arXiv:2601.01765v1 [cs.AI] 5 Jan 2026

用于恰当评估寄存器传输级代码优化的新基准测试

Yao Lu、Shang Liu、Hangan Zhou、Wenji Fang、Qijun Zhang、Zhiyao Xie*
香港科技大学（*通讯作者：eezhiyao@ust.hk）

摘要

人工智能的快速发展日益依赖高效的集成电路（IC）设计。近期研究探索了利用大语言模型（LLM）生成寄存器传输级（RTL）代码，但现有基准测试主要评估语法正确性，而非从功耗、性能和面积（PPA）角度衡量优化质量。本文推出RTL-OPT，这是一个用于评估大语言模型 RTL 优化能力的基准测试。RTL-OPT 包含 36 个手工设计的数字电路设计，涵盖多种实现类别，包括组合逻辑、流水线数据通路、有限状态机和存储器接口。每项任务均提供一对 RTL 代码：一个次优版本和一个由人工优化的参考设计，后者体现了传统综合工具无法捕捉、且经业界验证的优化模式。此外，RTL-OPT 集成了自动化评估框架，用于验证功能正确性并量化 PPA 改进，从而能够对面向硬件设计优化的生成模型进行标准化且有意义的评估[1]。

1 引言

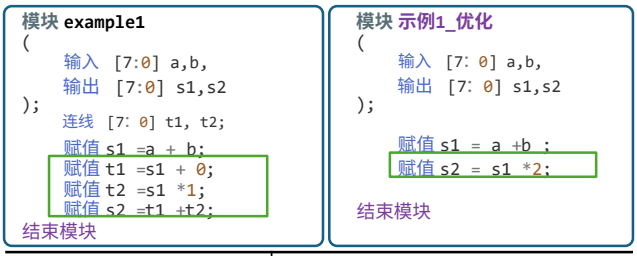
近年来，大语言模型（LLMs）在集成电路敏捷设计中的应用已成为一个颇具前景的研究方向 [8]。尤其是，近期许多工作 [5, 6, 9–11, 13, 14, 16, 18, 22, 27]致力于开发定制化大语言模型，以直接生成寄存器传输级（RTL）代码格式的集成电路设计，例如 Verilog。

RTL 代码生成基准测试。 RTL 设计是数字集成电路设计实现的起点，需要投入大量人力和专业知识。大语言模型辅助的 RTL 代码生成技术 [10, 14, 16, 27] 旨在将工程师从繁琐的 RTL 编码过程中解放出来。为了公平比较不同大语言模型在 RTL 生成方面的能力，高质量的基准测试变得必不可少。具有代表性的 RTL 代码生成基准测试包括 VerilogEval [12] 和 RTLLM [17], VerilogEval v2 [19], RTLLM 2.0 [15], CVDP [20], 以及其他基准测试 [4, 7]。

评估 RTL 优化的难点。 然而，上述基准测试主要关注 RTL 代码生成的正确性，并未明确评估

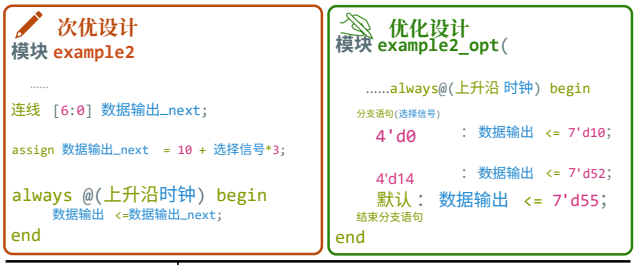
集成电路设计在功耗、性能和面积方面的极致优化（PPA）。RTL代码将使用综合工具进行综合，生成最终的电路实现；在将RTL代码转换为实现的过程中，综合工具会执行广泛的逻辑优化。因此，PPA结果取决于RTL代码质量和下游综合流程。正如我们将在本文中指出的，这种紧密的相互作用使 RTL 优化的基准测试尤其具有挑战性，有时甚至会产生误导。

[1]RTL-OPT 可在 https://anonymous.4open.science/r/RTL-OPT-20C5 获取。其中还包含 RTL-OPT 的详细实验结果和全面评估。



DC 结果	单元	Area	功耗	WNS	TNS
example1 & 示例1_优化	106	94.7	0.51	-0.18	-1.81

（a）现有基准测试 [26]中一对过度刻意构造的次优和已优化RTL代码示例。



DC 结果	单元	Area	功耗	WNS	TNS
example1	38	56.39	0.44	-0.11	-0.38
example1_opt	31	49.74	0.39	-0.08	-0.24

(b) 我们的基准测试 RTL-OPT 中一个次优与已优化寄存器传输级代码的示例，体现了现实中的优化机会。

图1：少数现有寄存器传输级优化工作中的示例与我们提出的基准测试的比较，该基准测试捕捉了现实中的寄存器传输级优化机会。次优设计是供大语言模型改进的输入，而已优化设计是黄金参考。

RTL代码优化基准测试。 由于开源电路设计极其稀缺，而这些设计对于半导体公司而言是宝贵的知识产权，因此构建高质量的 RTL 优化基准测试本身就极具挑战性。最近，一些大语言模型研究 [23, 25, 26] 开始致力于生成更加已优化的RTL代码，期望以此在功耗、性能和面积方面实现更优的最终芯片质量。这些研究均在唯一相关的基准测试 [26], 上进行评估，该基准测试为大语言模型提供次优 RTL 代码以供改进。然而，我们的研究表明，该基准测试 [26]在以下几个方面存在不足：1）**设计不切实际：** 该基准测试中的许多次优 RTL 代码过于刻意，无法反映实际应用中的真实低效问题；2）**综合设置过于简化：** 依赖 Yosys 等较弱的综合工具 [24]，导致结果容易受到表面 RTL 代码修改的影响，且与工业级流程的契合度较低

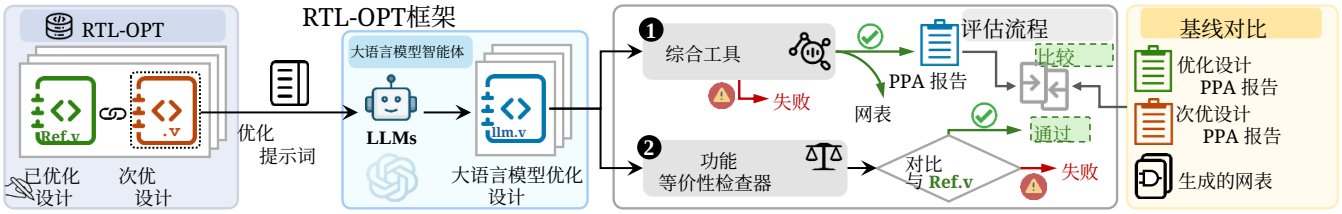


图2：用于自动化基准测试 RTL 优化的 RTL-OPT 工作流程。

；3）**评估不充分：**其评估仅关注与面积相关的指标，却忽略了功耗和时序。此类评估指标忽视了典型集成电路设计流程中普遍存在的权衡关系。

在本研究中，我们首先考察了现有的 RTL 优化工作，并重新思考一个关键问题：**如何恰当地对 RTL 代码的优化进行基准测试？**[2] 我们仔细考察了现有工作和下游综合流程。这项研究表明，对 RTL 优化进行评估并非易事，而且很容易得出误导性结论。具体而言，一段 RTL 代码是否优于另一段（即优化程度更高），很大程度上取决于综合工具和设置。以往工作所指出的许多“已优化”RTL 代码 [26] 在采用不同的、通常更先进的综合选项后，结果与其“次优”RTL 代码对应版本相同，甚至更差。

图1a突出了 [26] 设计中的一个常见缺陷：其 RTL 配对基于不切实际的转换，未能解决硬件设计中的真正优化难题。这些人为构造的次优示例存在不必要的低效问题，例如冗余计算和多余的算术运算，而这些情况在实践中几乎不会出现。综合工具可以轻松优化这些人为构造的模式，从而导致对大语言模型改善 RTL 质量效果的评估过高。优化版本（example1_opt）通过直接计算 s1 = a + b 来实现逻辑，然后将输出推导为 s2 = s1 * 2。相比之下，次优版本（example1）引入了人为且不必要的步骤：它首先计算 s1 = a + b，然后冗余地加上 0 并乘以 1，以生成中间连线 t1 和 t2，最后将它们相加得到 s2。这类构造并不自然，在实际 RTL 编码中几乎不会出现，因此使基准测试示例缺乏现实性。图1b展示了一个来自 RTL-OPT 的示例，详细分析见第 3.2 节。

基于上述观察，我们提出了一个全新的基准测试 **RTL-OPT**，专门用于系统评估大语言模型在 RTL 设计优化方面的能力。RTL-OPT 包含 36 项手工设计的 RTL 优化任务，旨在针对 PPA 指标进行优化。其一项关键特色是提供了**多样且贴近实际的优化模式**，例如位宽优化、预计算与查找表转换、运算符强度削减、控制简化、资源共享以及状态编码优化，这些模式均源自经过验证的行业实践。它们涵盖了对 RTL 优化真正重要的转换，即使在采用先进综合技术时仍然有效。这使 RTL-OPT 有别于以往通常缺乏实际优化效果的工作。如图 2 所示，每项 RTL-OPT 任务都提供一对 RTL 代码：一个刻意设计的次优（待优化）版本，以及一个作为黄金参考的优化版本。大语言模型以次优代码为输入，尝试在保留设计功能的同时生成更优化的 RTL 代码。具体而言，RTL-OPT 提供：❶ 36 项手工设计的任务，确保全面且具有代表性地覆盖真实世界设计挑战；❷ 一个集成的评估框架（图 2），可自动验证功能正确性，并将大语言模型优化设计最终的功耗、性能和面积与设计人员优化的黄金参考进行比较。

2 重新思考 RTL 代码优化

本节将全面研究现有的 RTL 代码优化基准测试，以及我们提出的 RTL-OPT 基准在多种综合配置下的表现。这项研究揭示了仅依赖传统基准测试中人为设计的示例所固有的局限性。

2.1 综合流程对 RTL 评估的影响

根据我们的研究，我们指出，RTL 优化的评估（即判断哪种 RTL 代码能够带来更好的功耗、性能和面积）并非易事。**一个主要原因是，最终的设计质量还取决于综合流程**。综合工具、优化模式和时序约束 的差异，都会显著影响 RTL 代码中的结构差异是否以及如何体现在最终实现中。

综合工具的影响。 不同工具采用不同的优化策略和启发式方法。Synopsys Design Compiler (DC) [1]等商业工具通常会进行更激进、更复杂的优化，而 Yosys [24]等开源工具则提供了性能较弱、但在学术研究中颇具价值的替代方案。因此，同一对 RTL 代码的质量差异程度可能因所用综合工具而异。

编译模式的影响。 商业工具支持多种编译模式，优化力度各不相同。更先进的模式（例如，Design Compiler 中的 compile_ultra）往往会大幅重构逻辑，从而掩盖寄存器传输级代码之间的细微差异。

时钟周期约束的影响。目标时钟周期也会影响综合行为。严格的约束通常会促使工具积极进行时序驱动优化，而宽松的约束则可能缩小不同 RTL 变体之间的差异。选择现实且一致的时序目标，对于公平、可解释地评估 RTL代码至关重要。

[2]请注意，开创性工作 [26] 并未声称其设计是标准基准测试，我们也感谢他们所做的开源贡献。

基准测试	总计	Yosys			Design Compiler（编译，时钟 = 1纳秒 | 0.1 纳秒）				Design Compiler（编译_ultra，时钟 = 1纳秒 | 0.1 纳秒）			
		相同	更差	更好	相同	权衡	更差	更好	权衡	相同	更差	更好
[26]的基准测试	43	13	6	24	13 | 13	7 | 9	7 | 6	16 | 15	21 | 22	1 | 3	8 | 7	13 | 11
[26]的论文	12	1	0	11	1 | 1	4 | 4	2 | 1	5 | 6	4 | 4	1 | 1	3 | 1	4 | 6
SymRTLO [23]	13	2	1	10	2 | 2	2 | 2	1 | 2	8 | 7	4 | 4	1 | 0	3 | 3	5 | 6
RTL-OPT	36	3	0	33	0 | 0	6 | 13	0 | 0	30 | 23	0 | 0	1 | 12	0 | 0	35 | 24

表1：[26] 与 RTL-OPT（本文工作）中每对次优和人工优化设计的比较。RTL-OPT 始终表现出改进（36 个设计中有 35 个在 compile_ultra 时钟 = 1纳秒条件下更优），符合专家优化后的寄存器传输级设计应优于次优版本的预期。相比之下，先前的基准测试在更强的综合设置下通常表现出很少或没有改进，这表明其用于寄存器传输级优化基准测试的可靠性有限。

大语言模型解决方案	总计	Yosys			Design Compiler（compile，时钟 = 1纳秒）				Design Compiler（编译_ultra，时钟 = 1纳秒）			
		相同或更差	更好	相同	的权衡		更差	更好	相同的权衡		更差	更好
GPT-4.0	13	4	3	6	3	3	2	5	4	4	2	3
模型 [26]	12	3	1	8	3	4	0	5	3	6	0	3

表2：来自 [26] 的次优 设计与大语言模型优化 设计的比较。与表1类似，我们评估在不同综合工具下，大语言模型优化后的参考设计与其次优 RTL 相比是更好、更差，还是相同。

2.2 现有基准测试的检查

现有基准测试 [26]提供了多组次优与人工优化的RTL设计配对，以及基于大语言模型的优化实验生成的其他 RTL代码。令人惊讶的是，我们的研究发现：1） [26] 中的人工优化的RTL设计和由大语言模型优化的RTL设计，在综合后往往无法胜过相应的次优设计。许多情况下，它们的表现基本相同，甚至更差，尤其是在采用高级综合选项时。2）我们观察到，不同综合工具对最终功耗、性能和面积的影响明显不同：优化能力强大的商用工具 Design Compiler 往往会消除次优 RTL 与优化后 RTL 之间的差异，而开源工具 Yosys 则常常会夸大这些差异。综合来看，这些结果表明，现有基准测试无法可靠地反映 RTL代码的真实改进。

现有基准测试的评估结果见表1和表2。我们仔细检查并评估了整个基准测试中发布的全部 43 对 RTL代码[3] [26]。具体而言，表1比较了原始基准测试中的每一对次优与人工优化设计。我们评估人工优化参考设计经过综合[4]后，相较于对应的次优 RTL设计究竟更好、更差还是相同。此外，功耗、性能和面积比较中也可能出现“权衡”结果，即一个功耗、性能和面积指标有所改进，而另一个则有所退化。对于 Yosys，与以往研究类似，我们只比较单元数量。表1显示，在 43 个分支语句中，使用 compile_ultra 时，只有 13 个人工优化RTL [26]优于对应的次优设计。使用 compile 时，这一数量增至 16；使用 Yosys 时则进一步增至 24。许多人工优化RTL并不优于次优 RTL，尤其是在采用高级综合选项时。这验证了商业工具能够消除许多刻意设计的低效之处，而开源工具往往会保留这些问题，凸显了两者之间的明显差异。

我们还通过设置更严格的时序目标 clockperiod，扩展了在不同 = 0时钟约束下的评估，目标为 .1纳秒。相应结果以格雷码条目显示在表1中。采用相同的综合模式（即compile 或 compile_ultra）时，收紧时序约束会使出现功耗、性能和面积权衡的分支略有增加，而实际更优的 RTL代码则进一步减少。

表2进一步将大语言模型优化 设计与先前工作 [26]直接发布的次优设计进行了比较。在表2中，对于 GPT-4.0以及 [26], 提出的模型，在使用 compile_ultra 时，12个分支语句中实际上只有3个大语言模型优化 RTL优于其次优 对应设计。使用 compile 时，这一数字上升到12个中的5个。总之，许多大语言模型优化 RTL最终并不优于次优 RTL，尤其是在采用高级综合选项时。这表明，在严格的综合流程下，该基准测试的有效性有限。

2.3 相同设置下对我们基准的检查

在表1中，我们还采用相同的设置评估了所提出的基准 RTL-OPT。根据表1，在采用 compile_ultra 时，RTL-OPT 中 36 个人工优化的RTL 代码有 35 个表现更好，而对于 Yosys 则有 36 个中的 33 个表现更好。在表 1 中，在严格的时序约束下，仍有 23 个分支语句表现更好，而 13 个产生了 PPA 权衡，没有任何分支语句达到相同的最终 PPA。与先前的基准相比， [26], RTL-OPT 在评估 RTL设计方面取得了显著改进。

RTL-OPT 基准质量得到了清晰验证，这源于其设计理念：它提供了确实次优的RTL实现，并留有切实的改进空间，而不是提供综合工具可以消除的刻意低效。

3 RTL-OPT 基准测试

本节介绍 RTL-OPT，这是一个旨在评估大语言模型进行 RTL 代码优化能力的基准测试。RTL-OPT 提供由专家手工制作的真实次优与已优化 RTL 配对，

[3]如表1所示， [26] 的论文与 SymRTLO [23] 是基准测试 [26]的不同子集。在基准测试 [26]的54对 RTL代码中，我们仅成功综合了43个分支语句。对于其余部分，我们综合了14对中的12对，并分别为 16 对中的 13 对。这些原始基准中的综合失败主要由 Verilog 语法错误导致。[4]用于评估的综合流程、工具和 PPA 指标的详细信息见第 3.3 节。

确保其中存在真实的低效之处，并提供有意义的黄金参考。该基准测试涵盖多种设计类型，并使用商业工具和开源工具进行评估，为推进 RTL 优化研究提供了可靠且实用的资源。

3.1 基准测试描述

RTL-OPT 由 36 个 RTL 设计优化任务组成。每个任务提供一对 RTL 代码：一个次优版本和一个对应的设计者优化版本，实现相同的功能。所有设计均由硬件工程师手工编写，以体现真实的编码风格和优化实践，其中优化版本作为黄金参考，用于衡量人工优化的 PPA 质量。次优 RTL 并非任意降级的产物；它代表一种有效且功能正确的设计，只是省略了特定的优化机会。这种设置形成了有意义的优化差距，并反映了半导体行业中常见的实际场景。

所提供的 36 个设计任务涵盖多种设计类型，包括算术单元、控制逻辑、有限状态机（FSMs）和流水线数据通路。这些设计在规模和复杂度上各不相同，逻辑面积范围从 **14** 到 **20K** 个单元，综合面积范围从 **15** 到 **19K** μm^2。这种多样性确保该基准测试能够代表实际的 RTL 设计任务。

表 3 汇总了 RTL-OPT 在 36 个基准测试中提供的次优设计和优化设计的平均值 PPA 指标。这些结果包括使用商用 Design Compiler 和开源 Yosys 工具进行的综合与评估。从表中可以明显看出，与优化设计相比，次优设计的 PPA 值显著更高。这些结果表明，即使使用 Design Compiler，次优设计仍有很大的改进空间，凸显了 RTL-OPT 作为大语言模型 RTL 代码 PPA 优化基准测试的有效性。

RTL-OPT 完全开源，并提供以下资源，用于评测大语言模型的 RTL 优化能力：（1）36 对经过精心设计的 RTL 代码；（2）由商用综合工具生成的相应综合网表；（3）电子设计自动化（EDA）工具针对次优设计和优化设计生成的详细 PPA 报告；（4）完整的工具链流程，包括标准单元库，以及用于综合、仿真和功能验证的脚本，还可用于验证大语言模型重写代码的正确性。

3.2 RTL-OPT分析：优化模式

这些优化模式提供了优化机会，源自经过验证的行业标准 RTL 编码实践，并直接影响逻辑综合的质量。这些模式体现了特定寄存器传输级修改最终如何影响下游综合结果。
RTL-OPT 基准中的主要优化模式类型概述如下：

- **位宽优化：** 在不需要完整精度时缩减寄存器和连线的位宽，从而优化面积和功耗。
- **预计算与查找表转换：** 用预计算的查找表替代运行时算术运算，以消除复杂的逻辑单元。

RTL-OPT 次优设计								
指标	DC 结果					Yosys 结果		
	单元	*Area*	功耗 *(mW)*	*WNS (ns)*	*TNS (ns)*	布线	单元	*Area*
平均值	1226.8	1337.4	14.0	0.0	0.0	636.9	360.4	559.7
RTL-OPT 优化设计								
指标	DC 结果					Yosys 结果		
	单元	*Area*	功耗 *(mW)*	*WNS (ns)*	*TNS (ns)*	布线	单元	*Area*
平均值	901.8	1047.7	9.1	0.0	0.0	597.4	313.7	503.1
改进 (%)	**36.04**	**27.64**	**53.37**	-	-	**6.62**	**14.88**	**11.25**

表 3：RTL-OPT 提供的次优设计与已优化设计的平均功耗、性能和面积对比。我们使用商业工具 DC 和开源工具 Yosys 进行 RTL 综合和 PPA评估。由于篇幅有限，我们仅展示所有设计的平均值，详细结果见 GitHub。

- **运算符强度削减：** 通过位操作，将高成本运算符替换为更简单的等效运算符。
- **控制简化：** 通过扁平化嵌套有限状态机（FSM），或减少不必要的状态，精简控制逻辑，并改善面积和时序。
- **资源共享：** 合并不同周期中的重复逻辑，以最大限度提高硬件资源效率。
- **状态编码优化：** 根据状态数量选择最优状态编码方案（独热编码、格雷码、二进制编码），以平衡功耗、面积和时序。

通过整合多种 RTL设计中的优化模式，RTL-OPT为大语言模型辅助的RTL代码优化构建了一个贴近实际且具有挑战性的基准：在保持功能正确性的同时，提升已优化代码的PPA 指标。

为说明这些优化模式，我们在图1b中给出了 RTL-OPT 的一个简单代码示例。该示例对比了特定模式类别下的次优和已优化 RTL实现，并讨论了结构变化以及下游 PPA 指标的定量改进。

示例： 图1b中的示例展示了**预计算与查找表转换**这一优化模式，即用预计算值替代实时算术运算。在次优设计中，输出根据 sel 输入动态计算，因此每个时钟周期都需要执行乘法。已优化设计则用一个case语句替代该运算，根据选择信号直接赋予预计算值。通过消除算术运算并减少信号翻转，该优化使面积减少14%，功耗降低12%。

3.3 评估方法与工具

RTL-OPT 提供了完整的评估流程，通过测量综合后的 RTL 代码的功耗、性能和面积，评估大语言模型的优化能力。这一流程结合了综合、功能验证和 PPA 评估，所有环节均使用符合行业标准的电子设计自动化工具完成。

3.3.1 综合流程。 逻辑综合流程会将初始 RTL 代码转换为门级网表，并以此为基础对功耗、性能和面积指标进行定量评估。在本研究中，我们主要使用 DC [1] 对 RTL-OPT 基准进行综合，因为它在工业设计流程中已被证明卓有成效。DC 展现出卓越的能力，能够识别低效的 RTL 结构并将其优化为更高效的电路实现，从而降低对初始代码质量的敏感度。

当基准测试质量不足时，DC 往往会将次优和已优化的 RTL 代码综合为功能等价的门级网表，从而产生相同的功耗、性能和面积结果。这种现象反映出低质量基准测试所提供的优化机会有限。相反，像 Yosys 这样的开源综合工具 [24],优化力度较弱，对于此类代码对仍可能产生不同的功耗、性能和面积结果，从而可能夸大某些代码转换的有效性。为完整起见，我们还提供了使用 Yosys 获得的综合结果，以支持更广泛的比较分析。

综合流程还涉及使用工艺库（也称为单元库），它是由预先表征的标准单元构成的集合，例如逻辑门、触发器和其他基本组件。这些单元经过设计，可满足特定的功耗、性能和面积约束。虽然我们在评估中使用 Nangate45 [21]，但也可以使用其他库，不过这些库通常需要许可证。库的选择会显著影响 RTL 优化流程，因为它决定了可用单元及其性能特征，最终影响设计的效率。

3.3.2 功能等价性验证。 成功综合后，RTL代码可正确转换为门级网表，因此可以确保不存在语法错误。综合之后，功能验证对于确保优化步骤未引入错误至关重要。此验证主要使用 **SynopsysFormality** [3], 进行，通过严格比较经大语言模型优化的 RTL 与黄金参考，执行功能等价性检查，以确保行为一致。然而，对于涉及时序调整的优化（例如流水线化），还需要进行额外的动态验证。此验证使用 **Synopsys VCS** [2], 执行，它通过全面的测试平台，在各种运行条件下验证设计行为。

这种结合方法确保优化后的设计既具有逻辑等价性，又具备运行可靠性。只有通过形式等价性检查，并针对时序关键型优化通过动态验证的设计，才被视为在功能上有效。

3.3.3 优化中的 *PPA* 指标与权衡。 为了评估综合后设计的质量，我们从三个方面进行分析：**功耗**、**性能**和**面积**:

功耗：综合工具报告的总功耗，包括动态功耗和漏电功耗。

性能：通过最差负裕量（*WNS*）和总负裕量（*TNS*）衡量时序质量。

面积：通过硅面积和单元数量表征物理实现成本。

这些功耗、性能和面积指标之间普遍存在权衡。例如，优化功耗可能会增加面积，而缩小面积则可能损害功耗效率。寄存器传输级优化的一项关键挑战是根据设计约束协调这些相互竞争的目标，以实现最佳平衡。

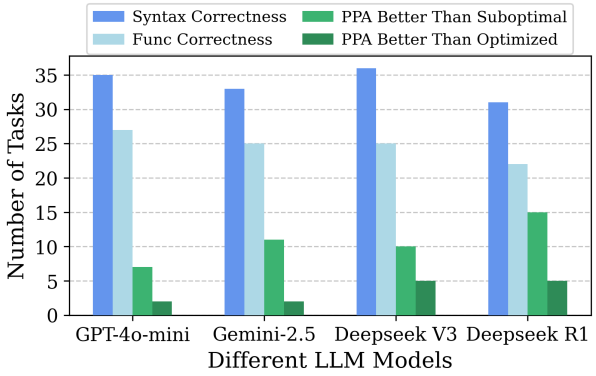


图 3：RTL-OPT 中次优与优化后的 RTL代码示例对比。

4 实验

本节介绍了为评估不同大语言模型在 RTL-OPT 基准测试上的优化能力而开展的实验。我们比较了多个大语言模型在优化 RTL代码方面的表现，包括**GPT-4o-mini**、**Gemini-2.5**、**Deepseek V3**和**Deepseek R1**。实验重点评估优化在 PPA 指标方面的效果，以及优化设计的功能正确性。结果表明，与其他受评估的大语言模型相比，两个 Deepseek 模型展现出更强的优化能力。

4.1 基准测试结果总结

图 3 总结了对四个大语言模型进行的基准测试结果，展示了各大语言模型的语法正确性、功能正确性以及优化后的功耗、性能和面积质量。总体结果表明：❶ 与人工设计人员相比，大语言模型在寄存器传输级优化方面仍有显著的提升空间。❷ 我们的基准测试旨在贴近现实，提供一组具有挑战性的任务，以反映现实世界硬件设计中遇到的复杂性。

值得注意的是，所有大语言模型的整体表现都不太理想，这反映了 RTL-OPT 基准所面临的挑战。许多大语言模型有 10 个以上的已优化分支语句未能保持功能正确性。Deepseek R1 能够成功优化约 15 个次优设计，并且在约 5 个设计上优于人工设计人员的方案。

比较这四个大语言模型时，Deepseek R1 在功耗、性能和面积方面通常优于其他模型。然而，与其他模型相比，Deepseek R1 的功能差异发生率也更高。相比之下，GPT-4o-mini 和 Gemini-2.5 等模型虽然保持了较高的语法正确性，但在功耗、性能和面积方面取得的改进较少。这表明，它们的优化策略要么更为保守，要么缺乏有效的优化。

4.2 详细基准测试结果

图 3 中评估结果显示的一个现象是优化与功能之间的权衡。虽然 Deepseek R1

类别	GPT-4o-mini						Gemini-2.5						Deepseek R1					
	单元	*Area*	功耗 *(mW)*	*WNS (ns)*	*TNS (ns)*	检查	单元	*Area*	功耗 *(mW)*	*WNS (ns)*	*TNS (ns)*	检查	单元	*Area*	功耗 *(mW)*	*WNS (ns)*	*TNS (ns)*	检查
加法器	-	-	-	-	-	✘	-	-	-	-	-	✘✘	433	511.52	368.04	0.00	0.00	✔
加法器_选择器	450	439.17	264.34	0.00	0.00	✔	450	439.17	264.34	0.00	0.00	✔	306	350.06	225.74	0.00	0.00	✔
alu_64位	1683	1645.21	887.78	0.00	0.00	✔	1683	1645.21	887.78	0.00	0.00	✔	1697	1658.24	896.08	0.00	0.00	✔
alu_8bit	146	179.55	79.58	0.00	0.00	✔	146	179.55	79.58	0.00	0.00	✔	146	179.55	79.58	0.00	0.00	✘
计算	-	-	-	-	-	✘	888	1044.32	788.67	0.61	5.66	✔	755	859.98	617.26	0.60	4.83	✔
比较器	59	52.67	26.44	0.00	0.00	✔	54	47.61	24.64	0.00	0.00	✔	46	42.29	20.87	0.00	0.00	✔
比较器_16位	-	-	-	-	-	✘	102	88.31	44.57	0.00	0.00	✔	**70**	**65.70**	**30**	**0.00**	**0.00**	✔
比较器_2位	10	9.58	4.51	0.00	0.00	✔	10	8.78	3.82	0.00	0.00	✔	**9**	**7.45**	**3.75**	**0.00**	**0.00**	✔
比较器_4位	21	18.35	9.25	0.00	0.00	✔	21	18.35	9.25	0.00	0.00	✔	-	-	-	-	-	✘✘
比较器_8位	-	-	-	-	-	✘	48	43.36	18.56	0.00	0.00	✔	-	-	-	-	-	✘✘
解码器 _6bit	**86**	**71.29**	**19.76**	**0.00**	**0.00**	✔	87	71.55	19.54	0.00	0.00	✔	96	76.61	22.56	0.00	0.00	✘
解码器 _8bit	312	246.85	50.28	0.00	0.00	✔	**308**	**246.85**	**49.75**	**0.00**	**0.00**	✔	**329**	**267.33**	**54.11**	**0.00**	**0.00**	✔
分频器_16 位	3445	3377.67	5030	-3.32	-73.2	✔	2461	2372.72	3750	7.22	156.	✔	3445	3377.67	5030	3.32	73.2	✔
分频器_32位	14400	14295.90	23800	10.19	468.	✔	-	-	-	-	-	✔	-	-	-	-	-	✘✘
分频器_4bit	-	-	-	-	-	✔	-	-	-	-	-	✘	39	40.43	21.58	0.00	0.00	✘
分频器_8bit	571	575.36	681.84	0.50	4.57	✔	**266**	**264.14**	**188.61**	**0.51**	**3.16**	✔	-	-	-	-	-	✘✘
fsm	89	128.74	92.56	0.00	0.00	✔	-	-	-	-	-	✘✘	73	92.04	51.44	0.00	0.00	✔
fsm_编码	287	416.82	337.09	0.00	0.00	✔	**151**	**302.18**	**273.32**	**0.00**	**0.00**	✔	151	302.18	273.32	0.00	0.00	✔
gray	-	-	-	-	-	✘	51	69.69	66.78	0.00	0.00	✔	52	70.22	65.86	0.00	0.00	✔
mac	-	-	-	-	-	✘	-	-	-	-	-	✘	352	632.02	787.88	0.00	0.00	✘
mul	315	378.52	421.67	0.05	0.13	✔	315	378.52	421.67	0.05	0.13	✔	315	378.52	421.67	0.05	0.13	✔
mul_sub	233	337.02	256.53	0.00	0.00	✔	234	341.81	255.66	0.00	0.00	✔	234	336.22	259.8	0.00	0.00	✔
mux	-	-	-	-	-	✘	-	-	-	-	-	✘	-	-	-	-	-	✘
mux_编码	-	-	-	-	-	✘	-	-	-	-	-	✘	61	73.42	35.39	0.00	0.00	✔
饱和_加	-	-	-	-	-	✘✘	18	67.56	65.62	0.00	0.00	✔	-	-	-	-	-	✘✘
选择器	21	44.69	39.6	0.00	0.00	✔	18	39.37	36.06	0.00	0.00	✔	18	39.37	36.06	0.00	0.00	✔
sub_16 位	132	136.19	77.92	0.00	0.00	✔	132	136.19	77.92	0.00	0.00	✔	**122**	**126.88**	**71.97**	**0.00**	**0.00**	✔
sub_32位	278	252.17	146.9	0.00	0.00	✔	278	252.17	146.9	0.00	0.00	✔	-	-	-	-	-	✘
sub_4bit	N/A	N/A	N/A	N/A	N/A	✔	-	-	-	-	-	✘	14	19.42	10.61	0.00	0.00	✔
sub_8bit	27	41.50	23.91	0.00	0.00	✔	-	-	-	-	-	✘	**27**	**41.50**	**23.91**	**0.00**	**0.00**	✔
add_sub	124	130.34	101.74	0.00	0.00	✔	124	130.34	101.74	0.00	0.00	✔	-	-	-	-	-	✘
addr_计算	78	131.40	96.07	0.03	0.06	✔	-	-	-	-	-	✘	-	-	-	-	-	✘
mult_if	10	10.91	3.53	0.00	0.00	✔	10	10.91	3.53	0.00	0.00	✔	-	-	-	-	-	✘
多路复用器_large	81	96.82	40.84	0.00	0.00	✔	-	-	-	-	-	✘	89	100.02	50.57	0.00	0.00	✔
寄存器	4625	9226.74	7540	0.00	0.00	✔	-	-	-	-	-	✘✘	3731	8745.55	7710	0.00	0.00	✔
票 _机器	-	-	-	-	-	✘	-	-	-	-	-	✘	-	-	-	-	-	✘

表 4：使用 RTL-OPT 基准测试，对 GPT-4o-mini、Gemini-2.5 和 Deepseek R1 优化的所有设计进行 PPA 质量（Design Compiler 编译_*ultra*，*1* 纳秒）和功能正确性评估。

在 PPA 方面表现出最显著的改进，但也最容易引入功能错误。相比之下，GPT-4o-mini 和 Gemini-2.5 采取了更为均衡的方法，在优化 PPA 的同时保持了语法正确性。这表明，Deepseek R1 的激进优化虽然有效，但往往会增加错误，尤其是在时序或控制逻辑复杂的设计中。相反，GPT-4o-mini 和 Gemini-2.5 虽然不那么激进，却保持了功能正确性，并实现了有意义的 PPA 改进。

例如，在 sub_32bit 设计中，DeepseekR1 将面积降低了 9%，将功耗降低了 13%，但也导致了功能差异。这表明，Deepseek R1 激进的优化方法虽然能有效改善功耗、性能和面积，但也可能增加出错的概率，尤其是在时序或控制逻辑复杂的设计中。

表 4 总结了各大语言模型优化设计的功耗、性能和面积表现，并将其与给定的次优 RTL 和优化后的 RTL（黄金参考）进行比较。绿色单元表示功耗、性能[illegible]次优 RTL。**粗体绿色单元**则表示功耗、性能和面积优于[illegible]该表还展示了验证后的功能正确性（Func 列），其中 ✔ 和 ✘ 表示验证结果。 ✘✘ 表示相应设计存在语法[illegible]无法通过 Design Compiler 综合。

关于实验结果，我们总结了四种大语言模型在语法正确性、功能正确性和 PPA 改进方面的表现。1) **GPT-4o-mini** 正确性表现良好，语法正确率为 97.2%，功能正确率为 75%。不过，其生成的代码中仅有 19.4% 的 PPA 优于次优版本。2) 同样，**Gemini-2.5** 呈现出与 GPT-4o-mini 相同的趋势：功能正确率相对较高，但 PPA 优化性能较低。3) 对于 **Deepseek V3**，其语法正确率最高，达到 100%；功能正确率为 69.4%，与 Gemini-2.5 相同。它在所有指标上表现均衡。4) 相比之下，**Deepseek R1**的语法正确率为 86.1%，功能正确率为 61.1%；尽管功能正确率较低，其生成的代码中有 41.7% 的 PPA 优于次优版本，13.9% 优于设计者方案。

除了定量结果之外，我们还随机检查了 40 个大语言模型优化设计通过语法检查但未通过功能验证的案例。我们观察到三种主要的失败模式：**控制逻辑不一致**（例如比较器中的布尔条件错误）、**过于激进的流水线化**（例如违反有限状态机中的延迟要求），以及**不当的资源共享**（例如由于寄存器复用而产生过时数据）。这些结果表明，大语言模型的错误往往源于细微的设计语义问题，而非表层语法问题。

5 结论

本文介绍了RTL-OPT，这是一个旨在提升集成电路设计质量的RTL代码优化基准测试。RTL-OPT包括

36个手工设计的数字集成电路设计，每个都包含次优和优化后的RTL代码，从而能够评估大语言模型生成的RTL。集成的评估框架可验证功能正确性并量化功耗、性能和面积改进，为评估硬件设计中的生成式人工智能模型提供标准化方法。RTL-OPT具有显著潜力，可通过提供有价值的见解并推动进步来影响人工智能辅助的集成电路设计。

参考文献

[1] 2021. Synoposys Design Compiler®。h t t p s:// w ww.synopsys.co m/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html。[2] 2021. VCS® 功能验证解决方案。https://www.synopsys.com/verification/simulation/vcs.html。[3] 2023. Formality® 等价性检查。h t t p s:// w w w.synopsys.co m/implementation-and-signoff/signoff/formality-equivalence-checking.html。[4] Ahmed Allam 和 Mohamed Shalan。2024 年。RTL-Repo：用于评估大语言模型在大型RTL设计项目中表现的基准测试。*arXiv* 预印本 *arXiv:2405.17378*（2024 年）。[5] Kaiyan Chang、Ying Wang、Haimeng Ren、Mengdi Wang、Shengwen Liang、Yinhe Han、Huawei Li 和 Xiaowei Li。2023 年。ChipGPT：自然语言硬件设计的发展距离我们还有多远？*arXiv* 预印本 *arXiv:2305.14019*（2023 年）。[6] Fan Cui、Chenyang Yin、Kexing Zhou、Youwei Xiao、Guangyu Sun、Qiang Xu、Qipeng Guo、Demin Song、Dahua Lin、Xingcheng Zhang 等。2024 年。OriGen：通过代码到代码增强和自我反思改进RTL代码生成。*arXiv* 预印本 *arXiv:2407.16237*（2024 年）。[7] Matthew DeLorenzo、Vasudev Gohil 和 Jeyavijayan Rajendran。2024 年。CreativEval：评估基于大语言模型的硬件代码生成的创造力。*arXiv* 预印本 *arXiv:2404.08806*（2024 年）。[8] Wenji Fang、Jing Wang、Yao Lu、Shang Liu、Yuchao Wu、Yuzhe Ma 和 Zhiyao Xie。2025 年。电路基础模型综述：面向超大规模集成电路设计和电子设计自动化的基础人工智能模型。*arXiv* 预印本 *arXiv:2504.03711*（2025 年）。[9] Yonggan Fu、Yongan Zhang、Zhongzhi Yu、Sixu Li、Zhifan Ye、Chaojian Li、Cheng Wan 和 Yingyan Celine Lin。2023 年。GPT4AIGChip：借助大语言模型实现新一代人工智能加速器设计自动化。载于国际计算机辅助设计会议（*ICCAD*）。[10] Chia-Tung Ho、Haoxing Ren 和 Brucek Khailany。2024 年。Verilogcoder：具有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编码智能体。*arXiv* 预印本 *arXiv:2408.08927*（2024 年）。[11] Mingjie Liu、Teodor-Dumitru Ene、Robert Kirby、ChrisCheng、Nathaniel Pinckney、Rongjian Liang、Jonah Alben、Himyanshu Anand、Sanmitra Banerjee、Ismet Bayraktaroglu 等。2023 年。ChipNeMo：用于芯片设计的领域适配大语言模型。*arXiv* 预印本 *arXiv:2311.00176*（2023 年）。[12] Mingjie Liu、Nathaniel Pinckney、Brucek Khailany 和 Haoxing Ren。2023 年。VerilogEval：评估大语言模型的Verilog代码生成能力。*arXiv* 预印本 *arXiv:2309.07544*（2023 年）。[13] Mingjie Liu、Yun-Da Tsai、Wenfei Zhou 和 Haoxing Ren。2024 年。CraftRTL：利用正确性由构造保证的非文本表示和定向代码修复，为Verilog代码模型生成高质量合成数据。*arXiv* 预印本 *arXiv:2409.12993*（2024 年）。[14] Shang Liu、Wenji Fang、Yao Lu、Qijun Zhang、Hongce Zhang 和 Zhiyao Xie。2024 年。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（*TCAD*）（2024 年）。[15] Shang Liu、Yao Lu、Wenji Fang、Mengming Li 和 Zhiyao Xie。2024 年。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。载于国际计算机辅助设计会议（*ICCAD*）。[16] Yi Liu、Changran Xu、Yunhao Zhou、Zeju Li 和 Qiang Xu。2025 年。DeepRTL：利用统一表示模型衔接Verilog理解与生成。*arXiv* 预印本 *arXiv:2502.15832*（2025 年）。[17] Yao Lu、Shang Liu、Qijun Zhang 和 Zhiyao Xie。2024 年。RTLLM：用于大语言模型RTL设计生成的开源基准测试。载于亚洲及南太平洋设计自动化会议（*ASP-DAC*）。[18] 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*（2024 年）。[19] Nathaniel Pinckney、Christopher Batten、Mingjie Liu、Haoxing Ren 和 Brucek Khailany。2024 年。重新审视VerilogEval：更新的大语言模型、上下文学习和从规范到RTL的任务。*arXiv* 预印本 *arXiv:2408.11053*（2024 年）。[20] Nathaniel Pinckney、Chenhui Deng、Chia-Tung Ho、Yun-Da Tsai、Mingjie Liu、Wenfei Zhou、Brucek Khailany 和 Haoxing Ren。2025 年。全面的Verilog设计问题：用于评估大语言模型和智能体在RTL设计与验证方面表现的新一代基准数据集。*arXiv* 预印本 *arXiv:2506.14074*（2025 年）。[21] Si2。2018 年。NanGate 45纳米开放单元库。[22] Shailja Thakur、Jason Blocklove、Hammond Pearce、Benjamin Tan、Siddharth Garg 和 Ramesh Karri。2023年。AutoChip：利用大语言模型反馈实现 HDL 生成自动化。*arXiv* 预印本 *arXiv:2311.04887*（2023年）。[23] Yiting Wang、Wanghao Ye、Ping Guo、Yexiao He、Ziyao Wang、Bowei Tian、Shwai He、Guoheng Sun、Zheyu Shen、Sihan Chen 等。2025年。SymRTLO：通过大语言模型和神经元启发的符号推理增强 RTL 代码优化。*arXiv* 预印本 *arXiv:2504.10369*（2025年）。[24] Clifford Wolf、Johann Glaser 和 Johannes Kepler。2013年。Yosys——一款免费的 Verilog 综合套件。发表于奥地利微电子研讨会（*Austrochip*）。[25] Zhihao Xu、Bixin Li 和 Lulu Wang。2025年。通过时序逻辑蜕变重新思考基于大语言模型的 RTL 代码优化。arXiv:2507.16808 [cs.SE] https: //arxiv.org/abs/2507.16808[26] Xufeng Yao、Yiwen Wang、Xing Li、Yingzhao Lian、Ran Chen、Lei Chen、Mingxuan Yuan、Hong Xu 和 Bei Yu。2024年。RTLRewriter：大模型辅助 RTL 代码优化的方法。*arXiv* 预印本 *arXiv:2409.11414*（2024年）。[27] Yang Zhao、Di Huang、Chongxiao Li、Pengwei Jin、Ziyuan Nan、Tianyun Ma、Lei Qi、Yansong Pan、Zhenxing Zhang、Rui Zhang 等。2024年。CodeV：通过多级摘要赋能大语言模型进行 Verilog 生成。*arXiv* 预印本 *arXiv:2407.10424*（2024年）。

RTL-OPT 的全面功耗、性能和面积质量比较

由于篇幅有限，我们将完整的功耗、性能和面积评估结果移至附录。本节完整比较了 RTL-OPT 中次优与已优化的 RTL 设计。综合和评估均使用 Synopsys Design Compiler 和 Yosys。由于功耗、性能和面积评估本质上涉及权衡，数值越小表示设计质量越好。这些结果通过展示不同综合约束下的完整数值证据，对正文内容进行了补充。

A.1 Design Compiler compile_ultra，1纳秒下的结果

表 5 报告了使用 Synopsys DC 在“compile ultra”设置下、采用宽松的 1 纳秒时钟周期综合时，全部 36 个 RTL-OPT 设计的完整 PPA 指标。次优 RTL 代码和已优化 RTL 代码均经过评估，从而能够进行直接比较。这一设置凸显了专家优化后的 RTL 在功耗、性能和面积方面始终优于次优版本，证明了 RTL-OPT 作为基准测试的可靠性。

	设计列表	提供的次优设计							提供的优化设计						
		DC 结果(*compile_ultra*，*1*纳秒)					Yosys 结果		DC 结果(编译_*ultra*，*1*纳秒)					Yosys 结果	
		单元面积	功耗		最差负裕量	总负裕量	单元	Area	单元	Area	功耗	最差负裕量	总负裕量	单元	Area
1	加法器	397	510.4	370.4	0.0	0.0	323	456.2	372	477.2	351.3	0.0	0.0	266	413.9
2	加法器_选择器	450	439.2	264.3	0.0	0.0	363	428.3	306	225.7	0.23	0.0	0.0	241	318.4
3	alu_64位	1683	1645	887.8	0.0	0.0	1411	1554	1680	1557	677.1	0.0	0.0	1094	1307
4	alu_8bit	146	180	75.58	0.0	0.0	152	163.1	115	138	68.43	0.0	0.0	130	158.8
5	计算	888	1044	0.78	-0.61	-5.66	774	900.4	755	860	0.62	-0.60	-4.83	567	661.8
6	比较器	63	56.39	0.03	0.0	0.0	40	40.70	45	41.76	0.02	0.0	0.0	40	40.70
7	比较器_16位	102	88.31	44.57	0.0	0.0	64	62.24	116	105.3	38.83	0.0	0.0	104	105.1
8	比较器_2位	10	9.58	4.51	0.0	0.0	11	9.31	10	8.78	3.82	0.0	0.0	10	9.31
9	比较器_4位	23	21.28	10.86	0.0	0.0	22	21.55	21	18.35	9.25	0.0	0.0	18	17.29
10	比较器_8位	46	44.42	22.8	0.0	0.0	45	48.15	47	41.23	20.29	0.0	0.0	33	30.86
11	解码器 _6bit	97	76.61	22.74	0.0	0.0	93	99.22	87	71.55	19.54	0.0	0.0	91	78.20
12	解码器 _8bit	317	257.2	50.67	0.0	0.0	390	410.7	312	246.8	50.28	0.0	0.0	300	250
13	分频器_16 位	3471	3428	5088	-3.43	-75.6	1264	1412	1565	1560	2091	-3.83	-72.1	662	760.2
14	分频器_32位	14400	14296	23796	-10.69	-486	-	-	6724	6703.20	11335	-10.19	-468	2419	2941
15	分频器_4bit	39	40.43	21.58	0.0	0.0	52	59.05	33	33.78	15.9	0.0	0.0	39	40.96
16	分频器_8bit	571	575.4	681.8	-0.50	-4.57	302	339.7	322	330.37	275.2	-0.39	-2.43	171	189.7
17	fsm	89	128.7	92.56	0.0	0.0	85	138.6	73	92.04	51.44	0.0	0.0	70	97.09
18	fsm_编码	242	387.0	320.6	0.0	0.0	179	352.7	155	305.63	276.97	0.0	0.0	170	319.2
19	gray	48	67.30	85.32	0.0	0.0	63	85.92	51	69.69	66.78	0.0	0.0	67	88.84
20	mac	410	717.9	868.2	0.0	0.0	532	831.5	319	548.23	735.25	0.0	0.0	529	707
21	mul	315	378.5	421.7	-0.05	-0.13	399	485.5	315	378.5	421.7	-0.05	-0.13	397	476.4
22	mul_sub	234	338.09	262.35	0.0	0.0	299	352.9	233	337.02	256.53	0.0	0.0	289	332.5
23	mux	25	31.92	10.38	0.0	0.0	8	8.51	25	21.81	8.38	0.0	0.0	34	42.29
24	mux_编码	125	140.7	0.43	0.0	0.0	-	-	34	36.18	0.13	0.0	0.0	-	-
25	饱和_加	24	69.43	67.53	0.0	0.0	58	97.36	18	67.56	65.77	0.0	0.0	42	78.47
26	选择器	18	39.37	36.06	0.0	0.0	17	42.56	18	38.04	35.32	0.0	0.0	15	37.24
27	sub_16 位	132	136.2	77.92	0.0	0.0	93	96.56	124	131.9	75.46	0.0	0.0	92	98.95
28	sub_32位	270	251.9	148.9	0.0	0.0	189	191.5	265	244.7	142.52	0.0	0.0	188	200
29	sub_4bit	12	18.35	9.29	0.0	0.0	21	22.34	10	17.82	9.29	0.0	0.0	20	22.08
30	sub_8bit	25	41.76	24.96	0.0	0.0	45	47.88	20	36.97	19.26	0.0	0.0	46	48.15
31	add_sub	164	183.27	118.4	0.0	0.0	155	164.4	124	130.34	101.7	0.0	0.0	179	202.7
32	addr_计算	78	131.40	96.07	-0.03	-0.06	197	222.9	82	125.55	90.27	-0.01	-0.01	101	124.7
33	mult_if	10	10.91	3.53	0.0	0.0	12	10.9	11	10.11	4.13	0.0	0.0	12	10.91
34	多路复用器_large	81	97.62	48.27	0.0	0.0	65	90.17	81	96.82	40.84	0.0	0.0	112	120.5
35	寄存器	3731	8745	7712	0.0	0.0	4500	9735	3720	8744	7708	0.0	0.0	4507	9668
36	张票 _机器	36	58.52	60.31	0.0	0.0	29	47.88	22	32.45	36.03	0	0	29	47.88

表 5：RTL-OPT 提供的次优设计与优化设计的功耗、性能和面积质量对比（compile_ultra，1 纳秒）。使用商业版 Design Compiler 和开源 Yosys 进行 RTL 设计综合和功耗、性能和面积评估。这些功耗、性能和面积指标通常存在权衡，数值越小表示性能越好。

A.2 Design Compiler 在 0.1 纳秒下的编译结果

表 6 展示了相同 36 个设计在更激进的 0.1 纳秒时钟周期约束下的完整 PPA 指标。与 1 纳秒设置相比，这些结果体现了 PPA 指标之间更明显的权衡，其中激进的时序优化有时会增加功耗或面积。尽管如此，优化后的 RTL 始终优于次优 RTL，进一步证明 RTL-OPT 反映了现实且具有意义的优化挑战。

	设计列表	提供的次优设计								提供的优化设计							
		DC 结果（编译，*0.1* 纳秒）					Yosys 结果			DC 结果（编译，*0.1* 纳秒）					Yosys 结果		
		单元面积	功耗	总负裕量	最差负裕量		布线	单元面积		单元面积	功耗	总负裕量	最差负裕量		布线	单元面积	
1	加法器	626	688.4	5.1	0.57	16.0	501	323	456.2	531	639.7	4.26	0.46	13.2	440	266	413.9
2	加法器_选择器	813	812.1	4.85	0.39	10.8	724	363	428.3	514	522.6	3.49	0.42	12.0	490	241	318.4
3	alu_64位	3248	3028	16.95	0.51	29.7	2358	1411	1554	1706	1748.9	8.58	0.74	42.7	1791	1094	1307
4	alu_8bit	402	370	1.89	0.26	1.91	257	152	163.1	244	245.8	1.15	0.40	2.92	229	130	158.8
5	计算	670	997.5	6.59	3.22	42.6	1210	774	900.4	534	761.8	5.25	3.24	44.3	927	567	661.8
6	比较器	109	98.15	0.43	0.19	0.19	111	40	40.70	90	80.60	0.34	0.21	0.21	108	40	40.70
7	比较器_16位	296	244.4	1.06	0.18	0.52	141	64	62.24	151	169.2	0.75	0.19	0.52	357	104	105.1
8	比较器_2位	14	14.36	0.06	0.02	0.05	40	11	9.31	16	13.03	0.05	0.01	0.02	28	10	9.31
9	比较器_4位	25	25.80	0.11	0.06	0.15	83	22	21.55	29	23.14	0.09	0.10	0.22	45	18	17.29
10	比较器_8位	73	71.29	0.31	0.12	0.33	170	45	48.15	81	67.30	0.28	0.12	0.34	77	33	30.86
11	解码器 _6bit	204	208	0.37	0.18	9.22	106	93	99.22	132	106.7	0.32	0.04	2.53	195	91	78.20
12	解码器 _8bit	781	855.9	1.28	0.27	58.8	407	390	410.7	435	373.2	0.83	0.09	20.8	617	300	250
13	分频器_16 位	5037	5045	66.3	3.88	89.4	2385	1264	1412	2543	2426	26.71	4.10	90.4	2116	662	760.2
14	分频器_32位	21348	19849	275.51	12.26	586	-	-	-	16434	16053	164	13.68	875	8341	2419	2941
15	分频器_4bit	85	84.85	0.48	0.32	1.29	112	52	59.05	64	56.13	0.23	0.17	0.59	133	39	40.96
16	分频器_8bit	744	731.8	6.94	1.21	13.5	539	302	339.7	557	535.5	3.67	1.10	10.8	527	171	189.7
17	fsm	149	192.9	1.06	0.28	4.40	209	85	138.6	106	129.8	0.59	0.24	2.70	157	70	97.09
18	fsm_编码	353	488.9	3.6	0.41	10.8	380	179	352.7	334	426.7	3.07	0.39	8.11	342	170	319.2
19	gray	100	109.1	0.92	0.28	2.04	142	63	85.92	81	94.70	1.04	0.23	2.34	145	67	88.84
20	mac	563	880.7	8.46	1.23	20.9	737	532	831.5	477	701.4	7.26	1.38	16.2	699	529	707
21	mul	311	453	3.64	1.26	9.69	498	399	485.5	270	426.4	3.43	1.25	9.62	494	397	476.4
22	mul_sub	634	623.24	4.17	0.75	8.08	494	299	352.9	614	603.8	4.12	0.71	8.11	480	289	332.5
23	mux	40	38.30	0.11	0.01	0.07	27	8	8.51	25	31.92	0.1	0.02	0.14	68	34	42.29
24	mux_编码	125	140.71	0.43	0.07	0.59	-	-	-	34	36.18	0.13	0.08	0.58	-	-	-
25	饱和_加	159	176.6	1.31	0.27	2.25	122	58	97.36	127	140.9	1.08	0.36	2.86	100	42	78.47
26	选择器	38	56.4	0.44	0.11	0.38	42	17	42.56	31	49.74	0.39	0.08	0.24	29	15	37.24
27	sub_16 位	234	223.2	1.31	0.27	3.48	220	93	96.56	302	263.1	1.39	0.30	3.94	181	92	98.95
28	sub_32位	542	502.2	2.94	0.31	8.49	444	189	191.5	518	452.2	2.51	0.36	9.22	370	188	200
29	sub_4bit	37	36.71	0.17	0.10	0.35	52	21	22.34	31	29.79	0.15	0.13	0.40	43	20	22.08
30	sub_8bit	135	122.6	0.65	0.15	0.95	108	45	47.88	129	111.9	0.57	0.26	1.47	90	46	48.15
31	add_sub	496	444.2	2.57	0.31	4.20	277	155	164.4	387	356.9	2.13	0.37	4.87	323	179	202.7
32	addr_计算	405	388.1	3.07	0.62	8.56	340	197	222.9	229	214.4	1.8	0.60	8.47	192	101	124.7
33	mult_if	17	15.96	0.052	0.05	0.05	39	12	10.9	15	14.90	0.046	0.09	0.09	34	12	10.91
34	多路复用器_较大	296	273.45	0.84	0.11	0.85	210	65	90.17	164	176.62	0.74	0.13	1.01	270	112	120.5
35	寄存器	5003	9780	79.63	0.40	245	8096	4500	9735	4481	9583	77.87	0.37	244	8140	4507	9668
36	张票 _机器	53	74.48	0.77	0.22	1.79	73	29	47.88	48	51.34	0.45	0.17	0.83	73	29	47.88

表 6：RTL-OPT 提供的次优与已优化设计的 PPA 质量比较（编译，0.1 纳秒）。使用商用 Design Compiler 和开源 Yosys 进行 RTL 设计综合和 PPA 评估。这些 PPA 指标通常存在权衡，数值越小表示性能越好。

B 大语言模型优化设计的功耗、性能和面积与功能正确性

由于篇幅限制，详细的实验结果移至附录。结果报告了使用 RTL-OPT 基准测试对 **GPT-4o-mini**、**Gemini-2.5**、**Deepseek V3** 和 **Deepseek R1** 优化的所有设计的 PPA 质量和功能正确性。

表 4 和表 7 总结了各大语言模型优化设计的功耗、性能和面积表现，并将其与提供的次优 RTL 和优化后的 RTL（黄金参考）进行比较。绿色单元表示其功耗、性能和优 RTL，而**粗体绿色单元**表示其功耗、性能和面积超过了（金参考）。表格还展示了验证后的功能正确性（Func 列），其中 ✔ 和 ✘ 分别表示验证结果。 ✘✘ 表示相应设计存在语法错误，无法通过 Design Compil 综合。

类别	GPT-4o-mini 单元	*Area*	功耗（毫瓦）	最差负裕量（纳秒）	总负裕量（纳秒）	检查	Gemini-2.5 单元	*Area*	功耗（毫瓦）	最差负裕量（纳秒）	总负裕量（纳秒）	检查
加法器	-	-	-	-	-	✘	-	-	-	-	-	✘✘
加法器_选择器	813	812.1	4.85	-0.39	-10.8	✔	813	812.10	4.85	-0.39	-10.8	✔
alu_64位	3248	3028	16.95	-0.51	-29.7	✔	3248	3028	16.95	-0.51	-29.7	✔
alu_8bit	402	370	1.89	-0.26	-1.91	✔	402	370.01	1.89	-0.26	-1.91	✔
计算	-	-	-	-	-	✘	670	997.50	6.59	-3.22	-42.6	✔
比较器	96	83.26	0.38	-0.22	-0.22	✔	100	88.05	0.38	-0.22	-0.22	✔
比较器_16位	-	-	-	-	-	✘	296	244.45	1.06	-0.18	-0.52	✔
比较器_2位	14	14.36	0.06	-0.02	-0.05	✔	16	13.03	0.05	-0.01	-0.02	✔
比较器_4位	33	29.26	0.18	-0.05	-0.15	✔	29	23.14	0.09	-0.10	-0.22	✔
比较器_8位	-	-	-	-	-	✘	79	83.52	0.37	-0.11	-0.31	✔
解码器 _6bit	**128**	**109.9**	**0.31**	**-0.03**	**-1.74**	✔	204	208.01	0.37	-0.18	-9.22	✔
解码器 _8bit	781	855.9	1.28	-0.27	-58.8	✔	**354**	**297.4**	**0.66**	**-0.11**	**-24.8**	✔
分频器_16 位	5037	5045	66.25	-3.88	-89.4	✔	3926	3895	51.75	-6.85	-155	✔
分频器_32位	16638	16211	171.11	-13.23	-846.3	✔	14529	13375	154.6	-13.24	-629	✔
分频器_4bit	85	84.85	0.48	-0.32	-1.29	✔	-	-	-	-	-	✘
分频器_8bit	744	731.8	6.94	-1.21	-13.5	✔	**531**	**513.38**	**3.58**	**-1.15**	**-11.0**	✔
fsm	149	192.8	1.06	-0.28	-4.40	✔	-	-	-	-	-	✘✘
fsm_编码	389	504.9	3.69	-0.41	-12.5	✔	357	448.21	3.16	-0.40	-7.95	✔
gray	-	-	-	-	-	✘	100	109	0.92	-0.28	-2.04	✔
mac	-	-	-	-	-	✘	-	-	-	-	-	✘
mul	311	453	3.64	-1.26	-9.69	✔	311	453	3.64	-1.26	-9.69	✔
mul_sub	634	623.2	4.17	-0.75	-8.08	✔	282	381.71	2.67	-1.11	-11.6	✔
mux	32	25.54	0.08	0.0	0.0	✔	32	25.54	0.08	0.0	0.0	✔
mux_编码	-	-	-	-	-	✘	-	-	-	-	-	✘
饱和_加法	-	-	-	-	-	✘✘	127	140.98	1.08	-0.36	-2.86	✔
选择器	30	53.73	0.48	-0.18	-0.49	✔	38	56.39	0.44	-0.11	-0.38	✔
sub_16 位	266	230.6	1.22	-0.27	-3.40	✔	266	230.62	1.22	-0.27	-3.40	✔
sub_32位	551	476.4	2.57	-0.35	-9.13	✔	551	476.41	2.57	-0.35	-9.13	✔
sub_4bit	50	49.48	0.23	-0.16	-0.59	✔	-	-	-	-	-	✘
sub_8bit	**121**	**102.7**	**0.51**	**-0.23**	**-1.30**	✔	-	-	-	-	-	✘
add_sub	496	444.22	2.57	-0.31	-4.20	✔	496	444.22	2.57	-0.31	-4.20	✔
addr_计算	405	388.09	3.07	-0.62	-8.56	✔	-	-	-	-	-	✘
mult_if	17	15.96	0.05	-0.05	-0.05	✔	17	15.96	0.05	-0.05	-0.05	✔
多路复用器_大型	296	273.45	0.84	-0.11	-0.85	✔	-	-	-	-	-	✘
寄存器	4615	9480.24	78.21	-0.36	-251	✔	-	-	-	-	-	✘✘
票 _机器	-	-	-	-	-	✘	-	-	-	-	-	✘

类别	Deepseek V3 单元	*Area*	功耗（毫瓦）	最差负裕量（纳秒）	总负裕量（纳秒）	检查	Deepseek R1 单元	*Area*	功耗（毫瓦）	最差负裕量（纳秒）	总负裕量（纳秒）	检查
加法器	**530**	**618.7**	**4.51**	**-0.43**	**-12.5**	✔	597	669.8	4.78	-0.41	-11.8	✔
加法器_选择器	813	812.1	4.85	-0.39	-10.8	✔	514	522.7	3.49	-0.42	-12.0	✔
alu_64位	3248	3028	16.95	-0.51	-29.7	✔	3248	3028	16.95	-0.51	-29.7	✔
alu_8bit	402	370	1.89	-0.26	-1.91	✔	402	370	1.89	-0.26	-1.91	✘
计算	670	997.5	6.59	-3.22	-42.6	✔	550	808.1	5.73	3.29	44.8	✔
比较器	-	-	-	-	-	✘	110	96.29	0.4	-0.19	-0.19	✔
比较器_16位	-	-	-	-	-	✘	**122**	**107.2**	**0.48**	**-0.17**	**-0.48**	✔
比较器_2位	**10**	**9.84**	**0.04**	**-0.03**	**-0.03**	✔	**13**	**11.17**	**0.04**	**-0.03**	**-0.04**	✔
比较器_4位	**25**	**25.80**	**0.11**	**-0.06**	**-0.15**	✔	-	-	-	-	-	✘✘
比较器_8位	73	71.29	0.31	-0.12	-0.33	✔	-	-	-	-	-	✘✘
解码器 _6bit	204	208.01	0.37	-0.18	-9.22	✔	-	-	-	-	-	✘
解码器 _8bit	781	856	1.28	-0.27	-58.8	✔	**392**	**330.1**	**0.72**	**-0.10**	**-23.1**	✔
分频器_16 位	-	-	-	-	-	✘	5037	5045	66.25	-3.88	-89.4	✔
分频器_32位	-	-	-	-	-	✘	-	-	-	-	-	✘✘
分频器_4bit	-	-	-	-	-	✘	-	-	-	-	-	✘
分频器_8bit	-	-	-	-	-	✘	-	-	-	-	-	✘✘
fsm	**100**	**129.5**	**0.6**	**-0.22**	**-2.62**	✔	106	129.8	0.59	-0.24	-2.70	✔
fsm_编码	372	466.8	3.46	-0.38	-9.36	✔	357	448.2	3.16	-0.40	-7.95	✔
gray	100	109.1	0.92	-0.28	-2.04	✔	100	109.1	0.92	-0.28	-2.04	✔
mac	-	-	-	-	-	✘	-	-	-	-	-	✘
mul	-	-	-	-	-	✘	270	426.4	3.43	-1.25	-9.62	✔
mul_sub	657	654.4	4.44	-0.76	-8.08	✔	675	654.1	4.46	-0.70	-7.97	✔
mux	32	25.54	0.08	0.0	0.0	✔	32	25.54	0.82	0.0	0.0	✔
mux_编码	-	-	-	-	-	✘	113	121.6	0.46	-0.09	-0.69	✔
饱和_加	127	140.9	1.08	-0.36	-2.86	✔	-	-	-	-	-	✘
选择器	38	56.39	0.44	-0.11	-0.38	✔	30	49.74	0.41	-0.18	-0.49	✔
sub_16 位	266	230.6	1.22	-0.27	-3.40	✔	**266**	**230.6**	**1.22**	**-0.27**	**-3.40**	✔
sub_32位	551	476.4	2.57	-0.35	-9.13	✔	-	-	-	-	-	✘
sub_4bit	37	37.24	0.18	-0.10	-0.30	✔	37	37.24	0.12	-0.10	-0.30	✔
sub_8bit	**121**	**102.7**	**0.51**	**-0.23**	**-1.30**	✔	**121**	**102.7**	**0.51**	**-0.23**	**-1.30**	✔
add_sub	496	444.2	2.57	-0.31	-4.20	✔	-	-	-	-	-	✘
addr_计算	229	214.40	1.8	-0.60	-8.47	✔	-	-	-	-	-	✘
mult_if	-	-	-	-	-	✘	-	-	-	-	-	✘
多路复用器_large	173	174.5	0.62	-0.12	-0.95 [10]	✔	173	174.5	0.62	-0.12	-0.95	✔
寄存器	-	-	-	-	-	✘	5003	9780	79.63	-0.40	-245	✔
票 _机器	-	-	-	-	-	✘	-	-	-	-	-	✘

表 7：GPT-4o-mini、Gemini-2.5、Deepseek V3 和 Deepseek R1 使用 RTL-OPT 基准测试优化的所有设计的功耗、性能和面积质量（Design Compiler编译，*0.1* 纳秒）及功能正确性。

B.1 大语言模型优化设计的功耗、性能和面积与功能正确性（DC compile，1 纳秒）

表 7 总结了 GPT-4o-mini、Gemini-2.5、Deepseek V3 和 Deepseek R1 在 RTL-OPT 基准上优化的设计的功耗、性能和面积结果及功能正确性检查。所有评估均采用 DC compile，并设置 0.1 纳秒的时钟周期。这补充了表 4 的结果；表 4 使用了较宽松的时序设置。

C 复现　　使用多种综合流程复现现有基准测试

C.1 使用多种综合流程复现 [26] 设计

表 8 给出了原始版本和专家优化版本设计使用我们的流程进行评估，涵盖三种综合设置：Yosys、DC compile（0.1 纳秒）和 DC compile ultra（1 纳秒）。这确保了不同基准测试之间的公平比较，并展示了我们的流程在复现先前工作的有效性。

设计列表	Yosys		DC（compile，0.1 纳秒）					DC（compile_ultra，1 纳秒）				
	单元	*Area*	单元	*Area*	功耗	*WNS*	*TNS*	单元	*Area*	功耗	*WNS*	*TNS*
案例1	18	57.99	-	-	-	-	-	-	-	-	-	-
分支语句1_opt	12	30.86	-	-	-	-	-	-	-	-	-	-
案例2	37	44.95	-	-	-	-	-	-	-	-	-	-
分支语句2_opt	37	44.95	-	-	-	-	-	-	-	-	-	-
案例3	107	129.3	289	255.1	2.25 毫瓦	-0.41	-3.23	26	98.95	74.85 微瓦	0.0	0.0
分支语句3_opt	102	127.7	289	255.1	2.25 毫瓦	-0.41	-3.23	26	98.95	74.85 微瓦	0.0	0.0
案例4	304	347.1	576	550.9	3.75 毫瓦	-0.51	-3.72	204	267.06	169.5 微瓦	0.0	0.0
分支语句4_优化	262	310.7	553	529.9	3.80 毫瓦	-0.50	-3.65	233	291.27	165.6 微瓦	0.0	0.0
案例5	10933	12457	6546	11104	173.7 毫瓦	-4.15	-413.6	6047	9668	19.99 毫瓦	-1	-56.61
分支语句5_opt	10504	11980	6044	10469	168.9 毫瓦	-4.22	-418.1	5962	9600	19.98 毫瓦	-1.02	-56.7
分支语句6	37	59.32	53	60.38	484.8 微瓦	-0.22	-0.68	28	41.76	42.93 微瓦	0.0	0.0
分支语句6_opt	21	28.99	23	28.99	284.1 微瓦	-0.15	-0.28	11	19.95	23.42 微瓦	0.0	0.0
案例7	24	38.84	22	34.31	418.7 微瓦	-0.15	-0.46	16	28.99	40.65 微瓦	0.0	0.0
分支语句7_opt	11	19.95	11	18.35	215.8 uW	-0.12	-0.25	8	15.96	21.90 uW	0.0	0.0
案例8	1471	1720	1353	2242	13.90 毫瓦	-0.92	-58.2	703	831.2	634.7 uW	0.0	0.0
分支语句8_opt	661	758.1	1771	1962	14.87 毫瓦	-0.95	-57.0	711	834.7	647.3 uW	0.0	0.0
案例9	1716	2010	1336	2420	13.31 毫瓦	-1.05	-65.7	729	886.3	636.4 uW	0.0	0.0
case9_opt	776	926.5	1757	1984	16.50 毫瓦	-1.00	-70.6	973	965.1	982.7 微瓦	0.0	0.0
案例10	24	43.89	25	31.92	103.8 微瓦	-0.02	-0.14	25	31.92	10.38 微瓦	0.0	0.0
分支语句10_opt	24	8.51	40	38.30	114.8 uW	-0.01	-0.07	25	21.81	8.38 uW	0.0	0.0
案例11	2	1.86	9	5.32	16.07 uW	0.0	0.0	1	1.06	252.5 nW	0.0	0.0
分支语句11_opt	1	1.86	6	2.93	7.28 微瓦	0.0	0.0	1	1.06	252.5 纳瓦	0.0	0.0
case12	1	1.86	3	0.0	374.8 纳瓦	0.0	0.0	3	0.0	37.48 纳瓦	0.0	0.0
分支语句12_opt	1	1.06	1	1.06	2.59 uW	0.0	0.0	1	1.06	258.8 nW	0.0	0.0
case13	3	1.86	3	5.59	16.93 uW	-0.03	-0.03	5	5.05	1.77 uW	0.0	0.0
分支语句13_opt	2	1.86	2	3.72	12.41 微瓦	-0.03	-0.03	5	5.05	1.77 微瓦	0.0	0.0
case14	4	1.86	6	7.98	28.15 微瓦	-0.10	-0.10	5	5.05	1.77 微瓦	0.0	0.0
分支语句14_优化	3	1.86	6	6.65	23.20 微瓦	-0.08	-0.08	5	5.05	1.77 微瓦	0.0	0.0

表 8：使用 Yosys 和 Design Compiler 综合流程，对 [26] 中 14个设计（包括原始版本和专家优化版本）进行复现的功耗、性能和面积结果。

C.2 [26]GPT优化设计的复现

表9和表10分别展示了基于GPT的方法和 [26] 自身优化策略所优化设计的功耗、性能和面积复现结果。这些结果有助于评估我们的流程应用于 RTL-OPT 基准之外设计时的泛化能力，并对不同优化方法进行公平比较。在统一的评估流程下，我们采用了相同的三种综合设置（Yosys、DC compile 0.1ns 和 DC compile ultra 1ns）。

设计列表	Yosys		Design Compiler（编译，0.1 纳秒）					Design Compiler（编译_ultra，1 纳秒）				
	单元	*Area*	单元	*Area*	功耗	*WNS*	*TNS*	单元	面积	功耗	*WNS*	*TNS*
案例1 _GPT	18	57.99	18	57.99	584.6 微瓦	-0.24	-0.24	18	57.99	61.13 微瓦	0.00	0.00
案例2 _GPT	37	44.95	-	-	-	-	-	-	-	-	-	
案例3 _GPT	107	129.3	289	255.1	2.25 毫瓦	-0.41	-3.23	26	98.95	74.85 微瓦	0.00	0.00
案例4 _GPT	290	326.6	637	602.8	4.04 毫瓦	-0.47	-3.33	205	265.5	159.5 微瓦	0.00	0.00
案例5 _GPT	10627	12090	5987	10483	165.9 毫瓦	-4.18	-408	5962	9600	19.98 毫瓦	-1.02	-56.72
案例6 _GPT	37	59.32	50	59.85	492.7 微瓦	-0.25	-0.73	25	38.57	40.19 微瓦	0.00	0.00
案例7 _GPT	26	44.69	30	43.62	522.1 微瓦	-0.15	-0.55	17	34.31	47.46 微瓦	0.00	0.00
案例8 _GPT	844	952.3	2129	1915	15.04 毫瓦	-0.67	-44.17	704	932.1	722.7 微瓦	0.00	0.00
案例9 _GPT	844	952.3	2129	1915	15.04 毫瓦	-0.67	-44.17	704	932.1	722.7 微瓦	0.00	0.00
案例10 _GPT	24	21.38	32	25.54	81.96 微瓦	0.0	0.0	25	21.81	8.382 微瓦	0.00	0.00
案例11 _GPT	3	1.862	3	5.59	16.86 微瓦	-0.03	-0.03	4	3.724	1.339 微瓦	0.00	0.00
case12 _GPT	1	1.064	1	1.06	2.59 微瓦	0.0	0.0	1	1.064	258.8 nW	0.00	0.00
case13 _GPT	3	5.586	6	5.59	20.58 uW	0.0	0.0	5	5.054	1.767 uW	0.00	0.00
case14 _GPT	2	3.724	5	4.26	16.58 uW	0.0	0.0	4	3.724	1.248 微瓦	0.00	0.00

表9：使用 Yosys 和 Design Compiler 综合流程复现 GPT-4 已优化设计的功耗、性能和面积结果。

设计列表	Yosys		Design Compiler（编译，0.1 纳秒）					Design Compiler（编译_ultra，1 纳秒）				
	单元	*Area*	单元	*Area*	功耗	*WNS*	*TNS*	单元	*Area*	功耗	*WNS*	*TNS*
案例1 _ours	14	39.9	13	40.96	431.3 uW	-0.21	-0.28	14	39.90	44.67 uW	0.0	0.0
案例2 _ours	37	44.95	76	69.43	383.9 uW	-0.17	-1.07	9	32.45	15.31 uW	0.0	0.0
案例3 _ours	107	129.3	289	255.1	2.251 毫瓦	-0.41	-3.23	26	98.95	74.85 微瓦	0.0	0.0
案例4 _ours	168	178.8	396	371.9	2.212 毫瓦	-0.44	-3.46	153	163.32	110.87 微瓦	0.0	0.0
案例5 _ours	10540	12016	6329	10642	170.7 毫瓦	-4.14	-409.54	5962	9600.21	19.98 毫瓦	-1.02	-56.72
case6 _ours	20	28.73	23	28.99	284.1 uW	-0.15	-0.28	11	19.95	23.42 uW	0.0	0.0
案例7 _ours	23	35.64	23	34.58	385.6 uW	-0.14	-0.47	14	27.40	35.59 uW	0.0	0.0
案例8 _ours	908	1063	2136	1916	14.75 毫瓦	-0.69	-44.12	709	939.78	746.77 uW	0.0	0.0
案例9 _ours	956	1173	2595	2354	19.54 毫瓦	-0.95	-61.83	847	1341.97	1.075 毫瓦	0.0	0.0
案例10 _ours	-	-	-	-	-	-	-	-	-	-	-	
案例11 _ours	1	1.862	7	4.788	15.60 uW	0.0	0.0	2	1.330	370.2 nW	0.0	0.0
case12 _ours	1	1.062	1	1.064	2.588 微瓦	0.0	0.0	1	1.064	258.8 纳瓦	0.0	0.0
case13 _ours	-	-	-	-	-	-	-	-	-	-	-	
case14 _ours	4	1.862	6	7.98	28.15 微瓦	-0.1	-0.1	5	5.054	1.767 微瓦	0.0	0.0

表10：使用 Yosys 和 Design Compiler 综合流程复现 RTLRewriter 对其自身设计进行优化后得到的功耗、性能和面积结果。

D 大语言模型评估结果与统计分析详述

我们在 RTL-OPT（36 个设计）上**将每个大语言模型评估重新运行 5 次**，并使用相同的提示模板以确保一致性。对于每次运行，我们记录了：(1) 语法正确性；(2) 功能正确性；(3) 功耗、性能和面积优于次优方案；(4) 功耗、性能和面积优于已优化方案。以下总结了对 6 种不同大语言模型（**包括两个开源大语言模型**）的实验结果。

我们总结统计结果，并对两组有代表性的大语言模型进行配对 t 检验：Gmini 与 DS-R1，以及 LLaMA 与 Qwen。更新终稿时，我们将在本附录中加入完整的**平均值** $\pm\sigma$ 和**配对 t 检验**结果。

D.1 语法正确性（36 项中的 N 项）

Run	GPT-4o	Gmini-2.5	DS V3	DS R1	QWEN	Llama
1	36	35	36	32	34	15
2	35	31	34	32	34	17
3	36	30	36	31	31	15
4	35	34	36	30	30	14
5	35	35	36	32	34	15
Mean	35.4	33	35.6	31.4	32.6	15.2
σ	1.0	1.5	1.3	0.8	0.9	1.1

D.2 功能正确性（36 项中的 N 项）

Run	GPT-4o	Gmini-2.5	DS V3	DS R1	QWEN	Llama
1	29	26	27	24	23	13
2	28	25	21	26	20	18
3	25	24	26	25	23	17
4	27	25	25	24	21	12
5	28	28	25	26	19	17
Mean	27.4	25.6	24.8	25	21.2	15.4
σ	1.5	1.5	2.3	1.8	1.6	2.7

D.3 功耗、性能和面积 >次优（36 项中的 N 项）

Run	GPT-4o	Gmini-2.5	DS V3	DS R1	QWEN	Llama
1	7	11	10	15	4	2
2	8	9	6	16	4	5
3	7	10	8	16	6	4
4	7	9	10	13	4	1
5	9	10	8	14	4	2
Mean	7.6	9.8	8.4	14.8	4.4	2.8
σ	0.9	0.8	1.5	1.3	1.1	1.6

D.4 PPA >已优化（N/36）

Run	GPT-4o	Gmini-2.5	DS V3	DS R1	QWEN	Llama
1	2	2	5	5	1	0
2	3	2	4	6	0	1
3	1	4	5	5	2	1
4	3	2	5	5	2	0
5	2	2	3	4	1	1
Mean	2.2	2.4	4.4	5	1.2	0.6
σ	0.8	0.9	1.0	0.7	0.8	0.5

D.5 对比：GPT-4o 与 DS-R1

指标	平均差	t 值	p 值	显著性	效应量
语法正确性	+3.8	$^{+6.53}$	0.0028	显著	2.92（非常大）
功能正确性	+1.6	$^{+1.67}$	0.169	不显著	0.75（中等）
功耗、性能和面积 >次优	-7.2	-5.81	0.004	显著	2.60（非常大）
功耗、性能和面积 >已优化	-2.8	-5.83	0.004	显著	2.61（非常大）

D.6 比较：LLaMA 与 Qwen

指标	平均差	t 值	p 值	显著性	效应量
语法正确性	-18.2	-48.40	$^{<0.0001}$	显著	-21.7（极端）
功能正确性	-5.8	-5.39	0.006	显著	-2.41（非常大）
功耗、性能和面积 >次优	+1.6	$^{+2.53}$	0.065	边际	0.72（中等）
功耗、性能和面积 >已优化	+0.6	$^{+1.34}$	0.251	不显著	0.30（小）