0.26 / 24.294 / 2.5e-05**	0.66 / 220.248 / 5.09e-06	0.66 / 220.248 / 5.09e-06
定点_减法器	1.09 / 477.736 / 0.000381	**0.15 / 20.482 / 9.44e-06**	0.56 / 95.228 / 2.18e-06	0.56 / 95.228 / 2.18e-06
freq_除法byeven	0.25 / 40.166 / 3.63e-06	N/A	N/A	**0.17 / 20.482 / 1.87e-06**
freq_除法byfrac	0.2 / 48.678 / 4.6e-06	N/A	N/A	**0.15 / 32.452 / 3.14e-06**
freq_奇数除法	5.17 / 59.052 / 6.63e-06	5.18 / 62.776 / 6.91e-06	N/A	**0.1 / 36.708 / 3.61e-06**
指令_寄存器	0.14 / 117.04 / 1.03e-05	0.14 / 117.306 / 1.03e-05	**0.14 / 114.912 / 9.7e-06**	**0.14 / 114.912 / 9.7e-06**
JC_计数器	0.1 / 340.48 / 3.54e-05	0.1 / 340.48 / 3.54e-05	**0.1 / 340.48 / 3.41e-05**	**0.1 / 340.48 / 3.41e-05**
LFSR	**0.14 / 25.004 / 2.45e-06**	**0.14 / 25.004 / 2.45e-06**	N/A	0.15 / 24.206 / 2.51e-06
后进先出缓冲区	0.38 / 226.1 / 0.00027	0.36 / 216.79 / 0.000137	0.38 / 216.79 / 4.05e-06	**0.28 / 213.066 / 4.23e-06**
多 _16 位	2.03 / 933.394 / 7.36e-05	1.97 / 935.522 / 7.36e-05	2.14 / 881.79 / 7.04e-05	**0.79 / 553.28 / 4.48e-05**
多 _8bit	1.5 / 483.854 / 0.00085	0.79 / 373.996 / 5.45e-05	0.75 / 328.776 / 7.58e-06	**0.75 / 327.978 / 7.55e-06**
multi_pipe_4bit	0.34 / 174.762 / 1.51e-05	**0.11 / 54.546 / 1.28e-05**	0.1 / 154.546 / 1.01e-05	0.1 / 154.546 / 1.01e-05
多级_流水线_8位	0.8 / 874.608 / 7.52e-05	N/A	0.78 / 686.28 / 5.45e-05	**0.69 / 717.668 / 5.59e-05**
并行转串行	0.2 / 48.678 / 4.5e-06	0.19 / 47.082 / 4.36e-06	N/A	**0.17 / 45.486 / 4.14e-06**
pe	1.27 / 3651.382 / 0.000224	1.27 / 3651.382 / 0.000224	**1.22 / 3685.696 / 0.000104**	1.27 / 3649.786 / 0.000102
脉冲_检测	0.18 / 17.556 / 1.52e-06	**0.17 / 16.226 / 1.47e-06**	N/A	N/A
基数 2 _div	**0.59 / 414.162 / 3.39e-05**	N/A	N/A	N/A
RAM	0.25 / 635.74 / 5.56e-05	**0.19 / 475.076 / 3.92e-05**	0.21 / 479.864 / 3.68e-05	0.21 / 476.938 / 3.65e-05
right_移位器	**0.08 / 36.176 / 4.32e-06**	**0.08 / 36.176 / 4.32e-06**	**0.08 / 36.176 / 4.32e-06**	**0.08 / 36.176 / 4.32e-06**
ring_计数器	0.1 / 46.816 / 4.7e-06	0.1 / 40.964 / 4.01e-06	0.18 / 32.452 / 2.54e-06	**0.1 / 31.92 / 2.67e-06**
ROM	0.14 / 6.65 / 8.85e-07	0.14 / 6.65 / 8.85e-07	**0.14 / 6.65 / 1.48e-07**	**0.14 / 6.65 / 1.48e-07**
序列_检测器	0.15 / 36.442 / 3.35e-06	0.19 / 25.277 / 2.27e-06	N/A	**0.1 / 22.61 / 2.22e-06**
串并转换器	0.4 / 156.142 / 1.4e-05	0.28 / 157.738 / 1.39e-05	N/A	**0.28 / 157.206 / 1.34e-05**
信号_生成器	0.37 / 93.1 / 7.54e-06	0.38 / 74.214 / 6.08e-06	0.3 / 78.47 / 6.2e-06	**0.24 / 52.402 / 4.43e-06**
方波_波	0.41 / 100.282 / 8.39e-06	0.41 / 100.282 / 8.39e-06	**0.33 / 81.396 / 7.27e-06**	**0.33 / 81.396 / 7.27e-06**
sub_64 位	2.3 / 404.586 / 0.000268	2.08 / 400.862 / 0.000271	1.7 / 454.328 / 1.06e-05	**2.13 / 405.118 / 9.25e-06**
交通_信号灯	0.37 / 149.758 / 1.31e-05	0.39 / 139.916 / 1.22e-05	N/A	**0.32 / 90.174 / 7.59e-06**
向上_向下_计数器	0.7 / 217.854 / 1.74e-05	0.67 / 188.86 / 1.61e-05	0.53 / 183.54 / 1.41e-05	**0.41 / 161.994 / 1.29e-05**
宽度 _8到16	0.24 / 186.732 / 1.67e-05	**0.21 / 173.698 / 1.62e-05**	0.24 / 186.732 / 1.62e-05	0.24 / 180.348 / 1.63e-05
几何平均值	0.417 / 142 / 2.65e-05	0.329 / 112 / 1.93e-05	0.362 / 122 / 6.89e-06	0.294 / 109 / 5.72e-06
相对 PPA 分数	1.000	2.196	5.119	8.539

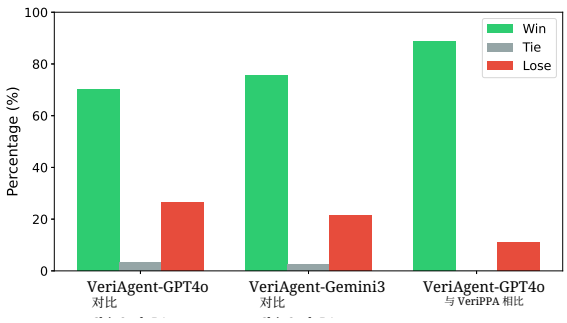


图 2：VeriAgent 与先前方法的胜率比较。

言，我们通过比较两种方法在均成功生成可综合设计的任务上的 PPA 得分，计算获胜率 。若某种方法在一项任务上的 PPA 得分更高，则认为该方法在该任务中获胜。

图 2展示了 VeriAgent 与两个具有代表性的基线：VeriPPA 和 ChipSeek-R1 的胜率对比。值得注意的是，VeriPPA（Thorat 等人，2025年）仅能成功优化一小部分任务，而我们的框架能够始终为基准测试中占比显著更大的任务生成有效实现。

表 3：VeriAgent 的消融研究。相对 PPA 分数（越高越好）。

方法	GPT-4o	Gemini-3
VeriAgent（完整版本）	5.119	8.539
不含记忆	3.839	6.541
不使用工具反馈	3.344	5.189
不使用 PPA 分析器	2.724	3.405

在重叠任务中，VeriAgent 对两个基线的胜率均更高，表明其在发现硬件高效实现方面具有更强的优化能力。

3.3 消融研究

为了解我们框架中各个组件的作用，我们通过逐步移除 VeriAgent 中的关键模块开展消融研究。评估的变体包括：

- **不使用记忆**：禁用记忆管理器。智能体仅根据当前任务上下文生成 RTL 代码，不利用历史优化指导。
- **不使用工具反馈**：禁用来自电子设计自动化工具的综合和仿真反馈。PPA 智能体仍保持启用，但在无法获取工具生成信号的情况下进行面向功耗、性能、面积的推理。
- **不含 PPA 智能体**：移除 PPA 智能体。

所有消融变体均采用与完整 VeriAgent 系统相同的基础大语言模型、提示词模板、交互轮次和评估协议。表 3展示了不同变体取得的相对 PPA 分数。消融变体采用渐进式构建方式，每种配置都在前一变体的基础上额外移除一个组件。

移除记忆模块会导致性能明显下降，表明积累的设计经验在指导优化方面发挥着重要作用。理想情况下，系统可以依赖由硬件专家精心整理的记忆库；而我们的记忆演化机制会在执行过程中自动构建并完善这类知识，使智能体能够重用有效的优化策略。禁用工具反馈会进一步降低性能，这凸显了综合和仿真信号在引导搜索过程中的重要性。缺少这些信号，智能体就无法可靠地评估候选实现并识别优化机会。这些结果表明，VeriAgent 的每个组件都对整体效能有所贡献。

4 相关工作

4.1 RTL 代码生成

自动化 RTL 代码生成（Zhang 等人，2016年）长期以来一直是电子设计自动化领域的一项核心目标。传统方法（Takamaeda-Yamazaki，2015；Amjad 等人，2019）通常依赖于根据高层规范或特定领域硬件描述语言进行基于规则的综合（Palnitkar，2003；Ashenden，2007）。这些技术在结构良好的场景中效果显著，但在处理各种自然语言规范时往往缺乏灵活性。

随着大语言模型的兴起，研究人员开始探索根据自然语言描述直接生成 RTL 代码。现有的基于大语言模型的 RTL 代码生成方法，通常通过在大规模真实数据集和合成数据集上进行监督微调（Liu 等，2024；Pei 等，2024；Zhao 等，2025a；Cui 等，2024）或强化学习（Chen 等人，2025；Zhang 等，2025；Li 等，2025），使大语言模型适应硬件描述任务（Wang 等，2025a）。尽管这些方法提升了特定任务的性能，但生成的数据集往往存在显著的分布偏差，限制了其对复杂现实设计的泛化能力。无需训练的方法（Ho 等，2025；Zhao 等，2025b）则利用提示策略、迭代反馈以及与商用大语言模型的集成，而不修改模型参数。近年来，少数研究开始考虑物理设计目标：VeriOpt（Tasnia等人，2025）和 VeriPPA（Thorat 等人，2025）将综合工具产生的 PPA相关信号纳入生成过程。然而，这些方法通常将工具反馈视为一次性信息，缺乏在多次迭代中保留和复用优化知识的机制。相比之下，我们的框架将结构模式和 EDA派生信号显式外化为可复用的结构化记忆，从而实现持续的反馈驱动优化，协同改善 PPA指标。

4.2 面向大语言模型的多智能体系统

大语言模型的最新进展引发了人们对多智能体系统日益浓厚的兴趣，这类系统协调多个基于大语言模型的智能体来解决复杂任务（Wang 等人，2025b）。这些框架不再依赖单次模型调用，而是将问题分解为规划、生成、评议和验证等专门角色，从而实现协作推理与迭代完善。多项研究探索了大语言模型智能体之间基于角色的协作。？ 提出了 ChatDev，这是一种多智能体框架，将软件工程工作流编码为标准化操作流程，使不同角色的智能体能够协作完成复杂任务。类似地，AutoGen（Wu等人，2023a）提供了一个通用框架，通过多智能体对话构建大语言模型应用，支持智能体、人类和外部工具之间的灵活交互。这些系统表明，结构化协作能够显著提升可靠性并改进任务分解。另一个研究方向则着重通过智能体交互提升推理质量。Reflexion（Shinn 等人，2023）引入了一种自我反思机制，智能体根据先前尝试生成的文本反馈迭代改进输出。多智能体辩论（Liang 等人，2024）通过允许多个智能体在给出最终解决方案前相互辩论和评议，拓展了这一思路，促使其探索多样化的推理路径。

5 结论

在本研究中，我们提出了一种面向功耗、性能、面积的工具集成多智能体框架，用于自动化 Verilog 生成。该系统在闭环工作流中协调程序员智能体、正确性智能体和 PPA 智能体，通过直接与电子设计自动化工具交互，联合优化功能正确性和物理设计指标。此外，我们引入了一种演化的结构化记忆机制，在多轮迭代中持续积累并优化经验，从而实现超越一次性生成的持续改进。在 VerilogEval 和 RTLLM 上的实验结果表明，我们的方法在保持较强功能正确性的同时，显著提升了功耗、性能、面积表现。这些结果凸显了工具集成与记忆演化相结合的潜力，有望实现由大语言模型驱动的实用硬件设计自动化。

参考文献

Josh Achiam、Steven Adler、Sandhini Agarwal、Lama Ahmad、Ilge Akkaya、Florencia Leoni Aleman、Diogo Almeida、Janko Altenschmidt、Sam Altman、Shyamal Anadkat及另外1位作者。2023年。GPT-4 技术报告。*arXiv* 预印本 *arXiv:2303.08774*。

Hafiz Muhammad Amjad、Jianwei Niu、Kai Hu、Naveed Akram 和 Loïc Besnard。2019。基于信号语言的 Verilog代码生成方案。载于*2019*年第*16*届布尔班应用科学与技术国际会议（*IBCAST*），第457–462页。IEEE。

Peter J Ashenden。2007。数字设计（*Verilog*）：采用 *Verilog* 的嵌入式系统方法。爱思唯尔。

Kaiyan Chang、Ying Wang、Haimeng Ren、Mengdi Wang、Shengwen Liang、Yinhe Han、Huawei Li 和 Xiaowei Li。2023年。Chipgpt：我们距离自然语言硬件设计还有多远。*arXiv* 预印本 *arXiv:2305.14019*。

Zhirong Chen、Kaiyan Chang、Zhuolin Li、Xinyang He、Chujie Chen、Cangyuan Li、Mengdi Wang、Haobo Xu、Yinhe Han 和 Ying Wang。2025年。Chipseek- r1：通过分层奖励驱动强化学习，利用大语言模型生成超越人类水平的寄存器传输级代码。预印本，arXiv:2507.04736。

Fan Cui、Chenyang Yin、Kexing Zhou、Youwei Xiao、Guangyu Sun、Qiang Xu、Qipeng Guo、Yun Liang、Xingcheng Zhang、Demin Song 及其他 1 位作者。2024年。Origen：通过代码到代码增强和自我反思提升寄存器传输级代码生成。发表于第 *43* 届 *IEEE/ACM* 国际计算机辅助设计会议论文集，第 1–9 页。

Daya Guo、Dejian Yang、Haowei Zhang、J unxiao Song、Peiyi Wang、Qihao Zhu、Runxin Xu、Ruoyu Zhang、Shirong Ma、Xiao Bi等1人。2025年。Deepseek-r1：通过强化学习激励大语言模型中的推理能力。*arXiv*预印本 *arXiv:2501.12948*。

Chia-Tung Ho、Haoxing Ren和Brucek Khailany。2025年。Verilogcoder：借助基于图的规划和基于抽象语法树（ast）的波形跟踪工具实现自主Verilog编码智能体。在*AAAI*人工智能会议论文集中，第39卷，第300–307页。

Shangzhan Li、Zefan Wang、Ye He、Yuxuan Li、Qi Shi、Jianling Li、Yonggang Hu、Wanxiang Che、Xu Han、Zhiyuan Liu等1人。2025年。Autotriton：利用强化学习和大语言模型实现自动Triton编程。*arXiv*预印本 *arXiv:2507.05687*。

Tian Liang、Zhiwei He、Wenxiang Jiao、Xing Wang、Yan Wang、Rui Wang、Yujiu Yang、Shuming Shi 和 Zhaopeng Tu。2024年。通过多智能体辩论激发大语言模型的发散性思维。

收录于 *2024*年自然语言处理实证方法会议论文集，第 17889–17904 页。

Shang Liu、Wenji Fang、Yao Lu、Jing Wang、Qijun Zhang、Hongce Zhang 和 Zhiyao Xie。2024年。Rtl-coder：完全开源且高效的大语言模型辅助的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，44(4)：1448–1461。

Samir Palnitkar。2003年。*Verilog*硬件描述语言：数字设计与综合指南，第 1 卷。Prentice Hall Professional。

Zehua Pei、Hui-Ling Zhen、Mingxuan Yuan、Yu Huang 和 Bei Yu。2024年。BetterV：利用判别式引导进行受控 Verilog 生成。*arXiv* 预印本 *arXiv:2402.03375*。

Noah Shinn、Federico Cassano、Ashwin Gopinath、Karthik Narasimhan 和 Shunyu Yao。2023年。Reflexion：通过语言智能体进行口头强化学习。神经信息处理系统进展，36：8634–8652。

Shinya Takamaeda-Yamazaki. 2015. Pyverilog：一种基于 Python 的 Verilog 硬件设计处理工具包。载于《应用可重构计算国际研讨会》，第 451–460 页。施普林格。

Kimia Tasnia、Alexander Garcia、Tasnuva Farheen 和 Sazadur Rahman。2025年。Veriopt：通过多角色大语言模型实现面向功耗、性能、面积的高质量 Verilog代码生成。载于*2025*年 *IEEE/ACM* 计算机辅助设计国际会议（*ICCAD*），第 1–9 页。IEEE。

Gemini团队、Rohan Anil、Sebastian Borgeaud、Jean-Baptiste Alayrac、Jiahui Yu、Radu Soricut、Johan Schalkwyk、Andrew M Dai、Anja Hauth、Katie Millican 和另外 1 位作者。2023年。Gemini：一系列能力强大的多模态模型。*arXiv* 预印本 *arXiv:2312.11805*。

Kiran Thorat、Jiahui Zhao、Yaotian Liu、Amit Hasan、Hongwu Peng、Xi Xie、Bin Lei 和 Cai wen Ding。2025年。Llm-verippa：面向功耗、性能和面积优化的、由大语言模型驱动的 Verilog代码生成。载于*2025*年 *ACM/IEEE* 第七届 *CAD* 机器学习研讨会（*MLCAD*），第 1–7 页。IEEE。

Yaoxiang Wang、Haoling Li、Xin Zhang、Jie Wu、Xiao Liu、Wenxiang Hu、Zhongxin Guo、Yangyu Huang、Ying Xin、Yujiu Yang 等。2025a。Epicoder：涵盖代码生成中的多样性与复杂性。*arXiv* 预印本 *arXiv:2501.04694*。

Yaoxiang Wang、Zhiyong Wu、Junfeng Yao 和 Jinsong Su。2025b。Tdag：一种基于动态任务分解和智能体生成的多智能体框架。《神经网络》，185:107200。

Haoyuan Wu、Zhuolun He、Xinyun Zhang、Xufeng Yao、SuZheng、Haisheng Zheng 和 Bei Yu。2024年。Chateda：一种由大语言模型驱动的电子设计自动化自主智能体。*IEEE* 计算机辅助设计集成电路与系统汇刊，43(10)：3184–3197。

Qingyun Wu、Gagan Bansal、Jieyu Zhang、Yiran Wu、Beibin Li、Erkang Zhu、Li Jiang、Xiaoyun Zhang、Shaokun Zhang、JialeLiu、Ahmed HassanAwadal- lah、Ryen W White、Doug Burger 和 Chi Wang。2023a。AutoGen：通过多智能体对话赋能新一代大语言模型应用。预印本，arXiv:2308.08155。

Zhenyu Wu、Yaoxiang Wang、Jiacheng Ye、Zhiyong Wu、Jiangtao Feng、Jingjing Xu 和 Yu Qiao。2023b。OpenICL：用于上下文学习的开源框架。载于第*61*届计算语言学协会年会论文集（第*3*卷：系统演示），第489–498页。

An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv 等另外1位作者。2025a。Qwen3技术报告。*arXiv*预印本 *arXiv:2505.09388*。

Guang Yang、Wei Zheng、Xiang Chen、Dong Liang、Peng Hu、Yukui Yang、Shaohang Peng、Zhenghan Li、Jiahui Feng、Xiao Wei 等人。2025b。用于 Verilog 代码生成的大语言模型：文献综述与未来展望。*arXiv* 预印本 *arXiv:2512.00020*。

Yang Zhang、Rui Zhang、Jiaming Guo、Lei Huang、Di Huang、Yunpu Zhao、Shuyao Cheng、Pengwei Jin、Chongxiao Li、Zidong Du 等人。2025年。Qimeng-salv：面向 Verilog 代码生成的信号感知学习。*arXiv* 预印本 *arXiv:2510.19296*。

Yu Zhang、Wenlong Feng 和 Mengxing Huang。2016年。利用软件技术与工具为寄存器传输级设计自动生成高覆盖率测试。在*2016*年 *IEEE* 第 *11* 届工业电子与应用会议（*ICIEA*），第 856–861 页。IEEE。

Yang Zhao、Di Huang、Chongxiao Li、Pengwei Jin、Muxin Song、Yinan Xu、Ziyuan Nan、Mingju Gao、Tianyun Ma、Lei Qi 等。2025a。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* 计算机辅助设计集成电路与系统汇刊。

Yujie Zhao、Hejia Zhang、Hanxian Huang、Zhong ming Yu 和 Jishen Zhao。2024。Mage：用于自动化 RTL 代码生成的多智能体引擎。预印本，arXiv:2412.07822。

Yujie Zhao、Hejia Zhang、Hanxian Huang、Zhongming Yu 和 Jishen Zhao。2025b。Mage：用于自动化 RTL 代码生成的多智能体引擎。载于*2025*年第*62*届 *ACM/IEEE* 设计自动化会议 *(DAC)*，第 1–7 页。IEEE。