Bench4HLS：大语言模型在高层次综合代码生成中的端到端评估

M Zafir Sadik Khan、Kimia Azar、Hadi Kamali中佛罗里达大学电气与计算机工程系（*ECE*），美国佛罗里达州奥兰多市，邮编 *32816*{mzafirsadik.khan,azar, kamali}@ucf.edu

arXiv:2601.19941v1 [cs.AR] 16 Jan 2026

摘要——**过去两年，大语言模型（LLM）在代码生成方面展现出强大的能力，其中包括寄存器传输级（RTL）硬件设计。尽管大语言模型在高层次综合（HLS）中的应用相对不够成熟，但过去六个月，聚焦 HLS 与 RTL 的研究比例已从 1:10 变为 2:10，这表明人们越来越关注利用大语言模型进行高层设计输入，并依靠下游综合进行优化。这一趋势凸显了构建专门面向基于大语言模型的 HLS 的全面基准测试与评估框架的必要性。为此，我们推出 Bench4HLS，用于评估大语言模型生成的 HLS 设计。Bench4HLS 包含 170 个经过人工编写和验证的案例研究，涵盖从小型内核到复杂加速器，并从广泛使用的公共代码库中精选而来。该框架支持全自动评估，包括编译成功率、通过仿真验证的功能正确性，以及综合可行性与优化。尤为重要的是，Bench4HLS 集成了可插拔 API，可针对各种 HLS 工具链和架构分析功耗、性能和面积（PPA）；本文以 Xilinx Vitis HLS 展示该功能，并在 Catapult HLS 上进行了验证。通过提供结构化、可扩展且即插即用的测试平台，Bench4HLS 为 HLS 工作流中的大语言模型基准测试奠定了基础方法**[1]。

索引词——**大语言模型（LLM）、高层次综合（HLS）、设计空间探索（DSE）、功耗、性能和面积（PPA）效率。**

I. 引言

高层次综合（HLS）通过支持从高级语言（HLL）自动生成硬件，成为解决传统寄存器传输级（RTL）设计局限性的一种方案 [1], [2]。其目标是在无需掌握寄存器传输级编码知识的情况下提升生产力，并简化应用专用设计空间探索（DSE） [3]。过去十年间，HLS 已显著成熟，支持快速设计周期，并使硬件中的高级优化成为可能[4]–[7]。尽管如此，实现最佳的功耗、性能和面积（PPA）仍然极具挑战，通常需要深厚的专业知识和大量人工干预 [6], [8]。

如今，距离高层次综合（HLS）问世已逾十年，大语言模型（LLM）的迅速崛起及其在编程任务中的广泛应用，为硬件设计开辟了新的机遇。近期研究已开始探索大语言模型如何辅助高层次综合工作流（甚至通过直接生成代码取代其中部分环节） [9]–[17]。这些研究大多聚焦于设计流程中的孤立环节，例如通过插入和修改**编译指示**或进行代码重构，使代码更符合综合工具的要求；通过 [10], [14], 将通用 C/C++ 代码转换（修复或重构）为兼容 HLS 的内核；甚至通过基于推理的方法直接将 HLL 转换为寄存器传输级（RTL），从而完全绕过传统的高层次综合流程 [12], [15], [11]。

尽管利用大语言模型进行高层次综合的工作不断取得进展，对此类解的系统性评估仍然有限。为说明这一差距，另一方面，寄存器传输级评估已有 VerilogEval[18], RTLLM [19], 以及近期的 CVDP 基准测试 [20]等成熟工作提供支持。有人认为，直接从大语言模型生成寄存器传输级代码可能会削弱高层次综合的作用，但这种观点忽略了一个事实：高层次综合工具针对基于 ***FPGA*** 的加速进行了深度优化，并为设计空间探索提供了必要的抽象层 [21]。因此，基于大语言模型的高层次综合仍是至关重要的研究方向，而建立一个全自动、系统性的评估基准测试势在必行。近期的基准测试 HLS-Eval [22], 已在这一领域迈出了第一步，但其范围有限：尽管它侧重于语法和可综合性检查，却忽略了功耗、性能和面积分析，且仅依赖小规模内核。因此，仍缺乏能够对大语言模型生成的高层次综合代码进行全面评估的稳健框架，尤其是贯穿整个综合流程，并纳入实际部署所必需的结果质量（QoR）考量 [23]。

为弥补这一空白，我们提出了 Bench4HLS，这是一个全面的评估框架，旨在对高层次综合工作流中的大语言模型进行基准测试。表 I 展示了 Bench4HLS 相较于已有工作的主要优势，我们的贡献如下：*(i)* 多样化的基准测试套件：Bench4HLS 包含 170 个经过整理和验证的 HLS 设计任务，涵盖从小型内核到加速器级应用；这些任务收集自广泛使用的开源代码库，并通过黄金测试平台进行标准化，以确保可复现性。

(ii) 可插拔的功耗、性能和面积评估：Bench4HLS 集成了一个模块化应用程序接口，可用于对多个 HLS 工具链进行功耗、性能和面积分析；此处展示的是其与 Xilinx Vitis HLS 的集成（作为主要测试用例），并且已在 Catapult HLS 上成功测试。

（*iii*）设计空间探索（设计空间探索）：Bench4HLS 在评估循环中探索综合级（工具在环）设计空间探索，在固定设计预算下呈现帕累托前沿。（*iv*）严格的正确性验证：Bench4HLS 在多个阶段进行严格的功能验证，包括综合前 C 仿真、HLS 协同仿真以及实现后的网表验证，以确保只有时序达标且保持正确性的设计才能获得评分。

[1]与本研究相关的代码和资源已公开发布，地址为：https://github.com/zfsadik/Bench4HLS

通过结合数据集广度、多目标评估和自动化优化，Bench4HLS 提供了首个全面且具有实际意义的基准测试，用于评估由大语言模型驱动的自动化高层次综合设计方法[2]。

II. 背景与相关工作

A. 用于直接生成寄存器传输级代码的大语言模型

寄存器传输级是大语言模型代码生成研究中最受关注的硬件目标，相关方法涵盖微调 [24]，、智能体方法 [25]，[26] 和解码方法 [27]，[28] [29]。在基准测试与评估方面，已有多个基准测试确立了事实上的基线。VerilogEval 使用 HDLBits 风格的问题评估大语言模型，近期还加入了规格到寄存器传输级任务，并改进了测试框架；它仍是常用的比较基准 [30]。RTLLM 不仅检查语法和功能，还纳入了“设计质量”目标，并提供了包含任务和测试平台的开放代码库 [19]。

B. 用于高层次综合代码生成与综合的大语言模型

与寄存器传输级相比，关注使用大语言模型进行高层次综合的既有研究相对较少。高层次综合依赖隐式的编译器优化、层次化变换和综合约束，这给大语言模型带来了挑战：模型必须生成语法正确的代码，同时还要推理结构、时序和资源约束 [31]。这些“用于高层次综合的大语言模型”研究可分为四个主要类别：

(1) 可综合性与功耗、性能和面积优化。初步研究表明，大语言模型能够生成在功能上正确的高层次综合代码，但若缺乏硬件感知优化，其性能和面积表现仍不及专家设计 [11]。HLSPilot 等由反馈驱动的方法通过在生成过程中引入性能分析和综合约束来提升结果 [10]。

(2) 代码重构与修复。大语言模型已被用于将通用 C/C++转换为适用于高层次综合的内核，处理不受支持的结构并提升可综合性。虽然这些模型在基础转换方面效果显著，但在层次化架构和存储器数据流等复杂优化方面仍面临挑战 [12]，[15]。

*(3)*结构化推理与智能体。Syn- thAI [13]，LIFT [31]等框架以及智能体优化方法，通过推理、图分析和工具交互扩展大语言模型。这些系统可自动执行编译指示插入、将任务模块化，并迭代优化实现，从而提升综合成功率和设计质量 [33]。

*(4)*数据集与模型设计。为增强对高层次综合的支持，人们引入了专用数据集和模型架构。例如，SAGE-HLS 利用 AST 引导训练，在合成 HLS-C 语料库上进行训练，取得了较高的综合准确率，但功能正确性仍然有限 [34]。

C. 大语言模型生成的高层次综合代码的基准测试

基于大语言模型的高层次综合评估的基准测试仍处于早期阶段，目前仅有少数针对性工作：（1）Gai 等人 [32]提出了首批将自然语言提示与高层次综合设计相结合的基准测试之一。其框架重点进行语法和语义正确性检查，旨在根据文本规范实现可靠的自动化高层次综合代码生成；（2）HLS-Eval [22] 提供了一个可扩展的基于 Python 的基础设施，可与商业高层次综合工具集成。该基础设施提供了一组经过整理、配备测试平台和自然语言提示的设计，使用户能够对大语言模型生成的高层次综合代码进行可配置的评估；（3）HLSTester [35] 通过检测 C/C++ 源代码与综合硬件之间的功能不匹配，进一步增强了评估能力。通过结合大语言模型推理、自适应模糊测试和引导式输入生成，该方法提高了测试覆盖率和错误检测效率。

现有工作在范围上仍较为有限，主要评估语法或功能正确性，却忽略了功耗、性能和面积等硬件关键指标及其设计空间探索。这一空白促成了 Bench4HLS 的开发。Bench4HLS 不仅关注正确性，还纳入结果质量指标和多工具评估，为基于大语言模型的高层次综合提供了更严格且更具实践意义的基准测试。与现有框架的详细比较见表 I。

[2]论文录用后将发布基准测试模型和脚本。

表 I： 基于大语言模型的高层次综合评估框架比较。

维度	Gai 等人 [32]	HLS-Eval [22]	Bench4HLS（本文提出）
基准测试数量	52	94	170
基准测试 Size（LoC*/基准测试）	小型到中等规模 (56)	小型到中等规模 (43)	小型到大规模 (88)
仿真	⚠ HLS 前	✓ HLS 前及 HLS 后	✓ HLS 前、HLS 后以及布局布线后
工具（综合）	⚠ 部分支持	⚠ 固定工具 支持 Vitis	✓ 可插拔（Vitis/Catapult）
功耗、性能和面积分析	✗ No	✗ No	✓ 是
DSE	✗ No	✗ No	✓ 基于 YAML
代码生成	✗ No	✓ 是	⚠ 未优化+
开源	✗ No	✓ 是	✓ 是
并行测试	✗ No	✓ 是	✓ 是

*：基准代码行数：每个基准测试中的平均代码行数。+：能够生成代码，但优化并非 Bench4HLS 的主要目标。

III. 提出的框架：Bench4HLS

为系统地评估基于大语言模型的高层次综合代码生成，Bench4HLS 包含四个主要阶段：(i) 数据集构建，(ii) 通过大语言模型生成代码，(iii) 使用 Vitis HLS 进行功能验证与综合，以及 (iv) 性能表征与比较分析，如图1所示。在这些阶段中，我们构建了一条端到端流程，不仅能够评估语法和语义正确性，还能够评估对实际部署至关重要的硬件层面指标。

A. 数据集：指令、设计和测试平台

第一阶段是构建一个精选数据集，其中包含 170 个不同规模的设计，每个设计都以三元组表示：一条自然语言指令、一个可综合的 HLS（C/C++）实现，以及一个对应的 HLL（C/C++）测试平台。这些三元组的构建方式如下：

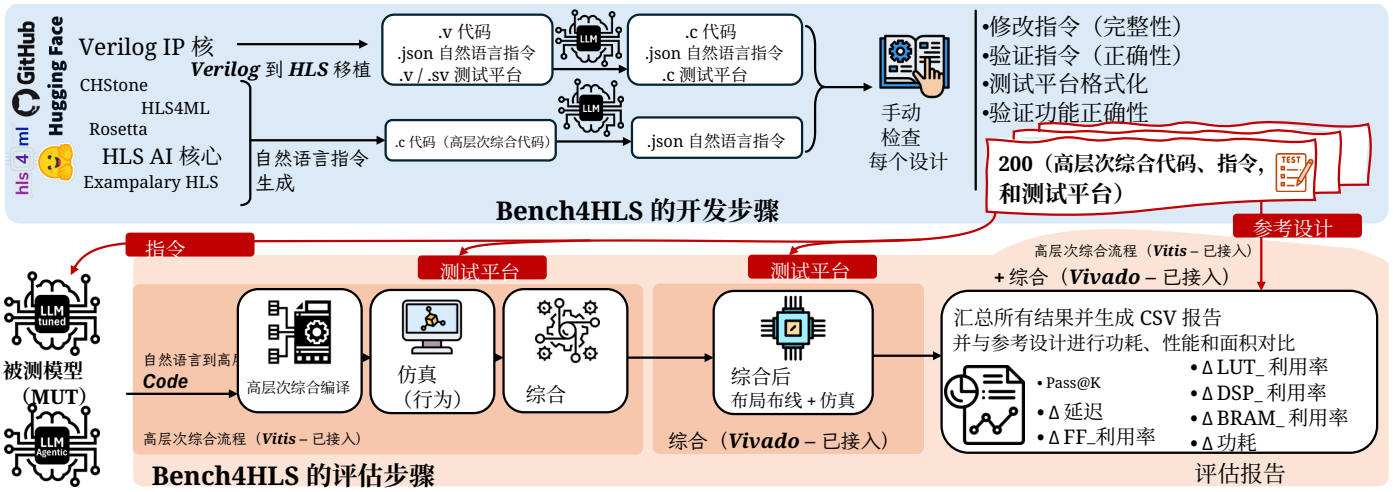


图 1： Bench4HLS 概述：评估方法的完整工作流程

(i) 参考设计集：我们整理了一个全面、多尺度的异构测试用例数据集。它涵盖小规模 Verilog 数据集（例如 VerilogEval [18]）、Vitis-HLS-Introductory-Examples [36]，和教科书参考资料中的高层次综合代码 [37]，[38]，，也包括来自 CHStone [39]，、HLS4ML [40], 和 Rosetta [41] 等成熟代码库的中大型加速器与算术电路。这种多样性使每个测试用例平均包含 88 行代码（LoC）（见表 I），由此构建了迄今为止规模最大、最具代表性的一类数据集。

(ii) 格式整理与指令准备： 对于选定的 Verilog 测试用例，我们使用 GPT-5 模型将每个设计移植为高层次综合 C++，随后通过人工验证确保其正确性（示例见提示 1）。总共生成了 170 份高层次综合代码，随后 GPT-5 为每个设计生成了自然语言指令。为进行全面评估，我们还构建并人工验证了相应的测试平台，以确保数据集的可靠性和完整性。

提示 1：将 Verilog 移植到高层次综合（C/C++）。

“
你是高层次综合 C++ 和 Verilog 代码生成方面的专家。我将提供一段针对 Verilog 硬件设计的提示、Verilog 参考代码以及其测试平台。

你的任务是为高层次综合 C++ 生成参考设计和测试平台。你生成的测试平台必须体现参考代码中的相同逻辑。

所有生成的代码都将在高层次综合工具中运行。编写测试平台时，请考虑所有可能的场景。测试平台必须能够自检。编写代码时不要包含“RefModule”。

测试平台将检测设计文件中的错误，但不得显示编译或仿真错误。它还必须始终返回 0。请在设计代码中使用所有必要且适用的编译指示，以实现更好的优化。

以下是 Verilog 的三个条目：

{提示}{参考代码}{
测试平台}””

(iii) 人工数据集验证：每条指令都经过多轮大语言模型提示（自我改进 [42]）验证，以确认其稳健性以及对设计意图的忠实度。测试平台旨在体现功能覆盖率，并包含源自相应设计的嵌入式逻辑。每组三元项的正确性均通过人工检查和仿真验证。尽管经过人工处理，我们精选数据集中的全部 170 条记录都针对指令到HLS生成任务的功能和PPA评估进行了定制。

B. 基于大语言模型的*HLS*代码生成

尽管高效代码生成并非 Bench4HLS 的首要目标，但该框架采用灵活的可插拔 API，可将大语言模型集成到其包含 200 个任务的数据集中，以生成高层次综合代码。目前，支持的自然语言指令充当任务提示，并通过 API 传递给集成的大语言模型，以生成相应的 HLS代码实现。由于目前尚无专用于高层次综合的开源大语言模型，我们选用广泛使用的开源和闭源代码模型（如 GPT-5、QwenCoder 和 Llama）进行案例研究。

使用这种可插拔模型，展示了 Bench4HLS 如何应用于真实场景：在这些场景中，被测模型（MUT）通过框架的评估流水线，接受系统化评估，以检验其功能正确性、可综合性以及 PPA效率。

(i) 接入模型（应用程序接口）：在测试阶段，我们集成并评估了多个模型，包括 GPT-5、QwenCoder 和 Llama 等专有模型与开源模型。Python API（vLLM 和 simonw/llm Python 库）针对基于指令的输入和面向代码的响应进行了全面标准化，屏蔽了不同供应商之间的差异（尤其是 HLS 工具在语法方面的差异）。

(ii) 提示与解析：使用提示 2 中所述的提示所生成的指令查询每个模型，该提示要求模型针对 HLS C++模块生成高层次的系统设计说明。随后对输出进行后处理，删除评论、标题或测试脚手架，确保保留的内容仅聚焦于模块的功能、数据流、流水线化、内存行为以及

控制逻辑。自定义脚本进一步统一了格式和术语，以确保结果的一致性。

提示 2：为高层次综合测试用例生成指令（规范）。

“

你是一位擅长编写和讲解高层次综合 C++ 硬件设计的专家。请从高层次的系统设计角度描述该高层次综合设计的功能。

重点说明模块执行的计算、数据如何在模块中流动，以及控制逻辑如何在不同执行阶段中运行。结合流水线化、存储器访问和综合约束，解释其功能行为。

避免变量名称或逐行映射等不必要的底层细节。应以精准、实用的语言说明其用途、算法步骤、输入输出行为，以及任何状态或时序依赖，为实现或验证该设计提供指导。

如有相关内容，请包含时序模型、流水线阶段和基本握手机制，但应使说明聚焦于硬件特性。

{参考高层次综合 C++ 代码}”

(ii) 数据集组织：提取出的设计采用系统化的命名方案进行存储，从而便于在后续编译与评估步骤中无缝进行批处理。此阶段生成一组结构化的大语言模型生成的 HLS 代码，作为后续功能与性能评估的主要对象。

C. 通过 *Vitis HLS* 进行自动化验证与综合

Bench4HLS 框架提供了一个用于评估高层次综合的全自动流程。该框架采用可插拔设计，能够无缝集成不同的 HLS 和实现工具链。在当前配置中，前端连接至 Vitis HLS，以进行编译、仿真和综合；后端则连接至 Vivado，以执行 RTL 后的布局布线，从而开展详细的功耗、性能和面积分析。这种集成全面呈现了设计质量，结合了周期精确语义验证与经综合验证的正确性。值得注意的是，该框架采用了适用于批处理模式评估的架构，支持多个设计并行执行。这一能力尤为关键，因为对于复杂的基准测试而言，HLS 流程可能需要大量计算资源。评估分为三个主要阶段：*(1)* 编译：对设计进行编译，以检测语法错误。编译失败的设计将被记录为编译失败。*(2)* 综合前仿真：使用相应的测试平台对已编译的设计进行仿真，以验证生成代码的功能行为是否符合预期输出。*(3)* 综合与综合后仿真：对通过仿真的设计进行综合，以获得 RTL 级实现。随后进行 RTL 综合，以获取资源利用率指标（例如查找表、触发器、块 RAM），然后进行综合后仿真，以执行周期精确的功能验证。

所有步骤均通过由批处理文件和 TCL 脚本组成的自动化脚本流水线执行。这确保了基准测试流程在数百种设计和模型上的可复现性与可扩展性。本阶段充当筛选器，用于识别哪些基于大语言模型生成的代码不仅

算法 1 Bench4HLS 中结合设计空间探索的自动化评估。

要求： 高层次综合设计集 $\mathcal{D}$、测试平台集 $\mathcal{T}$、Pass@K、探索策略 π、目标现场可编程门阵列。
确保： 结构化报告 $\mathcal{R}$，包含每个解的状态以及功耗、性能和面积指标。

初始化 $\mathcal{R} \leftarrow \emptyset$
将 $\mathcal{D}$ 划分为 K 个批次：$\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ ▷Top-K / Pass@K 批处理
对于 $r \leftarrow 1$ **到** K，**执行**
  对于 $d_i \in \mathcal{D}_r$，**执行**
    获取测试平台 $t_i \in \mathcal{T}$ 和 YAML 设计空间探索规范 Y_i
    将 Y_i 扩展为解集 $\Omega_i \leftarrow \text{EXPAND}(Y_i, \pi)$
    初始化按设计结果 $\mathcal{R}_i \leftarrow \emptyset$
    对于 每个解 $s \in \Omega_i$ **执行**
      打开 Plugged HLS 项目 d_i 并应用来自 s的指令
      CSIM_DESIGN →编译状态$\sigma^{\text{com}}_{i,s}$。
      将 $(d_i, s, \sigma^{\text{com}}_{i,s})$ 附加到 $\mathcal{R}_i$
      如果 $\sigma^{\text{com}}_{i,s}$ =通过 **则**
        使用 d_i 在 Plugged HLS 中 t_i 仿真 → 状态 $\sigma^{\text{sim}}_{i,s}$。
        将 $(d_i, s, \sigma^{\text{sim}}_{i,s})$ 追加到 $\mathcal{R}_i$
        如果 $\sigma^{\text{sim}}_{i,s}$ =通过 **则**
          CSYNTH 设计 → 状态 $\sigma^{\text{syn}}_{i,s}$，延迟 $L_{i,s}$，面积向量 $A_{i,s}$
          将 $(d_i, s, \sigma^{\text{syn}}_{i,s}, L_{i,s}, A_{i,s})$ 追加到 $\mathcal{R}_i$
          导出寄存器传输级 → 综合工具；从 s 设置策略
          解析现场可编程门阵列时序/功耗 → 最差负时序裕量$_{i,s}$, $F_{\max i,s} P_{i,s}$
          将（d_i, s, 最差负时序裕量$_{i,s}$, $F_{\max i,s}$, $P_{i,s}$）追加到 $\mathcal{R}_i$
      将（d_i, $\mathcal{R}_i$, Π_i）追加到 $\mathcal{R}$
使用按设计聚合的 $\{\sigma^{\text{com}}_{i,s}, \sigma^{\text{sim}}_{i,s}, \sigma^{\text{syn}}_{i,s}\}$ 计算各阶段的 Pass@K
将 Pass@K 汇总行和每个设计的最佳结果追加到 $\mathcal{R}$
返回 $\mathcal{R}$

语法有效，而且功能正确，并且在功耗、性能和面积方面达到可接受水平，适用于硬件实现。

D. 基于设计空间探索的表征与比较分析

Bench4HLS 不仅根据功能正确性评估大语言模型生成的 HLS 设计，还通过详细的功耗、性能和面积指标 [3] 进行评估。该框架主要集成设计空间探索，以评估所插入的编译器指令和实现策略如何影响这些结果。Bench4HLS 评估的主要指标包括：(i) 延迟（以纳秒为单位）；(ii) 硬件资源利用率（查找表、触发器、数字信号处理器、块 RAM）；(iii) 对综合后的寄存器传输级描述进行功耗分析。此外，Bench4HLS 还会在编译、仿真和综合阶段计算 Pass@K指标。

此外，每个设计都可以关联一个声明式 YAML 规范，其中列出了待进一步探索的设计空间。Bench4HLS 通过实例化多个解（串行或并行）、在 Plugged HLS（例如 Vitis HLS）中应用指令、收集综合报告，并比较这些报告来执行设计空间探索。该自动化流程的概览如算法 1 所示。根据设计规范，还将包含各种指令和编译指示，例如流水线化、可调间隔、数据流、循环展开等。如清单 1 所示，Bench4HLS 在此过程中使用了详细的 YAML 规范：

```
时钟_周期_纳秒：  [3.3，5.0]启用_流水线：  [真，假]流水线_间
```

清单 1： Bench 中用于设计空间探索的 YAML 配置示例 4HLS

[3]所有与功耗、性能和面积相关的结果，均使用归一化功耗、性能和面积指标与相应的参考实现（由 Bench4HLS 提供）进行比较。

表 II： 选定大语言模型的编译、仿真和综合状态 Pass@K 评估

评估模型	编译状态			仿真状态			综合状态		
	Pass@1	通过@5	通过@10	通过@1	通过@5	通过@10	通过@1	pass@5	pass@10
Qwen 2.5 Coder 14B	47.06%	58.82%	65.88%	18.82%	25.88%	30.59%	17.65%	24.71%	28.82%
Qwen 2.5 Coder 32B	82.35%	90.59%	92.35%	28.24%	36.06%	39.41%	24.12%	34.71%	38.24%
Llama 3.3 70B	77.65%	95.88%	97.65%	45.29%	58.24%	64.12%	44.12%	57.65%	63.53%
GPT-5	85.29%	97.65%	97.65%	52.35%	65.88%	72.35%	50.00%	65.88%	71.76%

通过以结构化、机器可读的 YAML 格式公开设计空间探索空间，Bench4HLS 使设计空间探索可复现且易于扩展。这一设计使 Bench4HLS 不仅是一个基准测试平台，也是一个用于系统研究大语言模型生成的 HLS 设计的实用框架。

提示 3：用于测试模型的指令。

```
“
你的角色是担任高层次综合 C++ 代码开发专家。你必须通过系统且审慎的思考过程，全面探究每个问题——通过分析、总结、探索、重新评估、反思、回溯和迭代等循环，制定经过深思熟虑的解。


根据所提供的指令，你需要生成精确、优化且准确的可综合高层次综合 C++ 代码，并满足以下要求。


1. 仅输出可综合的 C++ 代码（不附加其他文本或说明）。2. 包含必要的头文件（例如 ap int.h）。3. 顶层函数签名必须与指令中给出的一致。4. 插入合适的 HLS 编译指示。5. 自包含：除标准头文件外，不依赖外部库或文件。



### 指令：{}###
输入：{}### 响应：
{}”
```

IV. 实验设置

为展示 Bench4HLS 的全面性，我们评估了四种大语言模型的高层次综合代码生成能力：Qwen2.5-Coder（14B 和 32B）、LlaMA 3.3 70B 和 GPT-5。为实现高效推理，Qwen 和 LlaMA 模型均在本地以 4 位量化方式部署，而 GPT-5 则通过 OpenAI 的应用程序接口调用。对于 170 个基准任务中的每一个，我们都使用提示 3 中提及的结构化提示，明确指示模型生成可综合、带有 pragma 注释的 HLS C++代码（并结合用于设计空间探索的 YAML）。虽然流程支持插件式扩展，但在本文中，生成后和综合阶段的所有设计均使用 Vitis HLS 2024.1 进行编译、仿真和综合，并使用 Vivado 2024.1 进行功耗估算。所有设计均面向 Xilinx Artix-7xc7a200tffv1156-1 现场可编程门阵列，以确保比较一致。Bench4HLS 的整个工作流程通过自定义的可插拔脚本和 TCL 驱动程序实现自动化，从而支持对多种模型和 Pass@K 配置进行可扩展评估。

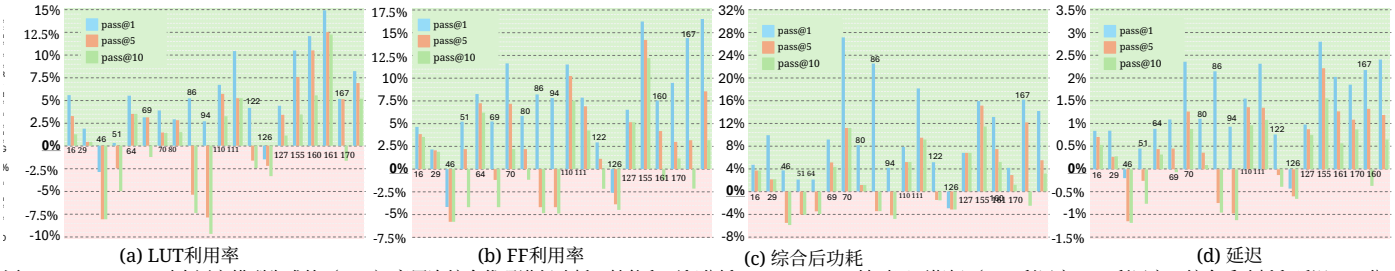
V. 实验结果

我们使用四个最先进的大语言模型对 Bench4HLS 进行了评估：**Qwen2.5-Coder 14B**、**Qwen2.5-Coder 32B**、**LlaMA 3.3 70B**和**GPT-5**。目标是评估 Bench4HLS 评估给定模型将自然语言描述转换为功能正确且可综合的高层次综合设计的能力。Bench4HLS 提供了一个完全自动化的设计生成、仿真、综合以及功耗、性能和面积分析流程，从而实现可复现且全面的基准测试。评估流程由四阶段流水线组成：**(i) 语法有效性**：通过 CSIM DESIGN 验证编译成功；**(ii) 语义正确性**：使用测试平台驱动仿真进行功能验证；**(iii) 硬件可行性**：通过在 Vivado 中进行寄存器传输级生成来验证可综合性；**(iv) 设计空间探索**：使用 YAML 规范（见清单 1）自动探索高层次综合指令和 Vivado策略，以获得最优的功耗、性能和面积。对于每个阶段，Bench4HLS 都会计算 Pass@K指标（$K \in \{1, 5, 10\}$），用于衡量至少有一个排名前-K 的生成候选项通过验证的可能性。表 II 总结了编译、仿真和综合阶段的结果。Bench4HLS 允许我们基于这一四阶段流程集成任何新的 MUT 进行评估。

Bench4HLS 提供了如表 II 所示的模型间比较。在这个示例中，**GPT-5** 表现最为出色，Pass@10 的编译成功率达到 97.65%，仿真和综合的成功率分别为 72.35% 和 71.76%。**LlaMA 3.3 70B** 紧随其后，而**Qwen2.5**系列则体现了规模扩大的影响：从 14B 增至 32B，语法和语义保真度均显著提升。这些结果凸显了面向高层次综合的代码生成对模型规模和能力的敏感性。

为进一步体现 Bench4HLS 的能力，我们在此评估了自动化的基于 YAML 的高层次综合设计空间探索。如图 3 所示，对于给定的四个模型，Bench4HLS 能够使用可配置参数对基准测试进行仿真和综合（如清单 1 所示）。图 3 展示了应用设计空间探索后的改进效果（在本例中，至少一个功耗、性能和面积指标的改善幅度超过 20%）。如图所示，对于 GPT-5，在 $> 20\%$ 的基准测试中，设计空间探索使至少一个功耗、性能和面积指标提升了 $> 20\%$，而 Qwen2.5-14B 的这一比例仅为 $\sim 10\%$。这表明，指令级搜索对于充分发挥大语言模型生成的设计的潜力至关重要。借助自动化设计空间探索，我们能够在各种配置下全面测试 MUT 的有效性。

此外，除高层次综合外，Bench4HLS 还会在指定的（可插拔）综合工具（此处为 Vivado）中，使用可配置的目标现场可编程门阵列（此处为*Xilinx Artix-7 xc7a200tffv1156-1*）集成 RTL 后端实现。相较于参考设计，可提取四项主要指标：延迟（相较参考设计的改进百分比）、FF利用率（相较参考设计的改进百分比）、LUT利用率（相较参考设计的改进百分比）、



(a) LUT利用率　(b) FF利用率　(c) 综合后功耗　(d) 延迟

图2： Bench4HLS 对大语言模型生成的（GPT）高层次综合代码进行功耗、性能和面积分析。Bench4HLS 针对四项指标（LUT利用率、FF利用率、综合后功耗和延迟——分别对应 pass@1、pass@5 和 pass@10 次生成），报告 MUT 生成的设计与 Bench4HLS 提供的参考设计之间的百分比差异，数值表示（生成设计 – 参考设计）%。正值表示资源使用量或延迟高于参考设计。

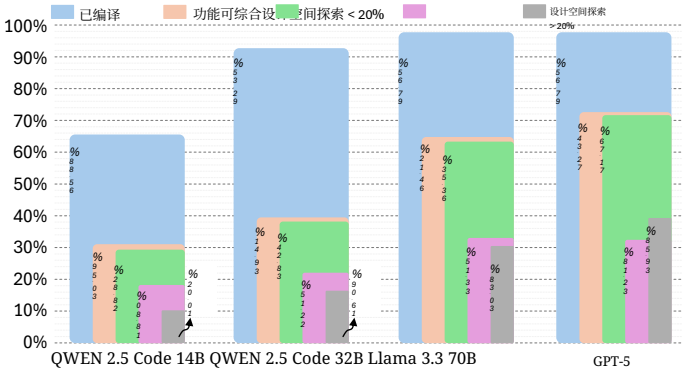
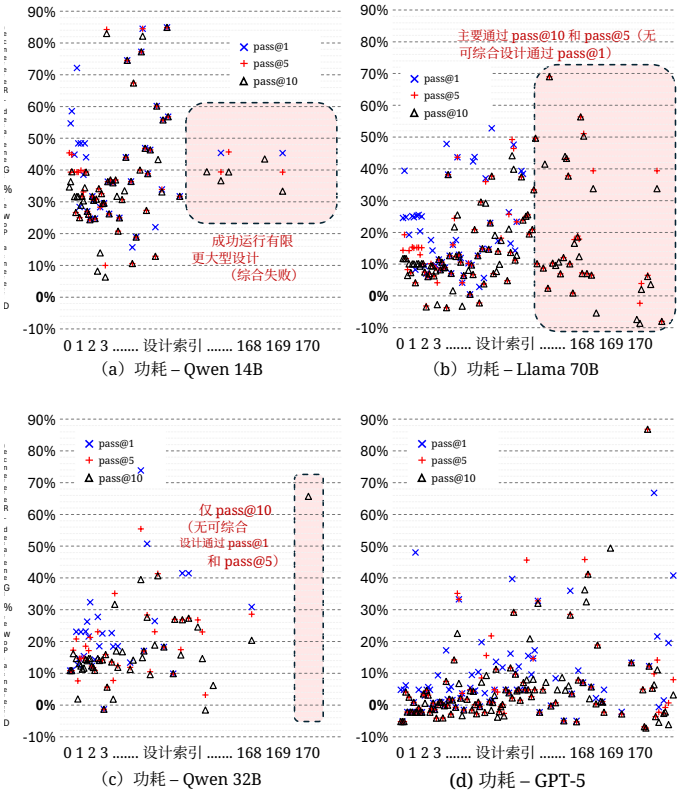


图3： Bench4HLS对高层次综合代码生成的Pass@10评估（DSE指至少一项功耗、性能和面积指标改善不足或超过20%）。

以及估算功耗（相较参考设计的改进百分比）。图2展示了 GPT-5 生成的设计在这些指标上的表现。Bench4HLS 的一大优势在于其精心筛选的参考设计集，可作为高质量的比较基准。如图2所示，在 PPA效率方面，参考实现始终优于表现最强的模型（GPT-5）。对于较小的基准测试（问题 #1–#100），大语言模型生成的解决方案所达到的指标接近参考基线；但对于更大、更复杂的基准测试（#150+），大多数模型都难以生成可综合设计。在部分设计中，最终的 PPA指标明显差于参考设计，凸显了这些精心筛选的设计的效率优势。对于 Qwen2.5-14B/32B 和 LlaMA 3.3 70B 等较小模型，这一趋势尤为明显：它们更频繁地失败，且与基线的差距更大。这些结果表明，Bench4HLS 中的参考设计不仅仅是功能基线，更是**经过充分优化、注重效率的实现**，为评估自动化方法提供了有意义的上限。

图4更清楚地揭示了这一可扩展性问题，其中展示了所有基准测试中的功耗分布。如图所示，较小的模型（例如 Qwen2.5-14B）无法为许多规模较大的问题生成可综合设计（设计索引 #120 之后结果稀疏），而 GPT-5 则能为更多类型的情况生成有效的寄存器传输级代码。成功案例的广度体现了 Bench4HLS 的自动化流水线能够持续进行 PPA提取和综合验证。这些实验表明，Bench4HLS 为评估大语言模型生成的 HLS 设计提供了有效且可扩展的框架。通过将 Pass@K正确性评估与 YAML-



（a）功耗 – Qwen 14B　（b）功耗 – Llama 70B

（c）功耗 – Qwen 32B　(d) 功耗 – GPT-5

图 4： Bench4HLS 测试的 MUT 中的功耗基准测试。

以 DSE 和 RTL 后功耗、性能和面积分析为驱动，该框架能够对不同模型进行全面比较。

VI. 结论

本文提出 Bench4HLS，这是一个全面的基准测试与评估框架，专用于评估大语言模型在高层次综合代码生成方面的能力。Bench4HLS 将精心整理的 170 个优化参考设计，与自动化的四阶段评估流程相结合；该流程采用可插拔的行业标准工具，执行编译、仿真、综合和基于 YAML 的设计空间探索，从而能够系统地改变高层次综合指令和后端策略，严谨地衡量正确性、可行性以及 PPA效率。为评估 Bench4HLS 的能力，我们对四种先进的大语言模型进行了评估，即 GPT-5、Llama 3.3 70B 和 Qwen 2.5 14B/32B。Bench4HLS 提供了一个可扩展、即插即用的测试平台，为系统研究大语言模型在高层次综合设计中的应用奠定了基础。

参考文献

[1] P. Coussy、D. D. Gajski、M. Meredith 和 A. Takach，《高层次综合简介》，IEEE《计算机设计与测试》，第 26 卷，第 4 期，第 8–17 页，2009 年。[2] S. Lahti、P. Sj¨ovall、J. Vanne 和 T. D. H¨am¨al¨ainen，《进展如何？高层次综合现状研究》，IEEE《集成电路与系统计算机辅助设计汇刊》第 38 卷，第 5 期，第 898–911 页，2018 年。[3] D. D. Gajski、N. D. Dutt、A. C. Wu 和 S. Y. Lin，《高层次综合：芯片与系统设计导论》。施普林格科学与商业媒体，2012 年。[4] S. Liu、F. C. Lau 和 B. C. Schafer，《通过基于预测模型的高层次综合设计空间探索加速 FPGA 原型设计》，载于《第56届年度设计自动化会议（2019）论文集》，2019 年，第 1–6 页。[5] J. J. Zhang 等，《通过端到端综合实现自动化、敏捷的 AI/ML 加速器设计》，载于《2021 年 IEEE 第32届专用系统、架构与处理器国际会议（ASAP）》。IEEE，2021 年，第 218–225 页。[6] J. Cong、J. Lau、G. Liu、S. Neuendorffer、P. Pan、K. Vissers 和 Z. Zhang，《当今的 FPGA 高层次综合：成果、挑战与机遇》，ACM《可重构技术与系统汇刊》（TRETS），第 15 卷，第 4 期，第 1–42 页，2022 年。[7] A. Cortes、I. Velez 和 A. Irizar，《使用 Vivado HLS 对 Zynq 片上系统进行高层次综合：图像处理案例研究》，载于《2016 年电路与集成系统设计会议（DCIS）》。IEEE，2016 年，第 1–6 页。[8] S. Shi、N. Pundir、H. M. Kamali、M. Tehranipoor 和 F. Farahmandi，《SecHLS：在高层次综合中实现安全意识》，载于《第28届亚洲及南太平洋设计自动化会议论文集》，2023 年，第 585–590 页。[9] S. Swaroopa、R. Mukherjee、A. Debnath 和 R. S. Chakraborty，《通过高层次综合评估大语言模型自动生成寄存器传输级逻辑的能力》，arXiv 预印本 arXiv:2408.02793，2024 年。[10] C. Xiong、C. Liu、H. Li 和 X. Li，《HLSPilot：基于大语言模型的高层次综合》，载于《第43届 IEEE/ACM 计算机辅助设计国际会议论文集》，2024 年，第 1–9 页。[11] Y. Liao、T. Adegbija 和 R. Lysecky，《大语言模型是否擅长高层次综合？》，载于《第43届 IEEE/ACM 计算机辅助设计国际会议论文集》，2024 年，第 1–8 页。[12] Kangwei Xu、Grace Li Zhang、Xunzhao Yin、Cheng Zhuo、Ulf Schlichtmann 和 Bing Li，《通过大语言模型自动修复用于高层次综合的 C/C++ 程序》，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》，2024 年，第 1–9 页。[13] S. A. Sheikholeslam 和 A. Ivanov，《SynthAI：用于自动化模块化高层次综合设计生成的多智能体生成式人工智能框架》，arXiv 预印本 arXiv:2405.16072，2024 年。[14] H. Xu、H. Hu 和 S. Huang，《利用检索增强型大语言模型优化高层次综合设计》，载于《2024 年 IEEE 大语言模型辅助设计研讨会（LAD）》。IEEE，2024 年，第 1–5 页。[15] L. Collini、S. Garg 和 R. Karri，《C2HLSC：大语言模型能否弥合软硬件设计鸿沟？》，载于《2024 年 IEEE 大语言模型辅助设计研讨会（LAD）》。IEEE，2024 年，第 1–12 页。[16] A. E. Oztas 和 M. Jelodari，《Agentic-HLS：基于智能体推理、使用大语言模型的高层次综合系统（2024 年面向电子设计自动化的人工智能研讨会）》，arXiv 预印本 arXiv:2412.01604，2024 年。[17] N. Mashnoor、M. Akyash、H. Kamali 和 K. Azar，《TimelyHLS：基于大语言模型、具备时序感知和架构专用特性的 FPGA 高层次综合优化》，载于《2025 年 IEEE 全层智能系统国际会议（COINS）》。IEEE，2025 年，第 1–6 页。[18] M. Liu 等，《VerilogEval：评估大语言模型生成 Verilog 代码的能力》，载于《2023 年 IEEE/ACM 计算机辅助设计国际会议（ICCAD）》。IEEE，2023 年，第 1–8 页。[19] Y. Lu、S. Liu、Q. Zhang 和 Z. Xie，《RTLLM：用于评估大语言模型生成寄存器传输级设计的开源基准测试》，载于《2024 年第29届亚洲及南太平洋设计自动化会议（ASP-DAC）》。IEEE，2024 年，第 722–727 页。[20] N. Pinckney、C. Deng、C.-T. Ho、Y.-D. Tsai、M. Liu、W. Zhou、B. Khailany 和 H. Ren，《全面的 Verilog 设计问题：用于评估大语言模型和智能体在寄存器传输级设计与验证方面能力的新一代基准数据集》，arXiv 预印本 arXiv:2506.14074，2025 年。[21] B. C. Schafer 和 Z. Wang，《高层次综合设计空间探索：过去、现在与未来》，IEEE《集成电路与系统计算机辅助设计汇刊》，第39卷，第10期，第2628–2639页，2019年。[22] S. Abi-Karam 和 C. Hao，“Hls-eval：用于评估大语言模型在高层次综合设计任务中表现的基准测试与框架”，arXiv 预印本 arXiv:2504.12268，2025年。[23] J. Cong 等人，“面向现场可编程门阵列的高层次综合：从原型设计到部署”，IEEE《集成电路与系统计算机辅助设计汇刊》，第30卷，第4期，第473–491页，2011年。[24] J. Pan、G. Zhou、C.-C. Chang、I. Jacobson、J. Hu 和 Y. Chen，“电子设计自动化领域大语言模型研究综述”，ACM电子系统设计自动化汇刊，第30卷，第3期，第1–21页，2025年。[25] S. Liu、W. Fang、Y. Lu、J. Wang、Q. Zhang、H. Zhang 和 Z. Xie，“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》，2024年。[26] M. Akyash、K. Azar 和 H. Kamali，“Rtl++：用于RTL代码生成的图增强大语言模型”，载于《2025年IEEE大语言模型辅助设计国际会议（ICLAD）》。IEEE，2025年，第44–20页。[27] Y. Zhao、H. Zhang、H. Huang、Z. Yu 和 J. Zhao，“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）》。IEEE，2025年，第1–7页。[28] H. Ping 等人，“Hdlcore：用于缓解大语言模型生成硬件描述语言中的幻觉的免训练框架”，arXiv 预印本 arXiv:2503 16528，2025年。.[29] M. Akyash、K. Azar 和 H. Kamali，“Decortl：用于大语言模型RTL代码生成的运行时解码框架”，arXiv 预印本 arXiv:2507.02226，2025年。[30] N. Pinckney、C. Batten、M. Liu、H. Ren 和 B. Khailany，“重新审视 VerilogEval：大语言模型在硬件代码生成方面一年来的进展”，《ACM电子系统设计自动化汇刊》，2月。2025 [ ] 10 1145 3718088 在线。可获取：https://doi.org/ 。/ [31] N. Prakriya、Z. Ding、Y. Sun 和 J. Cong，“Lift：通过图神经网络监督微调实现基于大语言模型的高层次综合编译指示插入”，arXiv 预印本 arXiv:2504.21187，2025年。[32] J. Gai 等人，“探索用于自动化高层次综合硬件生成的代码语言模型：基准测试、基础设施与分析”，载于《第30届亚洲及南太平洋设计自动化会议论文集》，2025年，第988–994页。[33] L. Collini、A. Hennessee、R. Karri 和 S. Garg，“推理模型能推理硬件吗？智能体式高级综合视角”，arXiv 预印本 arXiv:2503.12721，2025年。[34] M. Khan、N. Mashnoor、M. Akyash、K. Azar 和 H. Kamali，“Sage-hls：语法感知、抽象语法树引导的大语言模型，用于高层次综合代码生成”，arXiv 预印本 arXiv:2508.03558，2025年。[35] K. Xu、B. Li、G. L. Zhang 和 U. Schlichtmann，“Hlstester：利用大语言模型高效测试高层次综合中的行为差异”，arXiv 预印本 arXiv:2504.14641，2025年。[36] 赛灵思，“Vitis HLS入门示例”，https://github.com/Xilinx/Vitis-HLS-Introductory-Examples，2024年，访问日期：2025年5月18日。[37] M. Fingeroff，《高级综合：蓝皮书》。Xlibris Corporation，2010年。[38] R. Kastner、J. Matai 和 S. Neuendorffer，“面向现场可编程门阵列的并行编程”，arXiv 预印本 arXiv:1805.03648，2018年。[39] Y. Hara、H. Tomiyama、S. Honda 和 H. Takada，“针对实用的基于C语言的高层次综合，提出并定量分析ChStone基准程序套件”，《信息处理学报》，第17卷，第242–254页，2009年。[40] F. Fahim 等人，“HLS4ML：赋能科学计算低功耗机器学习设备的开源协同设计工作流”，arXiv 预印本 arXiv:2103.05579，2021年。[41] Y. Zhou 等人，“Rosetta：面向软件可编程现场可编程门阵列的真实高层次综合基准测试套件”，载于《2018年ACM/SIGDA现场可编程门阵列国际研讨会论文集》，2018年，第269–278页。[42] A. Madaan 等人，“Self-refine：利用自反馈进行迭代优化”，《神经信息处理系统进展》，第36卷，第46 534–46 594页，2023年。