EvolVE：基于大语言模型的 Verilog 生成与优化进化搜索

Wei-Po Hsin*
国立台湾大学电机工程系
b10901053@ntu.edu.tw

Ren-Hao Deng*
国立台湾大学计算机科学与信息工程系
r13922212@csie.ntu.edu.tw

Yao-Ting, Hsieh*
中央研究院信息科学研究所，台湾 elton731888@gmail.com

En-Ming Huang
国立台湾大学计算机科学与信息工程学系
r13922078@csie.ntu.edu.tw

Shih-Hao Hung†
国立台湾大学计算机科学与信息工程学系
hungsh@csie.ntu.edu.tw

arXiv:2601.18067v1 [cs.AI] 26 Jan 2026

摘要

Verilog 的设计周期本质上耗费大量人力，且需要丰富的领域专业知识。尽管大语言模型（LLM）为自动化提供了一条有希望的路径，但其训练数据有限，且固有的序列推理能力无法捕捉硬件系统所固有的严格形式逻辑与并发性。为克服这些障碍，我们提出**EvolVE**，这是首个针对芯片设计任务分析多种进化策略的框架。研究表明，蒙特卡洛树搜索（MCTS）在提升功能正确性方面表现出色，而思想引导的改进（IGR）则在优化方面更具优势。我们还利用结构化测试平台生成（STG）来加速进化过程。为解决复杂优化基准测试的缺乏问题，我们推出**IC-RTL**，其目标是源自全国集成电路竞赛的工业级问题。评估结果表明，EvolVE 达到了新的最先进水平，在 VerilogEval v2 上取得 98.1% 的成绩，在 RTLLM v2 上取得 92% 的成绩。此外，在工业级 IC-RTL 套件上，我们的框架优于竞赛参与者编写的参考实现：在霍夫曼编码问题中，功耗、性能和面积（PPA）乘积最多降低 66%，在所有问题上的几何平均数降低 17%。IC-RTL 基准测试的源代码可在 https://github.com/weiber2002/ICRTL 获取。[1]

关键词 演化框架 · Verilog 代码生成 · PPA优化 · 大语言模型辅助设计

1 引言

随着数字硬件日益复杂，业界迫切需要缩短开发周期，同时不牺牲电路性能。尽管 Verilog 等硬件描述语言（HDL）是对同步电路进行精细控制的标准方式，但仅依赖手动 RTL 编码已成为显著瓶颈，因为这一过程本质上耗费大量人力且容易导致延误。为克服这些局限，一种颇具前景的新范式应运而生：利用大语言模型自动化并加速硬件设计与验证。

从根本上说，大语言模型是基于海量文本和代码语料训练而成的强大概率模型。其本质是学习复杂的统计模式，以预测序列中的下一个词元；这种能力使其能够进行合乎逻辑、循序渐进的推理。近期的进展（Novikov 等，2025年；Yuan 等，2025年）进一步

*贡献相同
†通讯作者[1]EvolVE 代码库、Mod-VerilogEval v2 和评估脚本即将发布。

通过先进的推理技术和进化框架增强了这一潜力，使智能体能够反复改进输出并解决复杂的逻辑问题。然而，这种序列推理能力与硬件设计的要求并不相符。Verilog 等硬件描述语言描述的是高度并发的系统，其中数千项操作并行执行，并由时钟同步。这种严格的并发范式对基于时序逻辑训练的概率模型构成了重大障碍。因此，将大语言模型直接用于硬件描述语言生成，往往会产生功能错误或无法综合的 Verilog。这一性能差距凸显了仅靠基础模型的不足；集成电路设计的真正前景在于新型框架，它们能够使大语言模型推理并发性、正式探索设计空间，并反复优化结果。

基于这一认识，近期出现了一些旨在解决独立大语言模型在硬件设计中局限性的研究。这些工作主要分为两条路线。第一条路线聚焦于 RTL 专业化，通过在 Verilog 专用数据集（包括 Pyranet（Nadimi 等人，2025年）和 MGVerilog（Zhang 等人，2024年））上进行监督微调或强化学习，创建 RTLCoder（Liu 等人，2024a）、ScaleRTL（Deng 等人，2025年）和 CodeV-R1（Zhu 等人，2025年）等专家模型。第二条较新的路线采用多智能体系统，将复杂的设计过程拆解开来。这些框架协调大语言模型承担不同角色，例如利用中间表示（Wang 等人，2025c；Yao 等人，2024年）、探索多样化的设计候选方案（Zhao 等人，2025年），或管理由专业智能体组成的分层流水线（Tasnia 等人，2025年；Wang 等人，2025a）。其中一些工作甚至会优化工作流本身（Wei 等人，2025年）。

尽管取得了这些进展，准确评估仍然是瓶颈。为评估方法的有效性，研究社区建立了标准化基准测试，例如用于自然语言规范到 RTL 生成的 RTLLM v2（Liu 等，2024b）。与此同时，VerilogEval v2 项目（Pinckney 等，2024）将原始基准测试从代码补全扩展至规范到 RTL 任务。然而，我们的分析发现，VerilogEval v2 的黄金模型和问题规格存在不一致，降低了评估精度。此外，我们发现 RTLLM v2 缺乏工业设计的复杂性，无法对模型在复杂优化技术方面提出挑战。为弥补这些验证缺口，我们推出了**Mod-VerilogEvalv2**，这是一个经过修正的标准基准测试，详见附录 B。此外，我们还推出了**IC-RTL**，这是一个高复杂度测试套件，基于台湾全国集成电路设计竞赛（台湾教育部，2025年）以及由专家设计的定制设计构建而成。

为应对自动化 Verilog 生成面临的挑战，鉴于领域专用训练数据稀缺，且固定容量模型存在固有的推理局限性，我们提出了一种无需微调的灵活框架，利用进化算法统一设计探索与调试，有效模拟硬件设计中自然形成的迭代精炼过程。为缓解进化框架通常伴随的计算延迟，我们提出了**结构化测试平台生成（STG）**。

STG 通过自动对关键信号进行分类以生成严格的测试向量，从而提供高覆盖率的测试用例和细粒度反馈，超越简单的二元结果。这一机制有效地将迭代调试的高昂计算成本，从大语言模型的复杂推理转移到高效且免费的电子设计自动化仿真工具上，从而同时加快进化过程并显著增强框架的能力。本文的主要贡献总结如下：

- **EvolVE 框架：**我们提出了一种采用两种不同搜索策略的进化框架：创意引导精炼（IGR）和蒙特卡洛树搜索（MCTS）。通过整合这些搜索机制，我们的框架使大语言模型能够推理并发性并迭代探索设计空间，而无需依赖庞大的领域专用数据集或要求模型规模大到难以承受。该方法有助于实现有针对性的自我纠正，使 DeepSeek-R1-FP4 在 RTLLM v2 上达到 92%，在 VerilogEval v2 上达到 98.1%；同时也提升了较小模型的性能，例如 Siliconmind-7B 在后者上达到 92.3%。

- **IC-RTL 与 Verilog 优化：**为了解决缺乏对功耗、性能和面积进行严格评估的问题，我们进一步在新颖的 IC-RTL 基准测试套件上验证了我们的框架。在这一高复杂度环境中，我们的框架优于竞赛参与者编写的参考实现，在 Q5 中将功耗、性能和面积的乘积最多降低了 66%，在所有问题上的几何平均数降低了 17%。

- **结构化测试平台生成（STG）：**我们开发了 STG，这是一种可拆卸的自动化高覆盖率验证引擎，旨在加速演进周期。STG 可自动对信号进行分类并提供细粒度反馈，将迭代调试的成本转移给免费的电子设计自动化工具。

2 背景与动机

2.1 硬件描述语言

硬件描述语言弥合了行为意图与物理实现之间的差距。目前，业界在芯片设计中高度依赖 Verilog，因为它能够提供精细优化所需的粒度，确保综合电路满足严格的性能、功耗和面积（PPA）目标。此外，Verilog 的广泛应用形成了大量现有设计，这些设计的数量远超 Chisel 和 SystemC 等较新的高级综合语言。相比之下，Verilog 数据的丰富程度使其成为训练定制化大语言模型以生成硬件描述语言，以及利用大语言模型执行自动化 PPA 优化的理想选择。

2.2 用于代码生成的大语言模型

利用大语言模型作为自动化编程智能体，已发展成为一项重要的研究方向。Microsoft Copilot 和 Aider（Gauthier，2025年）等早期工具展现了处理 Python 和 C 等过程式语言的出色能力。然而，尽管这些单轮智能体在标准基准测试中表现良好，却往往难以应对项目级工程的复杂性和长周期问题求解。为克服这些局限，研究转向了更复杂的多智能体框架。MetaGPT（Hong 等人，2024年）和 AgileCoder（Nguyen 等人，2024年）等系统模拟人类的敏捷方法，将不同角色（例如架构师、工程师、质量保证人员）分配给不同的智能体，以实现协作式软件开发。

近期，自动化科学发现领域涌现出进化式编码智能体。AlphaEvolve（Novikov 等人，2025年）和 Dolphin（Yuan 等人，2025年）等框架通过利用闭环优化，突破了静态提示的局限。在这些系统中，智能体会反复进行推理、生成实现代码，并利用执行反馈来改进解决方案。例如，AlphaEvolve 采用进化流程来发现新颖算法，而 Dolphin 则将想法生成、实践和反馈构成的研究循环自动化。从生成转向迭代演化，为解决硬件设计的严格约束提供了基础方法。

2.3 用于 Verilog 生成的大语言模型

尽管大语言模型擅长逐步推理，但它们难以建模 Verilog 所需的大规模并行性和周期精确同步。为弥合这一差距，近期研究主要采用两种策略：通过微调进行领域适配，以及通过智能体框架提供结构化指导。

领域专用微调。 该方法将硬件领域知识直接嵌入模型权重中。RTLCoder（Liu 等人，2024a）通过采用 GPT 辅助的反馈质量方案生成 Verilog 数据对，建立了基线，其性能可与 GPT-4 相媲美。ScaleRTL（Deng 等人，2025）进一步利用思维链（Wei 等人，2022）推理轨迹，对模型进行微调以处理复杂逻辑。近期研究还引入了强化学习（RL）。VeriReason（Wang 等人，2025b）采用组相对策略优化（Shao 等人，2024）来增强推理能力，而 CodeV-R1（Zhu 等人，2025）则使用测试平台生成器进行往返数据合成。然而，尽管具备这些专用架构，微调模型仍常常无法持续超越顶尖专有模型，原因在于模型规模受到限制，而且与软件语言可获得的海量语料库相比，高质量开源 Verilog 训练数据依然长期稀缺。

智能体与搜索　　驱动的框架。 与微调并行，第二条路径通过构建通用大语言模型的　　s 问题解决工作流　　来引导其工作。这些方法可分为三类：

- **优化与调试：** 早期框架专注于局部代码修正。RTLRewriter（Yao 等，2024）利用成本感知蒙特卡洛树搜索（C-MCTS）重写 Verilog，而 SymRTLO（Wang 等，2025c）采用使用抽象语法树（AST）的神经符号框架来强制满足 PPA约束。
- **协作式多智能体系统**：受人类设计团队的启发，这些框架将复杂规格分解为易于处理的子任务。MAGE（Zhao 等人，2025年）采用专门的智能体，并通过检查点机制跟踪状态。RTLSquad（Wang 等人，2025年a）将工作流划分为探索、实现和评估小组，以生成可解释的决策路径；VeriOpt（Tasnia 等人，2025年）则通过基于角色的提示整合 PPA约束。VerilogCoder（Ho 等人，2025年）利用任务与电路关系图（TCRG）将模块描述分解为子任务，并结合基于抽象语法树的波形跟踪工具，使智能体能够自主调试并修复功能错误，从而进一步推进了这一方法。

- **工作流与进化搜索**：近期的进展将生成视为全局搜索问题。VFlow（Wei 等人，2025年）优化智能体工作流本身，并采用协同进化（CEPE-MCTS）来发现大语言模型调用的最佳序列。类似地，REvolution（Min 等人，2025年）采用双种群算法和进化式提示选择，对 Verilog 设计进行生成与优化。

2.4 PPA 基准测试的复杂性差距

目前，优化工作主要针对 RTLLM v2 基准测试，例如 REvolution（Min 等人，2025年）中的相关工作。然而，这些工作往往局限于小规模模块，而标准综合工具无需深入的领域知识即可轻松优化这些模块。为了让自动化设计适用于学术界和工业界，亟需复杂的大规模基准测试。此类基准测试必须既支持商业电子设计自动化工具，以便进行精确评估，又支持开源工具，以提高普及性；同时还应提供初始参考设计，以严格评估不同框架的实际硬件优化能力。

2.5 当前局限性与我们的方法

尽管微调模型展现出令人瞩目的能力，但其仍受限于模型容量不足，以及高质量开源 Verilog 数据匮乏。近期一项调查（Yang 等人，2025年）进一步印证了这些担忧，并指出了具体障碍：现有基准测试规模较小（<100 个样本），且侧重于基础模块，忽视了系统级复杂性。此外，当前指标通常仅依赖功能通过率（功能通过率@k），忽略了功耗、性能和面积等关键维度。最后，依赖单轮生成会限制架构探索，也会降低从逻辑故障中恢复的能力。

为克服数据稀缺和模型规模有限这两大瓶颈，我们将范式从以学习为中心转向基于搜索的范式。由于高质量 Verilog 数据具有专有性且数量有限，而在数据量没有相应增长的情况下，增加模型参数的收益会逐渐递减，因此我们转而利用进化搜索来发挥测试时计算资源的作用。这种方法将硬件描述语言生成视为状态空间探索，而非一次性过程。它使大语言模型能够针对严格的逻辑约束反复自我纠正，从而有效模拟人工调试过程。此外，为解决第 2.4 节指出的评估缺口，验证必须超越简单的功能通过率，采用能够评估优化指标的系统级基准测试。最后，迭代进化搜索的效果取决于高质量反馈。高质量反馈有助于更准确地调试，进而推动进化过程更快收敛。

3 方法论

鉴于 Verilog 生成的复杂性既要求广泛探索，也需要精确调试，我们设计了两种截然不同的算法策略：思想引导的改进（IGR）和蒙特卡洛树搜索（MCTS）。IGR 策略通过整合大语言模型的内部知识或参考论文中的知识，将想法生成与实现分离。MCTS 利用基于树的搜索探索设计空间。我们选择这两种独立策略，以实现互补的优化目标：IGR 用于确保对架构空间进行全局探索，以摆脱局部最优并实现 PPA 优化；MCTS 则用于精确利用搜索结果，在 Verilog 代码生成过程中严格解决局部功能约束和时序约束。本节将形式化定义我们框架中的 Verilog 生成与优化任务，概述核心算法和优化目标，并介绍能够显著提升性能的策略。

3.1 问题定义

我们将 Verilog 开发过程表述为一个搜索优化问题。生成过程会探索一组节点 $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$。第 i 个节点的状态定义为一个元组 $N_i = (V_i,\ S_i,\ F_i)$，其中 V_i 是候选 Verilog 代码，$S_i \in \mathbb{R}$ 是分数，F_i 是诊断反馈。

代码 V_i 通过测试平台 T 进行验证，该测试平台定义为一组互不相同的测试用例 $T = \{t_1, t_2, \ldots, t_{|T|}\}$。评估函数 E 会执行这些测试，以得出分数和反馈：

$$(S_i, F_i) = E(V_i, T) \tag{1}$$

我们的框架 rk 利用大语言模型迭代生成并优化这些节点，以找到最优设计 N^* ：

$$N^* = \arg\max_{N_i \in \mathcal{N}} S_i \tag{2}$$

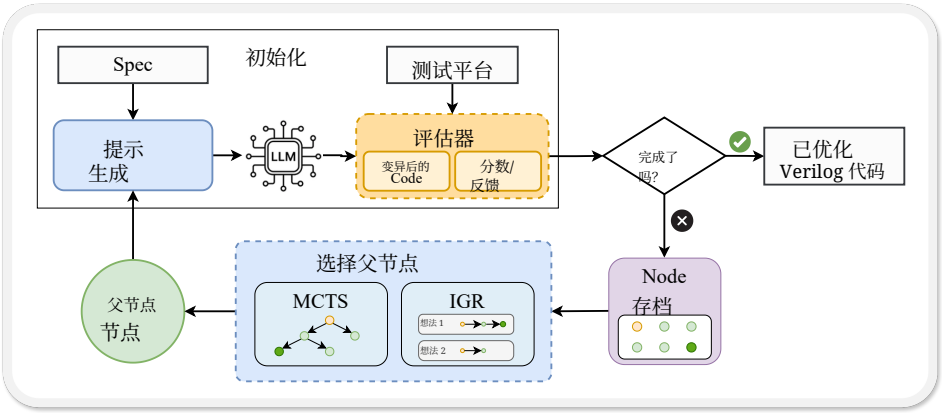

图1：EvolVE搜索框架概览。

评分指标的定义 S_i 取决于具体的任务目标：**生成式综合**或**PPA优化**。

对于**生成式综合**，目标严格限定为功能正确性。设 p_i 表示T 中由 V_i成功通过的测试用例数。分数定义为通过率；若编译或仿真失败，则施加较大的负常数 C_{penalty}：

$$S_{\text{gen}}(V_i) = \begin{cases} \frac{p_i}{|T|} & \text{if simulation success,} \\ C_{\text{penalty}} & \text{otherwise} \end{cases} \tag{3}$$

对于**PPA优化**，目标是在保持功能完整性的同时，最小化功耗、性能和面积指标。我们将其表述为一个约束最大化问题：根据面积 $A(V_i)$ 与延迟 $L(V_i)$ 的乘积，并通过缩放因子 η进行归一化，为有效设计评分：

$$S_{\text{opt}}(V_i) = \begin{cases} -\frac{A(V_i)\cdot L(V_i)}{\eta} & \text{if } p_i = |T|, \\ C_{\text{penalty}} & \text{otherwise} \end{cases} \tag{4}$$

最后，诊断反馈 F_i 会动态调整，以引导大语言模型进行下一轮迭代：

$$F_i = \begin{cases} \texttt{ErrorMsg} & \text{if } S_i = C_{\text{penalty}}, \\ \texttt{Design Summary} & \text{else if Task is Gen,} \\ \texttt{Opt Guidance} \oplus \texttt{Design Summary} & \text{else if Task is Opt} \end{cases} \tag{5}$$

公式4将面积-延迟乘积作为评估函数的快速近似指标，应用于我们的Verilog优化框架。这种方法有两项主要优势。首先，与Synopsys VCS和Design Compiler等商业工作流程相比，使用Yosys和Iverilog等开源工具可将速度提升超过 $30\times$ 。其次，由于面积缩减与功耗优化高度相关，专注于面积可加快仿真，同时仍能取得正向的PPA结果。因此，无需生成FSDB或VCD文件来评估功耗，从而进一步加快流程。如图6所示，我们使用PrimeTime和上述商业工具套件验证了这些改进。

3.2 框架

我们的两种进化策略共享图1所示、附录A中详述的统一框架。从问题描述开始，该框架提示大语言模型生成初始代码。评估器根据测试平台对代码进行评估，以获得量化评分和文本反馈。如第3.1节所定义，节点由代码、分数和反馈组成。在进化阶段，父节点选择策略通过两种方法之一——IGR和蒙特卡洛树搜索（MCTS）——从存档中检索父节点。大语言模型利用父节点的代码和反馈生成优化版本。该周期持续进行，直至设计在生成任务中通过所有测试用例（$S_{\text{gen}} = 1$），或达到最大节点限制。

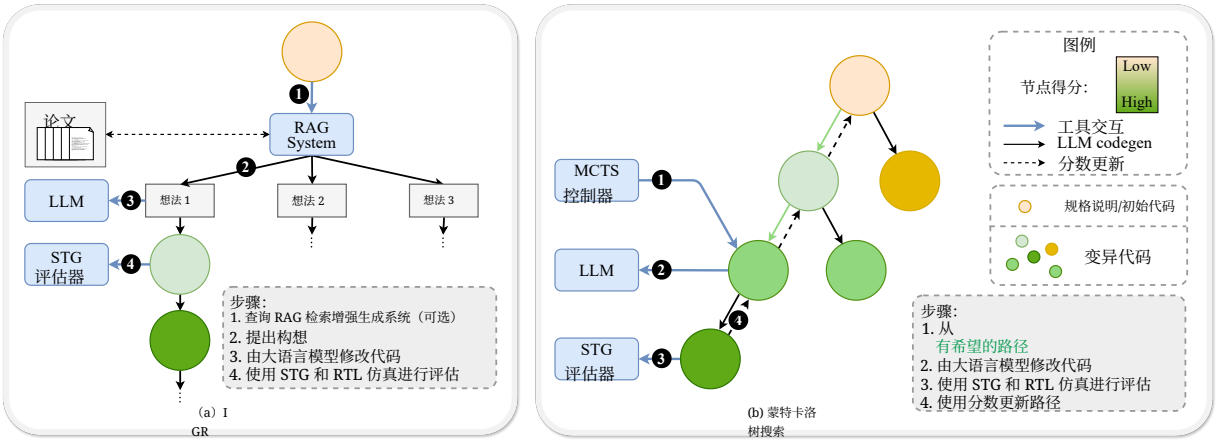

图2：IGR 与蒙特卡洛树搜索（MCTS）的流程对比。

思想引导的改进（IGR）。图2(a)展示了IGR框架的多起点策略。我们将这一过程形式化为两个不同阶段：

1. **想法生成**。大语言模型首先生成一组高层架构概念 $\mathcal{I} = \{I_1, I_2, \ldots, I_k\}$。此过程利用大语言模型的内在知识，也可通过检索增强生成（RAG）从引用论文中获取外部见解来增强这些知识。每个想法 I_k 均以问题描述和先前想法的历史记录 $\{I_1, \ldots, I_{k-1}\}$ 为条件，以确保解空间的多样性。2. **顺序细化**。每个想法 I_k 都会实例化一条独立的细化链 C_k。该链以一个根节点 $N_{k,1}$ 为初始值，该节点由实现 I_k得出。随后，我们依次执行 $m-1$ 个细化步骤。对于每一步 $j \in \{2, \ldots, m\}$，Aider 风格编码器（Gauthier，2025年）会根据反馈 $F_{k,j-1}$，通过对父节点代码$V_{k,j-1}$ 应用差分编辑来生成子节点代码 $V_{k,j}$。这种方法采用有针对性的、基于差异的编辑，而不是重新生成整个文件，从而减少大语言模型令牌消耗。此外，这种单链方法通过在上下文窗口中保留连贯一致的代码状态，有助于顺序调试。相比之下，在蒙特卡洛树搜索中应用此类局部编辑会带来问题，因为管理多个彼此不同的子节点会占满上下文窗口。由此得到一系列节点 $C_k = (N_{k,1}, \ldots, N_{k,m})$。在并行链中总共探索 $N_{\text{total}} = k \times m$ 个节点后，框架会选出使分数 S达到最大的最优节点 N^*。

蒙特卡洛树搜索（MCTS）。我们的框架采用蒙特卡洛树搜索（MCTS）（Coulom，2007）在搜索树 $\mathcal{T}$中组织候选节点，其中根节点 N_{root} 使用大语言模型生成的 Verilog 代码进行初始化。每个节点 $N \in \mathcal{T}$都会维护访问次数 $C(N)$ 和累计质量值 $Q(N)$，后者汇总通过反向传播从其子节点传播而来的分数。搜索通过以下步骤迭代进行：

1. **选择**。从 N_{root}开始，算法递归选择子节点遍历搜索树，直至到达叶节点 N_{leaf}。在每一步中，都会选择基于树的上置信界（UCT）（Kocsis 和 Szepesvári，2006）的一种变体计算出的分数最高的子节点 N_{child}。实际操作中，会优先选择尚未探索的子节点，以最大程度地鼓励探索。选择分数 $UCT(N_{\text{parent}}, N_{\text{child}})$ 定义如下：

$$UCT(N_{\text{parent}}, N_{\text{child}}) = \begin{cases} +\infty, & \text{if } C(N_{\text{child}}) = 0, \\ \frac{Q(N_{\text{child}})}{C(N_{\text{child}})} + c \cdot \frac{\sqrt{\max(1, C(N_{\text{parent}}))}}{1 + C(N_{\text{child}})}, & \text{otherwise} \end{cases} \tag{6}$$

其中，c 用于控制探索项的相对强度。

2. **代码生成与评估**。我们使用所选叶节点 N_{leaf} 的设计摘要来引导代码演化。将此摘要与父节点的代码 V_{leaf} 一并纳入大语言模型提示中，可生成改进导向的新程序 V_{new}。生成后立即使用测试平台评估 V_{new}，以获得质量分数 S_{new}。

3. **扩展与反向传播**。根据生成的代码及其评估分数，创建一个新节点 N_{new}，并将其作为 N_{leaf} 的子节点插入搜索树 $\mathcal{T}$ 中。为该节点分配一个深度值

$\text{depth}(N_{\text{new}}) = \text{depth}(N_{\text{leaf}}) + 1$。最后，分数 S_{new} 沿树向上反向传播至根节点。对于路径上的每个祖先节点 N_a，我们按如下方式更新统计数据：

$$C(N_a) \leftarrow C(N_a) + 1, \quad Q(N_a) \leftarrow Q(N_a) + S_{\text{new}} \tag{7}$$

3.3 评估机制：结构化测试平台生成

现有基准测试（如 VerilogEval）受限于稀疏的输入激励覆盖和二元输出反馈。评估保真度的不足，使得近乎正确的解决方案与根本性失败难以区分。为解决这一问题，我们引入结构化测试平台生成（STG）机制。这是一种确定性验证引擎，分为三个阶段，可提供高保真度、密集的奖励。

阶段 1：自动信号分类。STG 首先使用一组严格的正则表达式解析被测设计（DUT）的端口接口，将信号归入不同的语义类别：时钟/复位、控制信号和数据通路。通过针对标准行业命名规范（例如 clk、rst、valid、ready），STG 无需依赖基于大语言模型的推理，即可确保测试平台实例化过程快速且确定。

阶段 2：可扩展混合激励生成。为兼顾穷举覆盖与仿真可行性，STG 采用受位宽约束的激励策略。对于位宽为 $w \leq 8$ 的控制信号，STG 会生成穷举的 2^w状态空间覆盖，以验证所有可能的模式转换。对于更宽的控制总线（$w > 8$），系统会自动切换为约束随机采样，以避免运行时间爆炸。同时，对于数据通路总线，STG 会采用经过优化的随机采样，并以边界情况（例如零、最大值、交替位）作为种子。生成的测试平台遵循 IEEE 2005 Verilog 标准，因此 STG 不依赖特定仿真器，且完全兼容 Icarus Verilog 等开源工具以及 VCS 等商用仿真器。

阶段 3：细粒度功能梯度。STG 强制 DUT 与黄金参考模型之间严格的时序对齐。在关键状态转换时触发检查，例如复位后以及时钟沿之后短暂延迟时，以隔离时序错误。尤为重要的是，不同于依赖稀疏多臂老虎机反馈的先前工作，STG 会计算连续的细粒度正确性分数 $P_{stg} \in [0, 1]$，其定义为生成的测试向量中的通过率。这相当于功能梯度，使搜索引擎能够优先考虑部分正确的候选方案，从而加速复杂设计空间中的算法收敛。

4 实验

4.1 实验设置

我们使用三个自托管模型评估该框架：Siliconmind-7B（详见附录 C）、GPT-OSS-120B（Agarwal 等，2025）和 DeepSeek-R1-FP4（Guo 等，2025），并同时评估专有模型 Gemini-2.0-Flash。这些模型在三个侧重功能正确性的生成式基准测试上进行评估：VerilogEval v2（Pinckney 等，2024）、Mod-VerilogEval v2 和 RTLLM v2（Liu 等，2024b）。此外，我们还使用提出的 IC-RTL 基准测试评估 PPA优化能力。

我们的评估工具链整合了开源和商业生态系统。我们采用 Icarus Verilog (Iverilog)（Williams 和 Baxter，2002年）以及 Yosys（Wolf 等，2013年）进行快速仿真和综合，同时使用新思科技套件（VCS、Design Compiler 和 PrimeTime）进行严格验证。在此框架下，我们的自托管模型运行于由 8 个 NVIDIA H100 GPU 组成的集群上。具体而言，DeepSeek-R1-FP4 在其混合专家架构（MoE）中采用张量并行和专家并行，而 Siliconmind-7B 则使用纯数据并行。

关于超参数，我们将所有实验的评估预算统一设为 300 个节点。我们将大语言模型温度设置为 0.6，并将惩罚常数 C_{penalty} 设置为 -10^5。对于具体的搜索策略，IGR 设置为生成 $k = 60$ 个初始想法，并经过 $m = 5$步优化；而蒙特卡洛树搜索则采用 3 个子节点的扩展速率，以及值为 $c = 1.4$的探索常数。

4.2 Verilog代码生成

性能比较。本节定量分析 EvolVE 框架在不同大语言模型架构上的效果。通过与既有基线方法进行基准测试，我们表明，我们的推理时搜索策略能够持续提升底层模型的基础能力。

表 1 对原始 VerilogEval v2 和 RTLLM v2 基准测试进行了全面的定量评估。我们的结果表明，推理时搜索显著增强了大语言模型的设计生成能力，

表 1：在原始 VerilogEval v2 和 RTLLM v2 上的性能比较。Pass 列展示了计算节点预算为 **15 / 50 / 100 / 200 / 300** 时通过率（%）的变化。

方法	VerilogEval v2 通过率（%）	RTLLM v2 通过率（%）
基线方法（*Pass@15 / 50 / 100 / 200 / 300*）		
Gemini-2.0-Flash†	72.4 / 74.4 / 77.6 / 78.8 / 80.1	62.0 / 72.0 / 76.0 / 78.0 / 78.0
Siliconmind-7B	82.1 / 87.8 / 89.7 / 91.7 / 91.7	76.0 / 82.0 / 84.0 / 84.0 / 84.0
GPT-OSS-120B	92.9 / 93.6 / 94.2 / 94.2 / 94.2	76.0 / 76.0 / 76.0 / 76.0 / 76.0
DeepSeek-R1-FP4	91.0 / 94.2 / 94.9 / 94.9 / 95.5	82.0 / 84.0 / 84.0 / 86.0 / 86.0
先前工作		
REvolution（Deepseek-V3，使用 200 个节点）†	95.5	88.0
REvolution（Llama-3.3-70B，使用 200 个节点）†	88.5	84.0
VerilogCoder（Llama-3-70B，使用 100 个节点）	67.3	-
VerilogCoder（GPT-4-Turbo，含 100 个节点）†	94.2	-
我们的工作（不含 *STG*）（节点数：*15 / 50 / 100 / 200 / 300*）		
Gemini-2.0-Flash-IGR†	76.3 / 85.3 / 86.5 / 86.5 / 86.5	66.0 / 70.0 / 74.0 / 80.0 / 80.0
Gemini-2.0-Flash-蒙特卡洛树搜索†	77.6 / 81.4 / 84.6 / 86.5 / 87.8	68.0 / 78.0 / 80.0 / 82.0 / 82.0
Siliconmind-7B-MCTS	84.0 / 85.3 / 86.5 / 90.4 / 92.3	76.0 / 80.0 / 82.0 / 84.0 / 88.0
GPT-OSS-120B-IGR	89.1 / 95.5 / 96.2 / 96.8 / 96.8	80.0 / 84.0 / 88.0 / 90.0 / 90.0
GPT-OSS-120B-MCTS	89.7 / 93.6 / 94.9 / 95.5 / 96.2	78.0 / 84.0 / 86.0 / 88.0 / 88.0
DeepSeek-R1-FP4-MCTS	**94.2/96.8/96.8 /97.4/98.1**	82.0 / 84.0 / **86.0** / **90.0** / **92.0**

† 表示通过应用程序编程接口调用访问的模型。

持续优于标准基线和此前最先进的方法。在 DeepSeek-R1-FP4-MCTS 配置下，性能最为突出。即使在仅包含 15 个计算节点的高度受限计算预算下，该模型在 VerilogEval v2 上仍实现了 94.2% 的通过率，基本达到 VerilogCoder（GPT-4-Turbo，使用 100 个节点）报告的最佳性能。随着计算预算扩展至 300 个节点，DeepSeek-R1-FP4-MCTS 在 VerilogEval v2 和 RTLLM v2 上的通过率峰值分别达到 98.1% 和 92.0%。这相较于 REvolution 框架取得了显著提升；后者使用 Deepseek-V3 达到了 95.5%，从而验证了我们的蒙特卡洛树搜索实现对于探索复杂设计空间的优越性。

我们还观察到商业模型应用程序编程接口的性能提升。对于 Gemini-2.0-Flash，标准基线在 VerilogEval v2 上的通过率显著停滞在 80.1%。然而，通过应用我们的进化策略，我们成功释放了潜在的推理能力，使性能在采用 IGR 时提升至 86.5%，采用蒙特卡洛树搜索时提升至 87.8%。尽管这两种方法都带来了显著提升，但与 IGR 相比，蒙特卡洛树搜索在大多数架构上表现出更高的稳定性和更优的峰值性能。

可扩展性与计算预算。 图3展示了我们的蒙特卡洛树搜索方法在不同计算预算（从 1 到 300 个节点）下的性能扩展轨迹。与尝试几次后往往趋于平台期的标准基线方法不同，我们的框架随着节点数量增加，仍能持续提升性能。我们观察到，不同基础模型的推理能力会带来截然不同的扩展行为。对于 DeepSeek-R1-FP4 这类能力很强的模型，搜索的样本效率极高。正如我们收集的数据所示，该配置在第一个节点便解决了 84.6% 的基准测试问题，且平均只需 2.49 个节点即可找到解。相反，对于基线通过率较低的模型，蒙特卡洛树搜索框架能够有效利用增加的预算，解决一部分高难度问题。Siliconmind-7B-MCTS 实验最能体现这一点。虽然该模型很早就解决了大多数较简单的任务，但它在搜索树深入处仍能持续找到解。具体而言，Siliconmind-7B-MCTS 的通过率稳步上升，从 85.3%（15 个节点）提高到 92.9%（300 个节点）。这种单调增长证实，该框架并非仅执行拒绝采样，而是积极利用额外的计算资源来修正难以解决的基准测试中的细微逻辑错误。

结构化测试平台生成的效用。 为量化 STG 模块的具体贡献，我们进行了消融研究，对比了框架在有无语法引导时的表现。如图3所示，在计算预算上限（300 个节点）下，采用 STG 增强的配置与无引导的变体性能相当。然而，STG 的决定性优势体现在图4中，它

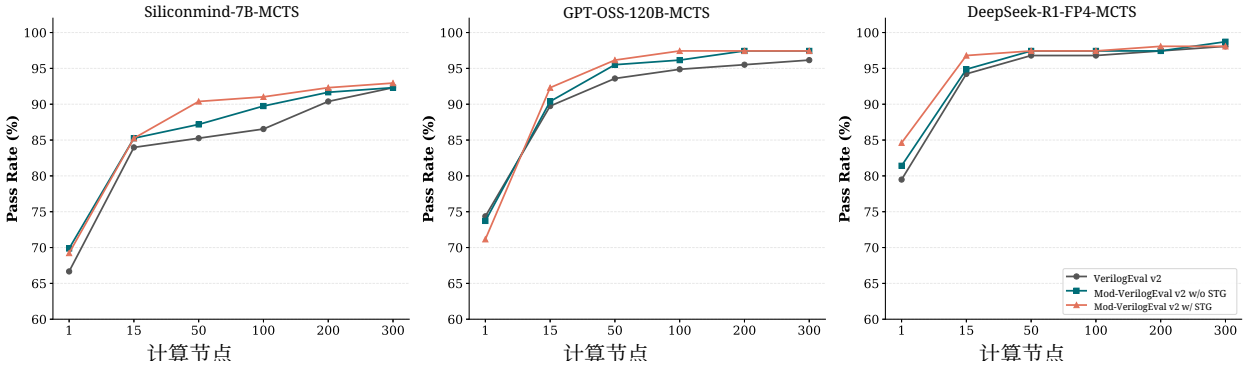

图3：蒙特卡洛树搜索在不同基准测试和模型上的性能扩展。

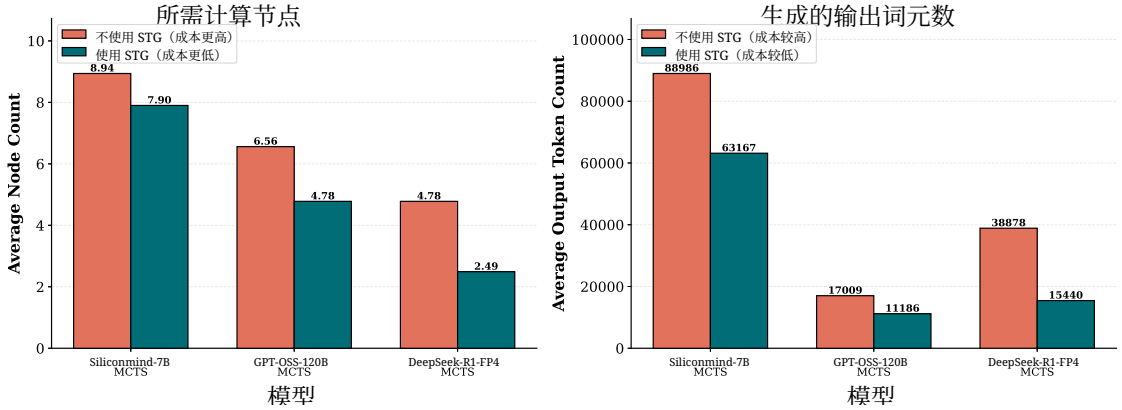

图4：STG在Mod-VerilogEval v2基准测试中带来的收敛效率提升。

凸显了收敛效率的显著提升。通过在搜索过程中严格确保语法有效性，STG 大幅降低了找到这些解所需的计算开销。对于 DeepSeek-R1-FP4 架构，解决 Mod-VerilogEval v2 基准测试所需的平均节点数从 4.78 降至 2.49。此外，总输出 token 消耗减少了 60% 以上（从 38,818 降至 15,440）。这些指标证实，尽管 STG 达到的渐近准确率相当，但它优化了搜索路径，使框架能够以大幅降低的延迟和推理成本达到目标性能水平。

4.3 PPA优化

基准测试：IC-RTL。 为了严格评估该框架处理复杂且具有行业相关性的硬件逻辑的能力，我们引入了IC-RTL。该基准测试套件包含手工设计的问题，以及源自台湾全国集成电路设计竞赛（台湾教育部，2025年）的高级任务。不同于现有侧重于有限规模简单语法的基准测试，IC-RTL要求实现特定的算法结构和内存优化技术，以达到高性能目标。重要的是，这些设计兼容商业平台和开源平台，并为PPA优化提供了广阔的设计空间。具体任务如下：

- **局部二值模式（LBP）：** 该模块使用滑动 3×3 窗口计算 128×128 灰度图像的纹理描述符。其核心挑战在于最大限度地降低存储器访问延迟。优化后的设计通常采用行缓冲，将之前的行存储在本地，从而能够并行处理像素邻域，而无需从主存中重复读取数据。
- **脉动阵列（通用矩阵乘法）：** 此任务涉及使用脉动架构设计高度并行的矩阵乘法引擎。关键优化目标是数据复用和同步。设计必须精确调度数据流，将操作数分块广播到处理单元（PE）阵列中，在最大限度提高吞吐量的同时尽可能降低 I/O 带宽。具体而言，报告的延迟仅涵盖脉动引擎的数据流和执行时间。

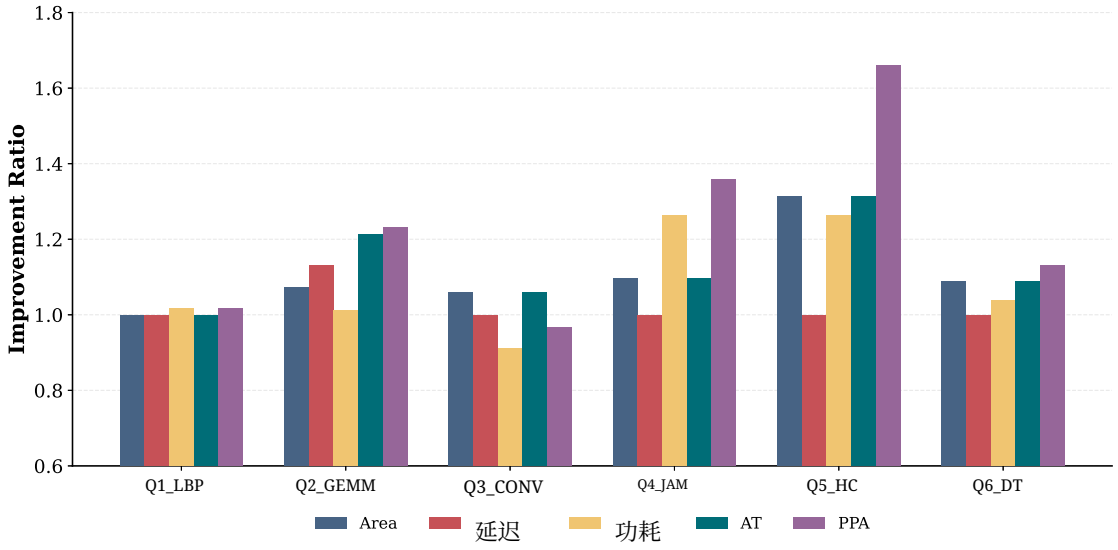

图 5：相较于 IC-RTL 基准测试中的人工基线，功耗、性能和面积方面的改进。

- **图像卷积电路（CONV）：** 此任务实现一个用于卷积神经网络（CNN）层的硬件加速器，包括零填充、 3×3 卷积、偏置加法、ReLU激活和 2×2 最大池化。一个重要约束是定点算术（Q4.16）的处理，以及卷积层与池化层之间中间数据依赖的管理。
- **作业分配机（JAM）：** 该模块用于解决组合优化问题，即将 8 名工人分配给 8 项工作，使分配成本最小。由于该问题需要探索全部 8! (40，320) 种排列，因此硬件必须实现高效的字典序排列生成算法，并采用流水线成本累加路径，以满足时序约束。
- **霍夫曼编码（HC）：** 此任务要求硬件根据符号频率统计生成可变长度霍夫曼编码。其复杂之处在于需要在硬件中实现排序和树构造算法。经过优化的实现通常会采用并行排序网络或优先队列结构，在有限的时钟周期内完成合并与拆分阶段。
- **距离变换（DT）：** 该引擎使用双遍算法（前向遍和后向遍）计算二值图像的棋盘距离。优化挑战在于处理读后写依赖：后向遍依赖前向遍的结果，因此需要高效的存储器仲裁和流水线停顿机制，以防止数据冒险。

优化指标。对于功耗、性能和面积优化任务（公式4），归一化因子设为 $\eta = 10^5$。这种缩放可确保功能有效的设计所得分数始终严格高于惩罚阈值，从而在搜索过程中有效地进行梯度辨别。

功耗、性能和面积性能分析。图 5 展示了 EvolVE 框架通过思想引导的改进（IGR）策略并利用 DeepSeek-R1-FP4 模型，在 IC-RTL 基准测试上的能力。为在真实环境中验证结果，所有生成的 RTL设计均使用 Synopsys Design Compiler 在 TSMC 180纳米工艺节点下进行综合。图表报告了相对于基线方案归一化后的改进比率，其中数值大于 1.0 表示效率更高。

至关重要的是，本评估会扫描离散的目标时钟周期（$T_{clk} \in \{3, \ldots, 7\}\ ns$），并将延迟定义为仿真周期数与时钟周期的乘积。我们优先采用这种方法，而非测量时序裕量，原因有二。首先，商业综合工具由约束驱动：它们会根据特定的时钟目标，从根本上改变电路拓扑（例如，选择不同的标准单元驱动强度或算术架构）。因此，在采用宽松时钟约束综合出的设计上单纯测量时序裕量毫无意义，因为这无法反映工具在更严格约束下会进行的结构优化。其次，PPA效率的全局最优解很少与可达到的最高频率相吻合。通过在离散时钟约束下进行优化，我们能够找出使面积-延迟（AT）乘积最大化的特定工作点，而不是迫使工具在理论极限下满足时序要求，从而避免付出过高的面积和功耗惩罚。

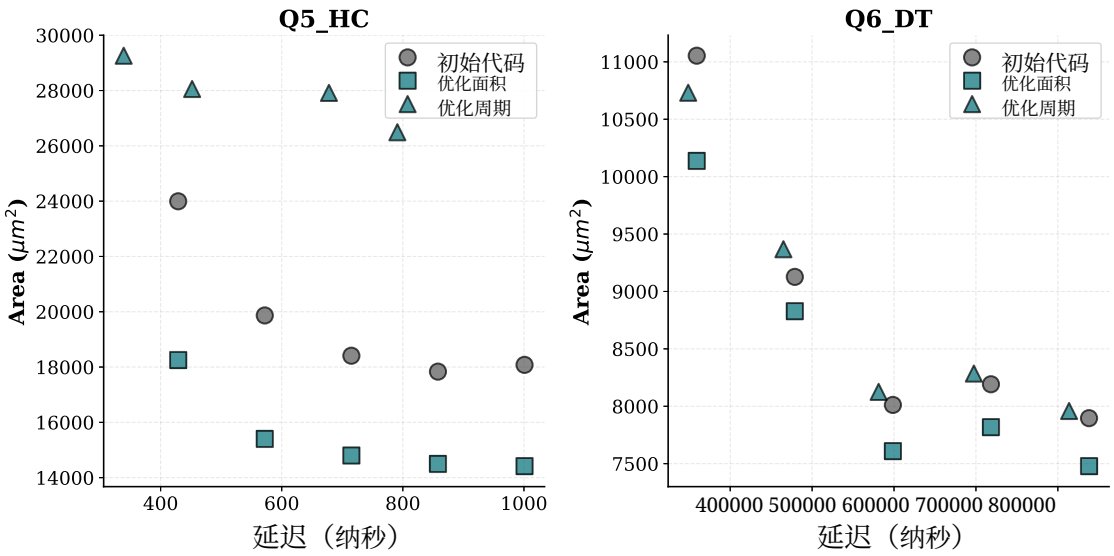

图6：IC-RTL 基准测试上的面积和延迟评估。

图 5 对比了在相同工作频率下优化后的实现与初始实现，完整细节见附录D。结果表明，在多种架构模式下，优化收益始终显著:

- **基线收敛（Q1_局部二值模式）：** 该框架收敛到了一个与人工基线性能特征相匹配的设计。优化后的延迟和面积指标与初始实现相比仍在 1% 以内，这表明人工设计的局部二值模式内核在所探索的设计空间中有效地处于局部最优。
- **脉动阵列效率（Q2_GEMM）：** 矩阵乘法任务展现了该框架在优化规则数据流方面的优势。该设计在面积保持不变、功耗降低的同时，将延迟降低了 12%。这一提升源于优化器有效地重新安排了脉动阵列的控制逻辑，以最大限度地提高数据复用并减少流水线气泡，从而直接改善面积-延迟（AT）乘积，且未增加硬件开销。
- **面积-功耗权衡（Q3_CONV）：** 对于卷积内核，该框架采取了有针对性的权衡：将面积降低 6% 以提高空间效率，同时保持延迟不变。然而，这种紧凑化设计需要更高的开关活动，导致功耗退化（从 0.45 mW 增至 0.50 mW ），抵消了整体 PPA 收益。
- **面积-功耗协同优化（Q4_JAM）：** 作业分配机任务凸显了一种优先考虑面积效率的策略。该框架在不牺牲延迟的情况下，大幅降低了面积（31%）和功耗（26%），使综合 PPA 提升了 36%。这证实了该模型能够识别复杂的优化路径，大幅减少逻辑门数量，同时最大限度地提高能效。这也进一步表明，AT 乘积的初步结果已有效体现在最终的 PPA 优化中。
- **排序网络优化 (Q5_HC)：** 这一显著提升体现了霍夫曼编码任务复杂的设计空间，在该任务中，框架优化了复杂的决策树和排序网络。面积（30%）和功耗（25%）同时降低，带来了峰值达 66% 的功耗、性能和面积提升。这证实了该模型能够高效地重构不规则的控制流逻辑。
- **数据通路扩展 (Q6_DT)：** 距离变换任务在面积和功耗指标上均有所改善，最终实现了稳健的 13% 功耗、性能和面积提升，表明我们的框架能够处理不同类型的复杂问题。

面向设计空间探索 除了一般性的标量改进外，图6还展示了该框架将优化轨迹引导至特定约束条件的能力。针对霍夫曼编码（Q5_HC）和距离变换（Q6_DT）任务，我们通过注入定向提示，引导智能体生成不同的架构变体：一种优先考虑空间效率（优化面积），另一种优先考虑时间速度（优化周期数）。

表 2：通用矩阵乘法候选程序的详细性能比较。

候选者（最佳面积-时间）	Area (μm^2)	时钟 (ns)	周期	时序 (ns)	AT（面积 * 时序）
初始程序	269657	4	362	1448	390463336
Dolphin 的程序	257906	4	320	1280	330119680

在不同的时钟周期（$T_{clk} \in \{3, \ldots, 7\}\ ns$）下进行扫描，图6中的结果呈现出不同的优化行为，从而有效地构成了帕累托前沿：

- **面积导向优化（优化面积）：** 如图6（左）中的方形标记所示，该策略严格遵守基线延迟约束，同时积极降低逻辑资源利用率。对于 Q5_HC，这使面积减少了 31%（在 $3ns$ 时从 2.39×10^4 降至 $1.82 \times 10^4\ \mu m^2$），且性能未受影响。基线（圆形）与面积优化（方形）数据点在垂直方向上对齐，证实该框架成功地单独降低了面积，而未影响时序收敛。
- **延迟导向优化（优化周期）：** 由三角形标记表示的这一变体成功地将延迟与标准时钟缩放解耦。对于 Q5_HC，该框架将执行时间缩短了 26%（在 $3ns$ 时钟下，延迟从 $429ns$ 降至 $339ns$）。至关重要的是，这种加速付出了必要的面积代价（面积从 2.4 $\times 10^4$ 增加到 2.9 $\times 10^4\ \mu m^2$），体现了该框架在需要时权衡经典面积-延迟关系、提供高性能解决方案的能力。

这种差异表明，EvolVE 并非只是收敛到单一局部最优，而是允许设计人员动态权衡 PPA 权衡空间。无论目标是为物联网应用减少逻辑门数量，还是为高性能计算提升吞吐量，该框架都能支持专门化的约束感知实现。

4.4 消融研究

案例研究：通用矩阵乘法中的架构演进。为展示优化的深度，我们分析了 Q2_通用矩阵乘法的结果，该框架同时改善了面积和周期数。人工设计的基线采用输出驻留脉动阵列，通过将矩阵分解为 4×4个矩阵乘法运算来处理可变的矩阵维度。该架构包含专用的输入和权重重塑缓冲区，用于对齐脉动数据流的数据；由于 Icarus Verilog 的限制，还需要采用扁平化的端口接口。

- **微架构改进：** 我们发现，该框架成功优化了数据缓冲方案。对于 $n \times n$ 矩阵乘法，基线设计采用了一个 $(2n-1)$ 行缓冲区，以处理脉动阵列所需的平行四边形数据排布。优化后的设计通过智能多路复用选择数据，将缓冲区压缩至 n 行。此外，该框架还消除了处理单元阵列与输入缓冲区之间冗余的缓冲寄存器，并对关键路径进行了重新定时。在 $4ns$ 时钟约束下，总延迟从 $1448ns$ 降至 $1280ns$，同时面积也从 339,266 减至 315,770 μm^2。

- **新颖架构发现：** 经过扩展的进化搜索，该框架自主地将设计从纯输出驻留模型演进为权重-输出驻留混合架构，使延迟从$1280ns$ 降至 $776ns$。这一转变取消了独立的权重缓冲区，改为直接插入权重，并向操作数矩阵广播输入。这一结果意义重大，因为它验证了该框架能够实现架构级优化，超越局部 RTL 修复，进而发现更优的高层拓扑结构。

5 结论

在这项工作中，我们提出了 EvolVE，这是一种统一且与模型无关的进化框架，使大语言模型能够直接根据自然语言规格生成并优化高质量、功能正确的 Verilog 代码。通过实例化两种不同的搜索策略——思想引导的改进（IGR）和蒙特卡洛树搜索（MCTS）——我们首次研究并证实，MCTS 更适用于功能生成，而 IGR 则最适用于 PPA优化。另一项创新是结构化测试平台生成（STG）引擎，它以细粒度分数取代粗粒度反馈，显著加快收敛，并使该框架能够解决基线模型遇到瓶颈的复杂逻辑问题。此外，我们还推出了 Mod-VerilogEval v2，以解决现有标准中的不一致问题，

并推出 IC-RTL 基准测试套件，弥合学术数据集与工业级复杂度之间的差距，为评估设计优化能力提供了严格的平台。

在 VerilogEval v2、Mod-VerilogEval v2、RTLLM v2 和 IC-RTL 上进行的大量实验表明，该框架达到了最先进的性能，在评估预算有限的情况下，通过率仍高达 98.1%。该方法在各种规模的模型上均效果显著，尤其将 Siliconmind-7B 模型的准确率从 82.1% 提升至 92% 以上。在优化方面，我们的框架优于人工设计的参考方案，在 Q5 上最多可将功耗、性能和面积乘积降低 66%，在整个 IC-RTL 测试套件上的几何平均数降低 17%。至关重要的是，我们发现 EvolVE 能够主动引导优化轨迹，并自主发现架构变换。这些结果表明，在进化指导的增强下，大语言模型能够超越模型规模和数据稀缺带来的局限，进化为能够从规格生成 RTL、进行定向优化以及开展微架构设计空间探索的主动智能体。

尽管取得了这些进展，仍存在一些局限。目前，STG 依赖可执行的参考模型（以 C 或 Verilog 编写的黄金设计）来验证信号正确性，而新型知识产权核可能并不总是具备此类模型。此外，专门针对 PPA优化的基准测试仍然稀缺，而集成全自动综合流程以提供实时 PPA 反馈也有待进一步开发。

未来的工作将聚焦于两个关键方向。首先，受 ACE（Zhang 等人，2025年）等方法的启发，我们旨在整合一个关于微架构优化模式的形式化知识库（优化指南），以指导由 PPA 驱动的探索。其次，我们计划利用蒙特卡洛树搜索的探索特性，更好地优化功耗、性能和面积，而非仅关注功能正确性。我们相信，这些方向将有助于推动 RTL 设计从依赖人工且受专家能力限制的流程，转变为自动化、系统化的学科，从而促使集成电路产业发生根本性变革。

参考文献

Agarwal, S.、Ahmad, L.、Ai, J.、Altman, S.、Applebaum, A.、Arbus, E.、Arora, R. K.、Bai, Y.、Baker, B.、Bao, H. 等（2025）。gpt-oss-120b 与 gpt-oss-20b 模型卡。*arXiv* 预印本 *arXiv:2508.10925*。Coulom, R.（2007）。蒙特卡洛树搜索中的高效选择性与回传算子。载于*Computers and Games*，第72–83页。Deng, C.、Tsai, Y.-D.、Liu, G.-T.、Yu, Z. 和 Ren, H.（2025）。ScaleRTL：利用推理数据和测试时计算扩展大语言模型，实现准确的 RTL 代码生成。*arXiv* 预印本 *arXiv:2506.05566*。Gauthier, P.（2025）。Aider：终端中的人工智能结对编程。Guo, D.、Yang, D.、Zhang, H.、Song, J.、Zhang, R.、Xu, R.、Zhu, Q.、Ma, S.、Wang, P.、Bi, X. 等（2025）。DeepSeek-R1：通过强化学习激励大语言模型的推理能力。*arXiv* 预印本 *arXiv:2501.12948*。Ho, C.-T.、Ren, H. 和 Khailany, B.（2025）。VerilogCoder：具备基于图的规划和基于抽象语法树（AST）的波形跟踪工具的自主 Verilog 编程智能体。Hong, S.、Zhuge, M.、Chen, J.、Zheng, X.、Cheng, Y.、Zhang, C.、Wang, J.、Wang, Z.、Yau, S. K. S.、Lin, Z.、Zhou, L.、Ran, C.、Xiao, L.、Wu, C. 和 Schmidhuber, J.（2024）。MetaGPT：面向多智能体协作框架的元编程。Kocsis, L. 和 Szepesvári, C.（2006）。基于赌博机的蒙特卡洛规划。载于 *Machine Learning: ECML 2006*，第282–293页。Liu, S.、Fang, W.、Lu, Y.、Wang, J.、Zhang, Q.、Zhang, H. 和 Xie, Z.（2024a）。RTLCoder：完全开源且高效的大语言模型辅助 RTL 代码生成技术。*IEEE* 集成电路与系统计算机辅助设计汇刊。Liu, S.、Lu, Y.、Fang, W.、Li, M. 和 Xie, Z.（2024b）。OpenLLM-RTL：用于大语言模型辅助设计 RTL 生成的开放数据集和基准测试。载于 第*43*届 *IEEE/ACM* 国际计算机辅助设计会议论文集，第1–9页。Min, K.、Cho, K.、Jang, J. 和 Kang, S.（2025）。REvolution：由大语言模型驱动的 RTL 生成进化框架。*arXiv* 预印本 *arXiv:2510.21407*。台湾教育部（2025）。全国大学生集成电路设计竞赛。https://proj.moe.edu. tw/moeisoc/cl.aspx?n=6147。访问日期：2026-01-19。Nadimi, B.、Boutaib, G. O. 和 Zheng, H.（2025）。Pyranet：面向 Verilog 的多层次分层数据集。载于 *2025*年第*62*届 *ACM/IEEE* 设计自动化会议（*DAC*），第1–7页。IEEE。Nguyen, M. H.、Chau, T. P.、Nguyen, P. X. 和 Bui, N. D. Q.（2024）。AgileCoder：基于敏捷方法论的软件开发动态协作智能体。

Novikov, A.、V˜u, N.、Eisenberger, M.、Dupont, E.、Huang, P.-S.、Wagner, A. Z.、Shirobokov, S.、Kozlovskii, B.、Ruiz, F. J.、Mehrabian, A. 等（2025）。AlphaEvolve：用于科学与算法发现的编程智能体。*arXiv* 预印本 *arXiv:2506.13131*。Pinckney, N.、Batten, C.、Liu, M.、Ren, H. 和 Khailany, B.（2024）。重新审视 VerilogEval：更新的大语言模型、上下文学习与规范到 RTL 任务。*arXiv* 电子论文，第 arXiv–2408 页。Shao, Z.、Wang, P.、Zhu, Q.、Xu, R.、Song, J.、Bi, X.、Zhang, H.、Zhang, M.、Li, Y. K.、Wu, Y. 和 Guo, D.（2024）。DeepSeekMath：突破开放语言模型的数学推理极限。Tasnia, K.、Garcia, A.、Farheen, T. 和 Rahman, S.（2025）。VeriOpt：通过多角色大语言模型实现面向功耗、性能和面积的高质量 Verilog 生成。*arXiv* 预印本 *arXiv:2507.14776*。Wang, B.、Xiong, Q.、Xiang, Z.、Wang, L. 和 Chen, R.（2025a）。RTLSquad：基于多智能体的可解释 RTL 设计。*arXiv* 预印本 *arXiv:2501.05470*。Wang, Y.、Sun, G.、Ye, W.、Qu, G. 和 Li, A.（2025b）。VeriReason：利用测试平台反馈进行强化学习，以实现推理增强型 Verilog 生成。*arXiv* 预印本 *arXiv:2505.11849*。Wang, Y.、Ye, W.、Guo, P.、He, Y.、Wang, Z.、Tian, B.、He, S.、Sun, G.、Shen, Z.、Chen, S. 等（2025c）。SymRTLO：利用大语言模型和受神经元启发的符号推理增强 RTL 代码优化。*arXiv* 预印本 *arXiv:2504.10369*。Wei, J.、Wang, X.、Schuurmans, D.、Bosma, M.、Xia, F.、Chi, E.、Le, Q. V.、Zhou, D. 等（2022）。思维链提示引发大语言模型中的推理。神经信息处理系统进展，35:24824–24837。Wei, Y.、Huang, Z.、Li, H.、Xing, W. W.、Lin, T.-J. 和 He, L.（2025）。VFlow：发现用于 Verilog 生成的最优智能体工作流。*arXiv* 预印本 *arXiv:2504.03723*。Williams, S. 和 Baxter, M.（2002）。Icarus Verilog：一年多后的开源 Verilog。*Linux Journal*，2002(99):3。Wolf, C.、Glaser, J. 和 Kepler, J.（2013）。Yosys——免费的 Verilog 综合套件。载于 第*21*届奥地利微电子研讨会论文集（*Austrochip*），第97卷。Yang, G.、Zheng, W.、Chen, X.、Liang, D.、Hu, P.、Yang, Y.、Peng, S.、Li, Z.、Feng, J.、Wei, X.、Sun, K.、Ma, D.、Cheng, H.、Shen, Y.、Hu, X.、Zhuo, T. Y. 和 Lo, D.（2025）。用于 Verilog 代码生成的大语言模型：文献综述与未来发展方向。Yao, X.、Wang, Y.、Li, X.、Lian, Y.、Chen, R.、Chen, L.、Yuan, M.、Xu, H. 和 Yu, B.（2024）。RTLRewriter：大模型辅助 RTL 代码优化的方法论。载于 第*43*届 *IEEE/ACM* 国际计算机辅助设计会议论文集，第1–7页。Yuan, J.、Yan, X.、Zhang, B.、Chen, T.、Shi, B.、Ouyang, W.、Qiao, Y.、Bai, L. 和 Zhou, B.（2025）。Dolphin：通过思考、实践和反馈迈向闭环自动研究。载于 计算语言学协会第*63*届年会论文集（第*1*卷：长篇论文），第21768–21789页。Zhang, Q.、Hu, C.、Upasani, S.、Ma, B.、Hong, F.、Kamanuru, V.、Rainton, J.、Wu, C.、Ji, M.、Li, H.、Thakker, U.、Zou, J. 和 Olukotun, K.（2025）。智能体上下文工程：为自我改进语言模型演化上下文。Zhang, Y.、Yu, Z.、Fu, Y.、Wan, C. 和 Lin, Y. C.（2024）。MG-Verilog：面向增强大语言模型辅助 Verilog 生成的多粒度数据集。Zhao, Y.、Zhang, H.、Huang, H.、Yu, Z. 和 Zhao, J.（2025）。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）。Zhu, Y.、Huang, D.、Lyu, H.、Zhang, X.、Li, C.、Shi, W.、Wu, Y.、Mu, J.、Wang, J.、Zhao, Y. 等（2025）。CodeV-R1：推理增强型 Verilog 生成。*arXiv* 预印本 *arXiv:2505.24183*。

详细的进化框架

算法 1 对进化过程进行了形式化描述，将搜索策略（IGR 和蒙特卡洛树搜索）统一到单一优化循环中。根据方法论的定义，搜索空间由一组节点 $\mathcal{N}$ 构成，其中每个节点$N = (V,\ S,\ F)$都包含 Verilog 代码、根据 STG 得出的量化评分以及诊断反馈。

算法 1 EvolVE 通用进化框架

1：输入：问题描述 D，测试平台 T2：输出：优化后的节点 N^* 3：参数：评估函数 $E(V,T)$，大语言模型 M，最大节点数 L_{max}4：5：使用由 D 生成的种子代码初始化节点集合 $\mathcal{N}$。 6：初始化最优解存档 $\mathcal{A} \leftarrow \emptyset$。 7：$n_{count} \leftarrow |\mathcal{N}|$8：9：当（$S_{max} < 1.0$或任务为 Opt）且（$n_{count} < L_{max}$）时，执行循环 10：{步骤 1：通过策略（IGR 或蒙特卡洛树搜索）选择父节点} 11：从 $\mathcal{N}$中选择父节点 $N_{\text{parent}} = (V_p, S_p, F_p)$。 12：13：{步骤 2：生成子节点} 14：使用 D、V_p、分数 S_p 和反馈 F_p构造 $prompt$ 。 15：生成子节点代码 $V_{\text{child}} \leftarrow M(prompt)$。 16：17：{步骤 3：通过 STG 进行评估} 18：计算分数和反馈：$(S_{\text{child}}, F_{\text{child}}) \leftarrow E(V_{\text{child}}, T)$。 19：创建新节点 $N_{\text{child}} = (V_{\text{child}}, S_{\text{child}}, F_{\text{child}})$。 20：21：{步骤 4：更新状态和存档} 22：更新 $\mathcal{N} \leftarrow \mathcal{N} \cup \{N_{\text{child}}\}$。 23：如果 S_{child} 有所改进，则用 N_{child} 更新 $\mathcal{A}$。 24：$n_{count} \leftarrow n_{count} + 1$25：结束循环 26：返回 $\mathcal{A}$ 中分数最高的节点 N^* 。

B 基准测试改进

Mod-VerilogEval v2. VerilogEval v2 是一项广泛采用的基准测试，用于评估 Verilog 生成模型，包含 156 个旨在考察硬件设计能力的问题。对于我们的评估，我们使用规格到 RTL 子集。

不过，我们的初步分析发现，原始数据集中存在一些严重缺陷，包括问题描述不准确、语法无法综合，以及参考模型存在逻辑矛盾。这些问题经常导致正确的设计受到惩罚，妨碍了有效的进化优化。为了解决这些问题，我们与经验丰富的集成电路工程师合作，创建了 Mod-VerilogEval v2。此修订版确保每个问题都有清晰的定义、符合标准 Verilog 语法且在逻辑上可解，从而为进化搜索提供稳定的真实标注。具体改进包括：

- **初始化与复位处理：** 我们从所有参考设计中移除了不可综合的 initial 块。为确保正确进行寄存器初始化，我们引入了显式复位信号，并更新了相应的问题提示（例如，问题 34、53、66、104）。
- **可综合性和逻辑修正：**在若干问题中（例如 116、124、151、156），参考设计会产生不可综合的高阻抗输出或未定义输出。我们重写了这些模块，使其符合标准的可综合 SystemVerilog 规范。
- **规格对齐与消除歧义：**我们解决了提示与实现之间的不一致问题。对于问题 62，我们修正了模块逻辑，使其符合选择要求（当 sel 为 1 时选择 a 或 b）。对于问题 63，我们在提示中明确了移位寄存器的方向，以消除歧义。

- **有限状态机与接口优化：**对于涉及有限状态机（FSM）和复杂接口的问题（例如问题 89、93、123、134），我们完善了描述，以明确状态转换条件和端口声明，并删除了多余信息。

- **命名标准化：**我们修正了命名不一致的问题，以确保测试统一。这包括修正问题 99 中的端口映射（将 Y2/Y4 更正为 Y3/Y1），以及解决问题 130 中的顶层模块命名问题。

C Siliconmind 7b 模型

Siliconmind-7B 是一款轻量级开源学生模型，拥有 70 亿个参数，是一个端到端大语言模型系统的核心。该系统旨在根据自然语言生成在功能上正确且功耗、性能和面积经过优化的 Verilog 模块。开发该系统是为了应对半导体行业面临的挑战，例如高质量 Verilog 数据稀缺，以及需要在有限的本地计算资源上进行训练和部署。其训练始于数据增强：一个规模更大的教师模型 DeepSeek-R1 会筛选并扩充公开数据集，生成高质量样本，其中包括 $< problem,\ reasoning,\ code,\ testbench >$ 组件。随后，Siliconmind-7B 使用这些增强数据，通过监督微调（SFT）和 GRPO 进一步提升，从而学习先进的 Verilog 设计能力。

D IC-RTL 详细结果

本节介绍 IC-RTL 基准测试的详细性能指标。下方表格采用并排排列，以最大限度地提高空间利用率。

表 3：Q1_LBP 结果

Cyc	Area (ns)	Time (μm^2)	Pwr (ns)	PPA (mW)	AT
初始实现					
3	6.49E+03	1.93E+05	0.29	3.68E+08	4.19E+08
4	6.06E+03	2.58E+05	0.29	4.67E+08	3.91E+08
5	6.05E+03	3.22E+05	0.29	5.75E+08	3.90E+08
6	6.05E+03	3.87E+05	0.29	6.90E+08	3.90E+08
7	6.05E+03	4.51E+05	0.29	8.05E+08	3.90E+08
优化版本					
3	6.56E+03	1.93E+05	0.29	3.75E+08	4.23E+08
4	6.06E+03	2.58E+05	0.29	4.59E+08	3.91E+08
5	6.05E+03	3.22E+05	0.29	5.65E+08	3.90E+08
6	6.05E+03	3.87E+05	0.29	6.79E+08	3.90E+08
7	6.05E+03	4.51E+05	0.29	7.92E+08	3.90E+08

表4：Q2_通用矩阵乘法结果

Cyc	Area (ns)	Time (μm^2)	Pwr (ns)	PPA (mW)	AT
初始实现					
3	3.12E+05	1.08E+03	0.66	2.25E+08	1.13E+08
4	3.39E+05	1.45E+03	0.68	3.36E+08	1.23E+08
5	2.90E+05	1.81E+03	0.55	2.91E+08	1.05E+08
6	2.47E+05	2.17E+03	0.50	2.73E+08	8.98E+07
7	2.34E+05	2.54E+03	0.50	3.02E+08	8.52E+07
优化版本					
3	3.14E+05	9.63E+02	0.64	1.96E+08	1.01E+08
4	3.15E+05	1.28E+03	0.67	2.73E+08	1.01E+08
5	2.96E+05	1.60E+03	0.54	2.57E+08	9.50E+07
6	2.49E+05	1.92E+03	0.50	2.44E+08	8.01E+07
7	2.35E+05	2.24E+03	0.50	2.69E+08	7.56E+07

表5：Q3_CONV 结果

Cyc	Area (ns)	Time (μm^2)	Pwr (ns)	PPA (mW)	AT
初始实现					
3	3.64E+04	1.78E+05	0.59	3.83E+08	6.49E+09
4	3.63E+04	2.37E+05	0.57	4.93E+08	8.51E+09
5	3.00E+04	2.97E+05	0.51	4.58E+08	8.93E+09
6	2.35E+04	3.56E+05	0.47	3.93E+08	8.37E+09
7	2.36E+04	4.16E+05	0.46	4.52E+08	9.84E+09
优化版本					
3	3.58E+04	1.78E+05	0.69	3.92E+08	6.37E+09
4	3.53E+04	2.38E+05	0.62	5.15E+08	8.38E+09
5	3.09E+04	2.97E+05	0.55	5.02E+08	9.18E+09
6	2.33E+04	3.56E+05	0.50	4.18E+08	8.30E+09
7	2.23E+04	4.16E+05	0.50	4.67E+08	9.29E+09

表6：Q4_JAM 结果

Cyc	Area (ns)	Time (μm^2)	Pwr (ns)	PPA (mW)	AT
初始实现					
3	1.31E+04	1.21E+06	0.82	1.30E+10	5.27E+09
4	1.09E+04	1.61E+06	0.77	1.35E+10	4.38E+09
5	9.14E+03	2.02E+06	0.53	9.83E+09	3.69E+09
6	8.98E+03	2.42E+06	0.44	9.59E+09	3.62E+09
7	8.97E+03	2.82E+06	0.37	9.34E+09	3.62E+09
优化版本					
3	1.30E+04	1.21E+06	1.05	1.66E+10	5.26E+09
4	1.01E+04	1.61E+06	0.61	9.91E+09	4.07E+09
5	8.77E+03	2.02E+06	0.41	7.32E+09	3.54E+09
6	8.74E+03	2.42E+06	0.34	7.22E+09	3.53E+09
7	8.73E+03	2.82E+06	0.29	7.18E+09	3.52E+09

表 7：Q5_HC 结果

Cyc (ns)	Area (μm^2)	Time (ns)	Pwr (mW)	PPA	AT
初始值					
3	2.39E+04	4.29E+02	2.49	2.56E+07	3.43E+06
4	1.98E+04	5.72E+02	1.41	1.60E+07	2.84E+06
5	1.84E+04	7.15E+02	1.05	1.38E+07	2.63E+06
6	1.78E+04	8.58E+02	0.85	1.30E+07	2.55E+06
7	1.80E+04	1.00E+03	0.73	1.32E+07	2.58E+06
面积优化					
3	1.82E+04	4.29E+02	1.97	1.54E+07	2.61E+06
4	1.54E+04	5.72E+02	1.17	1.03E+07	2.20E+06
5	1.48E+04	7.15E+02	0.90	9.56E+06	2.11E+06
6	1.45E+04	8.58E+02	0.71	8.94E+06	2.07E+06
7	1.44E+04	1.00E+03	0.61	8.92E+06	2.06E+06
周期优化					
3	2.92E+04	3.39E+02	2.25	2.23E+07	3.30E+06
4	2.80E+04	4.52E+02	2.42	3.07E+07	3.17E+06
5	2.74E+04	—	—	—	—
6	2.79E+04	6.78E+02	1.19	2.26E+07	3.15E+06
7	2.64E+04	7.91E+02	1.61	3.37E+07	2.99E+06

表 8：Q6_DT 结果

Cyc (ns)	Area (μm^2)	Time (ns)	Pwr (mW)	PPA	AT
初始实现					
3	1.10E+04	3.59E+05	0.53	2.14E+09	1.32E+09
4	9.12E+03	4.78E+05	0.37	1.63E+09	1.09E+09
5	8.01E+03	5.98E+05	0.34	1.64E+09	9.59E+09
6	8.19E+03	7.18E+05	0.36	2.13E+09	9.80E+08
7	7.89E+03	8.38E+05	0.34	2.26E+09	9.45E+08
面积优化					
3	1.01E+04	3.59E+05	0.52	1.89E+09	1.21E+09
4	8.82E+03	4.78E+05	0.40	1.70E+09	1.05E+09
5	7.60E+03	5.98E+05	0.34	1.57E+09	9.11E+08
6	7.81E+03	7.18E+05	0.36	2.06E+09	9.36E+08
7	7.47E+03	8.38E+05	0.34	2.18E+09	8.95E+08
周期优化					
3	1.07E+04	3.48E+05	0.52	1.94E+09	1.24E+09
4	9.36E+03	4.65E+05	0.38	1.67E+09	1.08E+09
5	8.12E+03	5.81E+05	0.35	1.66E+09	9.44E+08
6	8.28E+03	6.97E+05	0.36	2.09E+09	9.63E+08
7	7.95E+03	8.13E+05	0.34	2.22E+09	9.25E+08