计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(6): 1423−1436, 2026

LEMON: 面向高效硬件实现的大语言模型驱动集成约简与 Verilog HDL 生成

皇甫璟轩　王虹飞
（华中科技大学网络空间安全学院　武汉　430074）
（分布式系统安全湖北省重点实验室（华中科技大学）　武汉　430074）
（湖北省大数据安全工程技术研究中心（华中科技大学）　武汉　430074）
（jingxuan@hust.edu.cn）

LEMON: LLM-Driven Ensemble Reduction and Verilog HDL Generation for Efficient Hardware Implementation

Huangfu Jingxuan and Wang Hongfei
（*School of Cyber Science and Engineering*, *Huazhong University of Science and Technology*, *Wuhan* 430074）
（*Hubei Key Laboratory of Distributed System Security* (*Huazhong University of Science and Technology*), *Wuhan* 430074）
（*Hubei Engineering Research Center on Big Data Security* (*Huazhong University of Science and Technology*), *Wuhan* 430074）

Abstract　Ensemble learning has been widely adopted to enhance the generalization performance of artificial intelligence models. However, the additional memory overhead and computational costs induced by ensemble models limit their deployment in resource-constrained scenarios. To address this challenge, we propose LEMON, a large language model (LLM) driven framework for ensemble reduction and Verilog HDL generation, designed to enable efficient hardware implementation. By leveraging the advanced capabilities of LLMs in semantic understanding, code generation, and complex problem solving, LEMON significantly reduces the memory footprint and computational demands of ensemble models. Extensive evaluations on 20 public datasets demonstrate that while maintaining a compact model size, LEMON achieves comparable or superior prediction accuracy to the original ensemble methods across the majority of test sets. Notably, compared with state-of-the-art techniques, LEMON achieves speedups exceeding 9 times and 338 times in ensemble reduction, showcasing exceptional scalability across diverse scenarios. Furthermore, LEMON integrates an LLM-powered automated hardware deployment pipeline that significantly simplifies the transition from high-level software models to optimized FPGA implementations. Compared with conventional random forest hardware implementations, LEMON reduces power consumption by over 64.7% and hardware resource utilization increased by more than 92.6%, making it highly suitable for edge computing and embedded system applications.

Key words　ensemble reduction; large language models; hardware implementation; logic minimization; combinatorial optimization; machine learning

摘　要　集成学习已被广泛用于提高人工智能模型的泛化性能。然而，集成模型引起的额外内存开销和

收稿日期：2026-03-25；**修回日期**：2026-04-30
基金项目：国家自然科学基金项目（62372198, 62172173）；湖北省自然科学基金项目（2026AFB636）
This work was supported by the National Natural Science Foundation of China (62372198, 62172173) and the Natural Science Foundation of Hubei Province (2026AFB636).
通信作者：王虹飞（hongfei@hust.edu.cn）

计算成本限制了它们在资源受限场景下的应用。为应对这一挑战,提出了 LEMON,一种面向高效硬件实现的大语言模型(large language models, LLMs)驱动的集成约简与 Verilog HDL 生成框架。LEMON 利用 LLM 在文本理解、生成以及解决复杂问题方面的优势,显著降低了集成模型的内存占用和计算需求。在 20 个公共数据集上的广泛评估表明,LEMON 在保持紧凑模型尺寸的同时,在多数测试集上达到甚至超越原始集成方法。值得注意的是,与最先进的技术相比,LEMON 在集成约简方面实现了超过 9 倍和 338 倍的加速,展示了其在各类场景下的卓越可扩展性。此外,LEMON 集成了一种基于 LLM 的模型自动化硬件部署流程,显著简化了从软件模型到高效 FPGA 实现的过渡过程。与传统的随机森林(random forest)实现相比,LEMON 将功耗降低了 64.7% 以上,硬件资源利用率提高 92.6% 以上,使其更加适用于边缘和嵌入式场景下的应用。

关键词 集成约简;大语言模型;硬件实现;逻辑最小化;组合优化;机器学习

中图法分类号 TP391; TP18

DOI: 10.7544/issn1000-1239.202660161 **CSTR:** 32373.14.issn1000-1239.202660161

硬件集成的 AI 系统已在现代应用中无处不在,涵盖了从实时网络流量分类[1]到片上诊断系统[2]的各个领域。随着大语言模型(large language models, LLMs)的兴起,机器学习(machine learning)模型的软硬件协同设计再次成为电子设计自动化(electronic design automation, EDA)领域的焦点。尽管如此,由于严格的资源限制,将这些计算密集型模型部署到物联网(Internet of things, IoT)边缘节点(包括可穿戴设备和嵌入式系统)仍然充满挑战。在此背景下,树模型因其较低的计算资源需求而成为一种更可行的替代方案。此外,树模型的结构特性使其能够充分利用 FPGA(field programmable gate array)的并行计算能力,从而在资源受限的环境中实现高效部署。

虽然单棵树的性能可能无法达到预期,但随机森林(random forest, RF)等集成学习(ensemble learning)方法可以有效解决这一问题[3]。集成学习通过结合 2 个或更多的机器学习模型,实现比单独使用单个模型更好的性能。采用集成学习方法进行预测,通常能够有效降低泛化误差。集成学习方法大致分为提升法(boosting)、装袋法(bagging)和堆叠法(stacking)3 个类别[4],它们通过加权或元学习(meta-learning)来组合多个模型。然而,与单个模型相比,集成模型会消耗更多的计算资源,这就导致了计算资源需求增加和存储成本升高等挑战。为了克服这些挑战,研究人员提出了多种集成约简(ensemble reduction)方法以减轻由集成模型引起的计算和存储成本增加[5-8]。

集成约简本质上是一个组合优化(combinatorial optimization)问题。其主要目标是在集成模型内部识别最有效的模型子集,同时提高集成系统的预测准确性和计算效率。现有的集成约简方法能够生成多样化的模型子集。传统的约简技术大致可分为 3 类:基于排序(ranking-based)的方法、基于聚类(clustering-based)的方法和基于优化(optimization-based)的方法[9]。然而,这些传统方法往往存在固有的局限性,如性能次优或适用性受限,其主要原因是识别最优分类器子集是一个具有高计算复杂度的 NP 完全问题。为了克服这些挑战,CANOPY 算法[5]将集成约简问题转化为逻辑最小化问题,为提高模型性能提供了新的视角。

近年来,LLM 凭借其在文本分析与生成领域的卓越能力,不仅在传统自然语言处理任务中表现优异,更在解决组合优化问题方面取得了突破性进展[10]。这一趋势为集成约简技术的发展开辟了全新路径,凸显了 LLM 在该领域尚未被充分挖掘的巨大潜力。然而,目前针对该方向的研究仍显匮乏。

基于这一背景,本文提出了 LEMON,一种 LLM 驱动的集成约简软硬件协同设计框架,旨在为集成模型的约简和部署提供一种更高效、更简单的解决方案。LEMON 利用 LLM 实现集成约简以及模型硬件自动化部署。本文的主要贡献如下:

1)提出了一种基于 LLM 的集成约简框架,解决了传统集成约简方法性能不足和计算效率低下的问题。该框架基于逻辑最小化范式,通过聚焦于与集成模型相关的真值表,并结合少样本提示(few-shot prompts)、迭代生成机制和结果评分策略,显著提高了约简过程的效率和最终模型的预测准确性。

2)设计了一种基于多级逻辑最小化的增强方法,用于进一步优化约简后集成模型的性能。在基于 LLM 约简的基础上,该方法采用多级逻辑最小化生成逻辑表达式以替代投票机制,从而提升模型性能。

3）提出了一种由 LLM 驱动的针对树模型的高级语言（high-level language, HLL）到硬件描述语言（hardware description language, HDL）的转换范式。利用 LLM 强大的文本理解和生成能力，该方法直接基于模型结构描述，将树的节点映射到芯片上的逻辑门和导线，进而显著降低树模型硬件实现的复杂性。

1 相关工作

1.1 集成约简

集成约简的主要目标是通过选择性修剪，在提高计算效率的同时增强预测准确性，从而优化集成系统。这一过程本质上是一个极具挑战性的组合优化问题。经过多年的研究与发展，已经有多种集成约简方法可以克服传统集成学习的局限性。这些方法大致可分为 3 类：基于排序的方法、基于聚类的方法和基于优化的方法[9]。基于排序的约简方法对集成模型中的基学习器（base learners）进行排序，并根据特定的评估函数选择表现最好的学习器，例如方向排序（orientation ordering）[11]、基于间隔和多样性的排序（margin and diversity based ordering）[12] 以及 EPIC[13]。此外，Ibragimov 等人[8] 提出一种自适应区域早停（early stopping）框架，实现了梯度提升模型（gradient boosting model, GBM）的约简。尽管这些方法能有效实现集成约简的目标，但它们也存在一些明显的缺点，如模型多样性有限、计算成本高以及对评估函数的强依赖性。另一方面，基于聚类的方法使用聚类算法（如 K-means[14] 和一致性聚类[15]）对基学习器进行分组。然而，这些方法的有效性高度依赖于聚类模型的质量，并且对参数设置非常敏感。

基于优化的方法将集成约简问题转化为组合优化问题，旨在识别能够最大化集成泛化性能的最优子集。由于穷举搜索已被证明是一个 NP 完全问题[16]，研究人员采用了随机搜索[15]、半定规划（semi-definite programming）[17] 和帕累托优化（Pareto optimization）[18] 等技术来降低计算复杂度。尽管如此，这些方法仍面临收敛速度慢以及约简后集成规模依然较大等挑战。

CANOPY[5] 通过将集成约简问题转化为逻辑最小化（logic minimization）问题，成功构建了高效的分类集成模型。与其他方法相比，CANOPY 在保持甚至降低泛化误差的同时，显著降低了模型复杂度。具体而言，CANOPY 首先通过自助抽样（bootstrap sampling）构建初始分类树模型池（model pool），然后收集每棵树对每个样本的预测结果及对应标签以形成真值表。随后，利用逻辑最小化技术分析该真值表并推导出逻辑表达式，其中的每个变量都对应一颗最终要保留的树模型。模型的预测结果通过计算这些逻辑表达式获得。在此基础上，HERALD[7] 提出了一种分层集成约简与学习框架，有效解决了 CANOPY 在处理大规模集成模型时约简速度慢以及最终模型准确率低的问题，并进一步在硬件层面评估了该方法。

尽管这些方法在大多数数据集上表现优异，但逻辑最小化的固有局限性导致当训练样本或集成规模增大时训练时间显著增加。此外，基于二级逻辑最小化（2-level logic minimization）推导出的简化逻辑表达式在求解最终输出时，可能因过度化简而对预测准确率产生不利影响。

1.2 基于 LLM 的优化

随着人工智能的快速发展，如 ChatGPT（chat generative pre-trained Transformer）[19] 和 DeepSeek[20] 这样的 LLM 凭借其卓越的自然语言理解和生成能力，在多个领域引起了广泛关注。这些模型不仅应用于传统的自然语言处理任务，还在 EDA 流程自动化和优化算法设计等新兴领域展现出巨大潜力。目前，大多数研究主要聚焦于 EDA 流程的自动化。例如，Chang 等人[21] 利用 LLM 实现硬件代码自动生成，RTLFixer[22] 借助 LLM 完成语法错误的自动修复，ChatEDA[23] 则通过 LLM 生成脚本以驱动 EDA 流程自动化。上述工作旨在降低硬件开发门槛，压缩设计周期并提升整体工程效率，这与 LEMON 引入 LLM 以打通模型自动化部署链路的初衷高度一致。然而，专门面向机器学习模型构建环节的研究仍相对匮乏。

随着研究的深入，LLM 在解决优化问题方面的潜力逐渐显现[24]。其强大的自然语言处理能力使其能够有效处理复杂的问题描述和约束条件。目前，利用 LLM 解决优化问题主要有 2 种方法[25]：第 1 种是利用 LLM 的表示和生成能力创建新颖的优化算法；第 2 种则是专注于利用 LLM 生成新的解（solutions）。

在第 1 类方法中，LLM 被用于优化现有算法。最初的研究涉及通过单轮对话生成代码，并通过调试步骤优化代码。例如，Pluhacek 等人[26] 利用 LLM 生成了元启发式优化算法（metaheuristic optimization algorithms）。OptiMUS[27] 进一步开发了 LLM 在混合整数线性规划（mixed-integer linear programming, MILP）问题中的应用，实现了优化问题的自动建模和求解。随着研究的进展，迭代生成方法逐渐被用于算法优

化。例如，Lehman 等人[28]提出了一种基于进化算法(evolution algorithm, EA)的迭代优化框架，将 LLM 作为进化算子直接操作代码文本，以系统地改进算法。在此基础上，AEL[29]和 ReEvo[30]通过引入自动化启发式算法设计和反射进化(reflective evolution)机制，进一步完善了该框架。尽管这些方法行之有效，但 LLM 的应用仍主要局限于代码生成，难以直接用于机器学习模型的构建。

在第 2 类方法中，LLM 被视为“黑盒(black box)”，主要作为搜索算子来发现新的解。这种方法的可行性在于，当提供自然语言形式的问题描述和历史轨迹时 LLM 能够在优化任务中逐步完善解[31]。基于此，研究人员提出了 OPRO(optimization by prompting)方法，该方法通过将优化问题描述为自然语言并将其作为元提示(meta-prompts)，利用 LLM 作为优化器。在每次迭代中，LLM 使用之前生成的解作为提示来产生新解，这些新解经过评估后被纳入下一次迭代的提示词(prompts)中。

在此基础上，LMEA[32]利用 LLM 执行交叉和变异操作，并在每一代构建新的提示词以指导 LLM 选择父代解。EvoLLM[33]则专注于提示词设计，将 LLM 转化为用于黑盒优化的进化策略。Jiang 等人[34]提出了一种基于 LLM 的框架来解决组合优化问题，该框架整合了网络拓扑和领域知识，实现了设计结构矩阵排序的高效优化。此外，一些研究已将 LLM 应用于多目标优化，例如，利用基于分解的多目标进化算法(multi-objective evolutionary algorithm, MOEA)框架，并通过提示词工程(prompt engineering)和上下文学习(in-context learning)，采用 LLM 作为黑盒搜索算子为每个子问题生成新的后代[35]。这些方法直接利用了 LLM 处理复杂问题描述和约束的能力，将其融入优化问题的求解过程中。与通过编程正式定义问题和实现算子相比，这种方法更直接、更简单，无需额外训练，且具有很强的可扩展性。

1.3 机器学习模型的硬件实现

随着人工智能技术的飞速发展，机器学习模型的硬件实现已成为连接算法理论与实际应用的关键桥梁，相关研究已取得显著进展。然而，面对日益复杂的模型结构，如何在加速计算的同时优化硬件资源利用，仍是当前亟待解决的核心问题。

TreeLUT[36]提出了一种使用梯度提升决策树(gradient boosted decision trees, GBDTs)进行推理加速的高效神经网络替代方案，使用查找表(lookup tables, LUTs)代替传统的乘加运算，并结合高效的量化方案、硬件架构和流水线策略，显著提高了 FPGA 上的硬件利用率和吞吐量。Wang 等人[37]开发了一种基于 FPGA 的通用机器学习加速器，利用指令集加速器执行机器学习任务，并通过自动并行化提升性能。然而上述工作主要聚焦于推理加速，缺乏对模型硬件部署的关注。针对这一挑战，Wang 等人[7]提出了一种分层集成约简与学习框架，解决部署大规模 RF 时的资源限制问题。Alsharari 等人[38]采用感知量化训练(quantization aware training, QAT)方法优化 GBDT，使其仅使用整数和二进制算术即可进行推理。LG-BM2VHDL[39]自动将 LightGBM 模型映射到用 VHDL(very-high-speed integrated circuit hardware description language)描述的硬件电路，促进了梯度提升模型的硬件加速。上述方法虽能有效支持资源受限环境下集成模型的硬件部署，但其实现严重依赖 HLL 到 HDL 的转换流程。

模型部署的核心挑战之一是如何高效地将模型映射到 HDL。例如，Dinh 等人[40]提出的面向 FPGA 部署优化或模型架构定制的方法，通常要求使用者具备扎实的硬件设计专业知识，从而导致工程实施门槛较高。尽管 Wang 等人[7]尝试利用脚本自动化完成树模型 HLL 至 HDL 的转换，从而简化了部分部署流程，但该方法仍依赖用户的编程基础，且脚本开发与调试周期较长，实际易用性依然受限。此外，高层次综合(high-level synthesis, HLS)编译器虽为基于 HLL(如 C/C++/Matlab)编写的机器学习软件模型提供了一条向硬件映射的便捷途径，但其生成的硬件设计往往存在性能次优问题，难以充分挖掘 FPGA 的并行计算潜力[41]。因此，如何实现低门槛且高效的模型硬件部署，仍是当前领域面临的一项关键挑战。

LLM 有望成为解决该挑战的核心手段，其兼具强大的文本、代码生成能力与高效易用的特性。然而，目前关于 LLM 在硬件部署领域的研究主要集中在优化 LLM 自身的部署上，关于利用 LLM 辅助其他机器学习模型实现高效硬件部署方面的潜力尚未得到充分挖掘。例如，FlightLLM[42]提出了一套全面的映射流程，用于在 FPGA 上进行高效的 LLM 推理，有效解决了计算和内存开销问题，与现有的 GPU 相比提供了更高的能效和成本效益。FlightVGM[43]则通过在线稀疏化和混合精度技术，在 FPGA 上实现了高效的视频生成模型(video generation model, VGM)推理，显著提升了性能和能效。上述工作均聚焦于优化大模型自身的部署效率与能效瓶颈。相比之下，如何利用 LLM 强大的语义理解与代码生成能力，将

其作为自动化工具辅助其他传统机器学习模型完成从高级语言到硬件描述语言的转换，仍是尚未被充分探索的方向。本文针对这一空白，提出一种 LLM 驱动的集成约简与 Verilog HDL 生成框架，旨在为传统模型的软硬件协同部署提供低门槛、高自动化的新范式。

2 LLM 驱动的集成约简方法

本节介绍了用于集成约简的基于 LLM 的组合优化技术。利用 LLM 处理复杂问题描述与约束条件的能力阐述了将集成约简问题转化为简洁文本描述的方法论，从而使 LLM 能够在短时间内提供高质量的解决方案。

2.1 优化问题描述

提示词的构建对于 LLM 解决问题至关重要。提示词的质量往往决定了 LLM 的最终性能。为了设计高质量的提示词，首先需要对集成约简问题进行简洁的描述，以降低问题的复杂度。

对于 RF 模型 $RF_{\text{init}}=\{t_1,t_2,\cdots,t_n\}$，集成约简的目标是选择一个子集 RF_{rdc}，并且要求它与 RF_{init} 之间的性能差距尽可能小。该子集由 m 棵树组成，表示为 $RF_{\text{rdc}}=\{t_j\}^m$，其中 $0<j\leqslant n$，$0<m<n$。为了评估集成模型的性能，可以比较它们在相同数据集上的预测结果准确率。对于二分类任务数据集 $X=\{x_1,x_2,\cdots,x_N\}$，每个样本 x_i 都有对应的真实标签 $y_i\in\{0,1\}$。使用 RF_{init} 中的每棵决策树 t_j 预测 x_i，可获得相应的预测标签 $\hat{y}_{ij}\in\{0,1\}$。通过投票机制，可以得到森林 RF_{init} 对 x_i 的预测结果 $\hat{y}_i\in\{0,1\}$。随后，集成森林中每棵树的预测结果可以获得如图 1 所示的矩阵，其中第 i 行第 j 列代表树 t_j 对样本 x_i 标签的预测结果 $\hat{y}_{ij}$，最后一列则是样本的真实标签 y_i。这就是对应于模型的真值表，该方法最早由 CANOPY[5] 提出。真值表不仅清晰地描述了每个树模型的预测结果，还简明扼要地展示了每棵树对最终结果的贡献。这种形式提高了利用 LLM 解决集成约简问题的可行性。

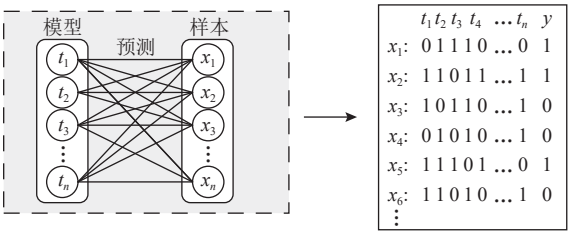


Fig. 1 Example of the truth table for an ensemble model

图 1 集成模型的真值表示例

2.2 基于 LLM 的约简框架

LEMON 的工作流程如图 2 所示。基于集成模型的真值表，构建一个组合优化求解提示词。通过多次迭代生成和评分机制，将 LLM 提供的最佳结果作为输出。

LEMON 的第 1 步是构建初始决策树模型池。随后，遍历数据集中的每个样本，并为所有树分别构建相应的真值表。为了给 LLM 提供一些有价值的子集参考，在第 1 次迭代开始前，通过随机选择模型组合来初始化历史解数据库。

2.2.1 解的初始化

少样本学习在约简过程中起着重要作用，因此选择使用数据库来记录生成的子集。该数据库主要用于：1）存储 LLM 生成的子集及其得分；2）为 LLM 提供历史解；3）在迭代结束时返回表现最佳的子集。为确保子集规模严格遵循预设的数量约束，LEMON 在初始化阶段会根据初始池的规模来生成不同大小的子集。在每个迭代周期中，LEMON 会基于量化评分指标从历史解数据库选出分数最高的 K（Top-K）个最优子集作为示范样本。这些选定的子集随后经过提示词工程优化。对于 LLM 新生成的解，采用一个带有评分机制的评估框架将其中合格的子集整合到现有库中，从而通过连续迭代逐步扩展解空间。

2.2.2 提示词构建

高质量的提示词是引导 LLM 高效求解的关键，其设计直接决定了最终性能表现。如图 3 所示，在

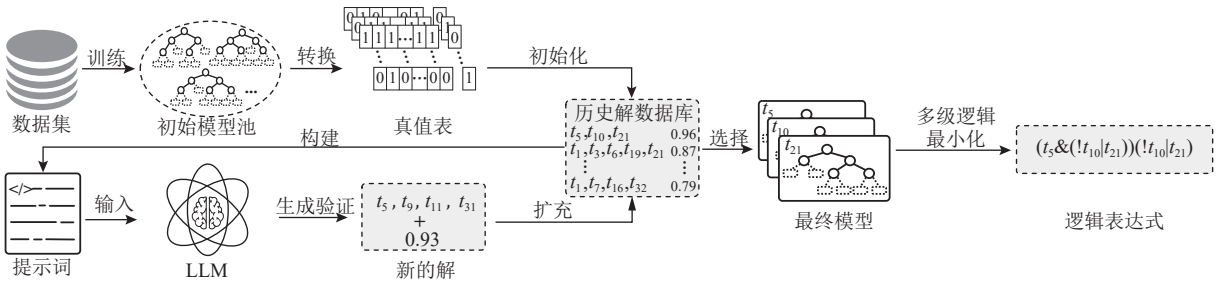


Fig. 2 The workflow of LEMON

图 2 LEMON 的工作流程

每次迭代中，提示词主要包含以下组成部分：

1)系统指令。这部分提示词主要帮助 LLM 明确其角色和任务目标。该指令首先会告知 LLM 其所扮演的角色，随后描述整体任务、格式及其他相关信息。

2)真值表。如上文所描述的那样，LEMON 使用真值表来描述与集成约简相关的组合优化问题。它包括树模型与真实标签之间的关系。

3)任务详情。这一部分提供关于约简问题的详细描述，包括变量信息和真值表的总体描述(例如，真值表中的每一列对应一个变量)。向 LLM 传达真值表的基础知识有助于其更好地理解任务目标。

4)历史子集。基于历史解数据库，通过评分机制筛选出 Top-K 的最优子集作为示范样本，用于构建 LLM 的少样本提示上下文，以引导其生成高质量的解决方案。

```
请根据下方的真值表，帮助我识别关键变量。对于最终的关键变量，至少要满足当半数及以上的关键变量取值为1时，输出值也为1。你的任务是给出一个变量的组合。

以下是真值表：
11111111100111111…1 1
11111111111111111…1 1
00011000111100100…0 0
⋮
11111111111110111…1 1

真值表说明：
①每行对应一个输入组合（最小项），其包含100 个布尔变量x[0]~x[99]的逻辑状态 + 空格 + 单比特输出（0/1）。
②输出列严格定义目标布尔函数的 ON集合（逻辑1）与 OFF集合（逻辑0）。
③从左至右每个比特依次对应输入变量 x[0]~x[ 99]。

请给出满足以下条件的变量组合：
①所选变量需要对输出逻辑具有决定性贡献，剔除无关或冗余变量。
②要求至少当变量组合中超过半数的变量取值为逻辑 1 时，组合逻辑输出应稳定为 1。
③在严格满足逻辑功能等价的前提下，追求变量数量最小化。
④结果必须以 <comb> 开头，以 </comb> 结尾。
⑤可选变量范围：x[0]~x [99]。

输出格式：
<comb>x[编号],x[编号],…,x[编号]</comb>

仅输出新的组合，不要包含任何其他内容。
```

Fig. 3 Example of prompts for ensemble reduction

图 3 集成约简提示词示例

构建完成的提示词输入至 LLM 后，模型将依据提示词中的真值表结构与任务指令生成候选子集。对于最终生成的结果，首先通过噪声过滤器过滤掉其中无关的信息。随后，使用评估器计算其中每个子集相应的得分。最后，子集及其记录的得分被添加到数据库中。

2.2.3 量化评估

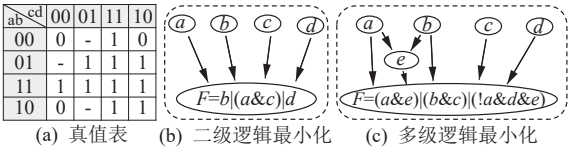
为量化评估历史解数据库中各候选子集的综合质量，本文设计了一套基于部分训练集分类准确率的量化评估器(evaluator)。鉴于 LEMON 框架的核心目标是在显著压缩集成模型规模的同时维持甚至提升其泛化性能，评估器直接以候选子集在对应数据集上的分类准确率作为核心量化指标。具体评估流程如下：对于 LLM 在单次迭代中生成的每一个树模型子集，评估器首先提取该子集内所有基学习器对评估样本的独立预测输出。随后，通过多数投票机制进行集成推理，生成最终预测标签并与真实标签进行逐样本比对，进而计算得出该子集的准确率得分。在此评估范式下，量化得分越高，表明该子集在保障模型预测性能的同时具备更优的约简质量。

2.3 集成模型性能增强

集成模型推理最直观的实现方式是多数投票机制。尽管该方法在多数情况下表现良好，但当基模型中存在噪声较大或高度相关的个体时，其整体性能往往会显著下降。加权投票和堆叠等替代方法提供了部分解决方案，但引入了复杂性或参数敏感性问题。CANOPY[5] 提出使用逻辑最小化表达式进行集成输出计算，在多个数据集上取得了显著的性能。然而，其对二级逻辑最小化的依赖从根本上限制了进一步的性能提升。

逻辑最小化是逻辑综合的一部分。在数字电路设计过程中，逻辑综合根据指定的行为，使用逻辑门创建电路的简洁描述。逻辑最小化(可分为二级和多级)用于通过减少逻辑表达式中的项数或变量数来简化数字电路。其目标是在保持功能的同时，降低电路的复杂度、面积、功耗和延迟。二级逻辑最小化仅以积之和(sum-of-products, SOP)的形式组织电路结构，难以处理复杂逻辑。相比之下，多级逻辑最小化没有这一限制，因此支持更复杂的逻辑结构。如图 4 所示，真值表中的0和1表示相应的最小项分别属于布尔函数的 OFF 集合(off-set)和 ON 集合(on-set)。可以看到，多级逻辑最小化在简化过程中生成可重用的中间项，从而显著降低最终表达式的复杂度和生成时间。然而，二级逻辑最小化只能生成 SOP 形式的表达式，导致在复杂逻辑场景下最终表达式的复杂度和生成时间显著增加。

因此，本文引入多级逻辑最小化来增强约简后



(a) 真值表 (b) 二级逻辑最小化 (c) 多级逻辑最小化

Fig. 4 Example of 2-level and multi-level logic minimization

图 4 二级和多级逻辑最小化示例

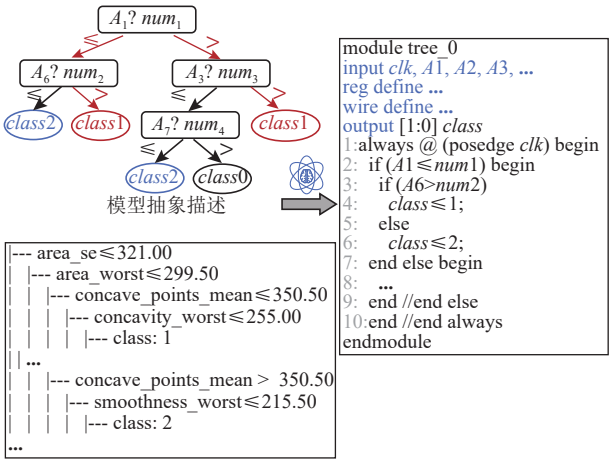
模型的性能。具体而言，首先基于 LEMON 生成的最终模型构建真值表。随后，对该真值表执行多级逻辑最小化，并基于生成的逻辑表达式计算模型输出。此处，使用逻辑最小化工具 ABC[44] 进行多级逻辑最小化。

3 基于 LLM 的自动化硬件部署

将集成模型的 HLL 描述转化为 HDL，是实现硬件部署的关键步骤。集成模型通常基于 Python 等高级语言训练与导出，其内部结构多以 JSON/字典等序列化格式存储（包含节点分裂条件、分支路径与叶节点标签等信息），缺乏可直接映射至数字电路的结构化硬件描述。传统方法（如使用 HLS 或脚本）虽已用于此类转换，但存在一定局限性。LLM 的出现为解决这一问题提供了新方案。从 HLL 到 HDL 的转换本质上属于自然语言处理范畴，而这正是 LLM 的强项。

使用 LLM 实现模型自动化硬件部署的优势主要体现在以下几个方面：首先，就所需专业知识而言，HLS 要求开发者精通 C/C++及 HLS 优化指令，脚本化转换器需掌握模型结构解析与 HDL 语法，人工 RTL 编写依赖资深数字集成电路设计经验，而 LLM 驱动生成仅需掌握模型结构描述与基础提示词工程，显著降低了技术门槛。其次，在开发与迭代周期上，HLS 需经历反复的编译、综合和调试循环，脚本转换需为不同模型结构定制解析逻辑，人工编写则面临高耗时与易错风险，而 LEMON 可通过多轮对话生成并自动补全，显著缩短部署周期。而在泛化能力方面，传统方法通常需针对特定模型定制，泛化能力受限，而 LLM 能够直接基于拓扑描述生成 HDL 代码。因此，LLM 驱动范式在降低工程门槛、提升部署效率与保障代码质量之间取得了良好平衡，为资源受限场景下的树模型硬件部署提供了更具可扩展性的解决方案。尽管早期 LLM 受限于专用训练数据的匮乏，在 Verilog HDL 代码生成任务中性能有限，但近期研究表明，当前主流 LLM 已经能够高效且准确地完成 Verilog HDL 代码相关任务[21-22]。

如图 5 所示，左侧呈现了决策树模型的抽象结构和文本描述（由 Python 库直接导出），右侧展示了 LLM 生成的 Verilog HDL 代码实例。可以看出，树模型的层级判定路径与硬件描述语言中的多路条件分支逻辑具有天然同构性。在推理阶段，输入样本自根节点起，依据各内部节点的特征阈值逐层路由，最终抵达叶节点并获得预测类别。基于该结构映射关系，LEMON 首先调用 Python 库导出树模型的序列化拓扑文本，进而构建专用提示词，驱动 LLM 自动化生成等效的可综合 Verilog HDL 代码。



模型结构的文本描述示例

Fig. 5 Examples of Verilog HDL code generated by LLM

图 5 LLM 生成的 Verilog HDL 代码实例

在获取 LLM 的生成结果后，LEMON 首先利用正则表达式提取 Verilog HDL 代码，并通过脚本自动清除注释及乱码内容。随后，调用商业 EDA 工具对提取的代码进行语法校验，以确保其符合硬件描述语言的语法规范。此外，针对 LLM 受 token 长度限制而在处理深层树状结构时可能引发的代码截断问题，LEMON 设计了一套多轮对话补全机制。当检测到输出代码不完整时，系统将上一轮生成内容整合至提示词上下文，引导 LLM 结合初始任务描述与历史交互信息逐步续写。该机制有效保障了生成代码的逻辑完整性与可执行性。

4 实验评估

本节通过软件和硬件实验对 LEMON 的有效性进行验证。首先介绍用于验证方法的数据集，随后分别阐述软件和硬件的实验流程，最后对结果进行分析。

4.1 数据集与实验设置

实验使用来自 UCI 机器学习库①和 Kaggle②的 20

① https://archive.ics.uci.edu/ml

② https://www.kaggle.com/datasets

个公开数据集来验证 LEMON。这些数据集源自各种现实世界的应用问题,并在机器学习社区中被广泛用于多种目的。表 1 总结了这些数据集的详细信息。对于 K 类多分类任务,LEMON 在集成约简时采用 1 对多编码策略,首先将多分类问题分解为 K 个独立的二分类任务。随后对分解后的每个二分类任务分别进行实验。所有树模型均通过 Python 库训练获得。

Table 1 Details of Datasets Used in the Experiments

表 1 实验所用数据集详情

数据集	样本数量	特征数量	标签数量
Maternal_health_risk	452	6	3
Breast_cancer	699	9	2
Pima_indians	768	8	2
Phishing_websites	1 353	9	2
Auction_verification	2 043	7	2
Obesity	2 111	16	7
kr_vs_Kp	3 195	35	3
Bank_additional	4 119	16	2
Students_dropout	4 424	36	3
TUANDROMD	4 464	241	2
Spambase*	4 601	57	2
Waveform	50 000	21	3
Shill_bidding	6 321	13	2
Taiwan_bankrupt	6 819	95	2
Customer_churn	7 043	20	2
Banana_quality	8 000	7	2
Room_occupancy	10 129	18	2
NATICUSdroid	29 333	86	2
Accelerometer_gyro	31 991	7	2
Internet_firewall	65 532	11	4

注:*表示数据集存在缺少特征的样本。

软件和硬件实验均采用 DeepSeek-V3[20] 作为 LLM 代理,用于执行集成约简和 HLL 到 HDL 的转换。在集成约简过程中,温度设置为 1 以保障解空间多样性,最大输出 token 数设置为 512。历史解数据库每轮选取评分最高的前 5 个子集作为少样本示范来引导 LLM 定向探索。迭代终止条件设为最大 10 轮或连续 3 轮无性能提升。在硬件部署部分,温度设置为 0.1 以避免生成语法错误,最大输出 token 数设置为 4 096 以支持长代码完整生成。

软件评估实验在一台配备 3.3 GHz CPU 的 Linux 工作站上进行。在初始化阶段,使用决策树构建初始模型池。随机选择大小为初始模型池 1/3、1/5 和 1/10 的子集,以初始化历史解数据库。此处子集采样比例是一个可配置参数。用户可根据实际应用需求灵活调整,例如可将初始化子集比例压缩至初始模型池的 1/20 即可进一步约束 LLM 生成解中的子集规模。

对于硬件实现实验,采用精心设计的提示词,引导 LLM 将基于 HLL 的软件集成模型转换为基于 HDL 编码的硬件设计。最终在采用 28 nm 高介电常数金属栅极(high-k metal gate, HKMG)工艺技术的 Xilinx 7 系列 FPGA 上进行物理实现。比特流(bitstream)文件使用 Vivado v2022.2 生成。

本文选取 CANOPY[5] 与 HERALD[7] 作为实验的基线方法,主要基于以下 2 点考量:首先,二者均专门针对 RF 这类集成模型设计。其次,它们在模型约简方面均采用逻辑最小化技术,与本文的研究范式高度契合。相比之下,近期涌现的代表性工作如 Ibragimov 等人[8] 提出的方法与 TreeLUT[36] 主要面向 GBDT 模型。尽管 RF 与梯度提升模型的基学习器均为决策树,但二者在集成机制与预测逻辑上存在本质差异。RF 通过并行采样构建相互独立的决策树,并依赖多数投票进行最终预测。而 GBDT 采用串行迭代方式训练决策树,后续每一棵新树均以前序模型预测误差为拟合目标,并通过加权累加各树输出以实现预测。这种底层机制的差异导致针对 GBDT 设计的约简策略难以直接迁移至 RF。因此,为确保实验对比的公平性与针对性,本文优先选取同属 RF 范畴且技术路线相近的 CANOPY 与 HERALD 作为基线,以期在统一的评价体系下客观验证本文方法的有效性与优越性。

4.2 约简后集成模型的性能评估

每个数据集被随机划分为 2 个不相交的集合用于训练和测试,划分比例为 10%~90%。其中 90% 的数据随机选作训练集,剩余 10% 用作测试集。

在 20 个公开数据集上将 LEMON 与 RF、CANOPY[5] 和 HERALD[7] 方法进行了性能对比。如表 2 所示,实验设置初始树池规模为 50 和 100,所有方法均采用与 RF 一致的树模型池以确保公平比较。如表 3 所示,LEMON-vote 与 LEMON-abc 分别表示采用投票机制和多级逻辑最小化方法,以生成经 LEMON 约简后的集成模型输出。

1)准确率。表 3 显示,在所有实验条件下,LEMON 的性能始终优于 CANOPY 和 HERALD,并且在大多数数据集上实现了与 RF 一样甚至更高的分类准确

Table 2 Comparison of Reduced Ensemble Size

表 2 约简后集成规模比较

数据集	初始模型池规模为 50		初始模型池规模为 100	
	LEMON	CANOPY	LEMON	CANOPY
Maternal_health_risk	4.40	7.50	15.82	7.02
Breast_cancer	3.76	4.34	14.58	4.26
Pima_indians	3.42	13.42	18.12	11.76
Phishing_websites	7.24	4.42	24.34	4.16
Auction_verification	3.88	5.38	24.62	4.14
Obesity	3.56	8.12	55.46	8.38
kr_vs_Kp	3.66	4.94	49.36	5.12
Bank_additional	3.46	18.90	50.76	18.58
Students_dropout	2.56	26.72	22.38	26.40
TUANDROMD	5.88	3.40	25.90	2.66
Spambase	3.24	15.4	40.02	15.40
Waveform	2.60	21.6	36.96	19.90
Shill_bidding	4.86	4.04	31.66	4.06
Taiwan_bankrupt	6.68	15.56	56.74	14.84
Customer_churn	3.50	38.74	31.26	42.20
Banana_quality	3.06	16.32	34.62	16.10
Room_occupancy	4.24	1.52	9.40	1.40
NATICUSdroid	3.08	20.50	51.88	19.10
Accelerometer_gyro	3.02	12.02	41.44	9.80
Internet_firewall	3.78	1.54	12.86	1.22
平均数量	3.99	12.22	32.41	11.83

率，在极少数数据集上略低，但仍保持相当水平。此外，HERALD 和 CANOPY 在准确率上的结果相近，这是因为二者在约简方法层面是相同的。除此之外，对于多级逻辑最小化对性能的影响，当初始池规模为 50 时，在大多数情况下，LEMON-vote 的表现优于 LEMON-abc。而当初始池规模增加到 100 时，LEMON-abc 的表现始终优于 LEMON-vote。造成这种性能表现差异的主要原因是，基于 LLM 的约简策略所导致的集成规模动态变化。尤为关键的是，多级逻辑最小化技术在处理复杂逻辑结构时展现出显著优势，这意味着 LEMON 的性能增益在大规模集成模型上更为突出。这一发现充分验证了 LEMON 框架具有良好的可扩展性，能够适应不同规模与复杂度的应用场景。

2）集成规模。表 2 显示，当初始池规模为 50 时，LEMON 约简后的集成规模普遍小于 CANOPY。但当池规模增加到 100 时，LEMON 的集成规模则普遍更大。这种行为源于 LLM 的少样本学习机制，该机制促使新生成的子集规模与历史解数据库中的子集规模相似。在实验初始化阶段，本文采用 1/3、1/5 与 1/10 的比例进行随机采样，旨在保障解空间的多样性。该策略在提升多样性的同时，也在一定程度上限制了 LLM 生成解中子集的规模，进而导致了前述的规模差异。需要注意的是，采样比例为一项用户可配置参数。在历史数据库初始化阶段调低该比例，即可直接限制 LEMON 约简后集成模型的规模。例如，将比例降至 1/20 或 1/50，可实现对模型规模的更严格约束。

3）运行时间。图 6 展示了 LEMON 和 CANOPY 在不同初始池规模下的运行时间对比分析。数据集在横轴上按真值表大小升序排列。根据实验结果观察到以下现象：相较于 CANOPY 与 HERALD，LEMON 在不同规模数据集上的运行时间波动更小，展现出更优的可扩展性；总体而言，HERALD 的运行时间显著短于 CANOPY，印证了原论文结论，即 HERALD 在维持精度相当的前提下，有效降低了计算开销；在多数测试场景中，LEMON 的计算效率与 HERALD 相当甚至更优，尤其在初始候选池扩容与数据样本量同步增长的条件下，其时间优势更为凸显。值得注意的是，对于最后 10 个具有较大真值表的数据集（如图 6 所示），LEMON 在集成约简任务上实现了显著的效率提升，平均加速比分别达到 CANOPY 的 9 倍和 338 倍。

图 6 中呈现的性能差异来源于各方法底层机制的本质区别。CANOPY 采用的逻辑最小化算法对真值表规模高度敏感，致使其在处理具有较大真值表的数据集（如 Students_dropout 和 Customer_churn）时，所需处理时间超过 1 h。HERALD 虽通过分层策略有效压缩了运行时间，但并未带来显著的准确率增益。相比之下，LEMON 的计算开销主要取决于 LLM 生成序列的长度，而非输入样本规模，因而能够在不同体量数据集上保持稳定的处理效率与一致的时序表现。此外，在运行成本方面，LEMON 单次迭代的平均输入 token 消耗约为 23 000，该指标主要受真值表规模影响。具体而言，大多数数据集的单次迭代消耗控制在 5 000 以内。而个别具有较大真值表的数据集（如 Customer_churn、Banana_quality）单次消耗超过 50 000，从而拉高了整体均值。由于输出只需要给出模型组合，因此 token 消耗显著低于输入，平均维持在 300 以内。

4）方法通用性。表 4 展示了 LEMON-abc 在接入不同 LLM 时的约简性能对比实验。为验证框架的模型无关性，在统一提示词模板与核心参数（温度、Top-

Table 3　Accuracy Comparison on Different Ensemble Sizes

表 3　不同集成规模下的准确率比较　%

数据集	初始模型池规模为 50 时					初始模型池规模为 100 时				
	RF	CANOPY	HERALD	LEMON-vote	LEMON-abc	RF	CANOPY	HERALD	LEMON-vote	LEMON-abc
Maternal_health_risk	72.696±6.198	68.823±6.770	65.299±7.399	**88.805±8.915**	78.849±7.602	74.478±4.706	71.770±6.135	73.005±2.841	78.186±6.848	**91.676±7.540**
Breast_cancer	96.686±2.225	86.131±7.258	80.846±9.869	**99.242±2.620**	92.394±4.854	96.629±1.931	88.541±5.574	89.069±4.069	94.303±5.208	**97.189±3.108**
Pima_indians	75.948±3.704	73.523±4.225	66.024±2.561	**93.436±8.798**	76.047±4.484	76.026±5.353	72.247±5.401	75.427±1.909	79.455±6.823	**94.847±6.811**
Phishing_websites	**92.986±3.076**	86.258±6.011	84.646±7.170	92.523±8.859	92.198±5.478	92.725±2.423	87.533±4.476	94.394±5.109	86.825±6.771	**98.215±2.440**
Auction_verification	98.312±0.977	92.178±3.448	95.717±1.494	**99.096±2.150**	92.707±4.491	98.332±1.111	92.806±3.577	98.032±1.713	95.863±3.234	**99.009±1.932**
Obesity	96.766±1.266	92.264±2.374	95.072±2.071	**98.939±2.438**	91.428±3.516	96.957±1.175	92.903±2.528	92.438±0.341	95.923±2.059	**98.838±3.082**
kr_vs_Kp	**99.062±0.602**	96.466±1.678	97.672±1.695	98.717±2.568	96.061±2.265	99.287±0.473	97.293±0.933	98.581±0.942	98.570±1.525	**99.457±1.244**
Bank_additional	**91.010±1.258**	79.022±2.771	69.952±1.836	90.791±8.211	81.312±3.753	91.097±1.176	82.810±2.216	70.465±2.566	81.432±3.795	**98.455±3.625**
Students_dropout	84.916±2.040	80.504±2.382	74.759±2.012	**99.862±0.683**	86.859±3.043	84.655±2.033	80.346±1.423	79.583±2.043	87.317±7.323	**91.069±8.489**
TUANDROMD	**99.477±0.337**	97.052±2.614	96.583±2.329	97.506±3.726	98.455±2.048	99.512±0.315	97.523±2.245	98.216±0.740	98.401±2.253	**99.615±0.861**
Spambase	95.074±0.973	90.998±1.763	91.377±0.211	**99.201±2.168**	89.070±2.383	94.336±1.068	92.503±0.899	90.334±1.809	94.988±3.527	**96.674±4.460**
Waveform	90.396±1.342	81.500±1.944	81.893±1.177	**99.066±3.012**	87.587±4.789	90.888±1.039	83.773±1.192	83.203±2.881	89.471±5.888	**92.859±8.284**
Shill_bidding	**99.601±0.235**	95.990±2.069	98.119±1.634	97.953±3.269	96.429±3.420	99.519±0.258	97.316±1.469	97.391±1.221	95.654±2.752	**99.654±0.971**
Taiwan_bankrupt	**96.956±0.599**	88.054±2.500	81.539±4.141	85.718±3.765	92.512±3.636	97.070±0.562	91.253±1.528	89.460±3.774	90.798±3.162	**98.409±2.299**
Customer_churn	78.877±1.406	73.505±1.800	67.968±2.047	**86.078±9.273**	83.489±2.505	78.996±1.199	75.764±1.534	64.922±0.434	77.948±4.594	**87.662±1.510**
Banana_quality	93.000±0.603	97.080±1.070	93.689±0.848	**99.067±2.390**	91.026±2.506	96.963±0.596	95.020±1.005	94.416±0.358	94.209±2.638	**98.558±1.811**
Room_occupancy	**99.982±0.042**	97.073±3.824	98.246±3.039	99.833±1.178	99.430±1.479	**99.973±0.054**	97.940±2.132	99.708±0.756	99.666±1.211	99.726±0.883
NATICUSdroid	92.893±0.853	86.612±1.345	89.162±1.830	**99.230±2.806**	94.522±2.512	93.131±1.000	88.930±1.185	90.678±1.026	89.464±5.034	**96.080±5.520**
Accelerometer_gyro	**99.494±0.111**	89.501±2.094	87.374±3.971	98.857±3.314	97.910±3.015	**99.493±0.138**	92.252±1.541	91.033±2.975	90.992±3.452	99.302±1.733
Internet_firewall	**99.987±0.012**	96.079±4.072	97.436±4.441	99.800±1.414	98.773±2.294	**99.991±0.013**	96.479±3.494	97.483±2.181	99.212±2.333	99.529±1.394
平均结果	93.597±1.080	88.536±2.666	85.669±3.089	**96.426±3.890**	91.434±3.201	93.831±1.110	89.589±2.524	88.392±1.984	91.740±4.022	**97.012±3.340**

注：黑体数值为指标最优值，下划线数字为指标次优值。该过程重复 50 次，并计算平均值和标准差。

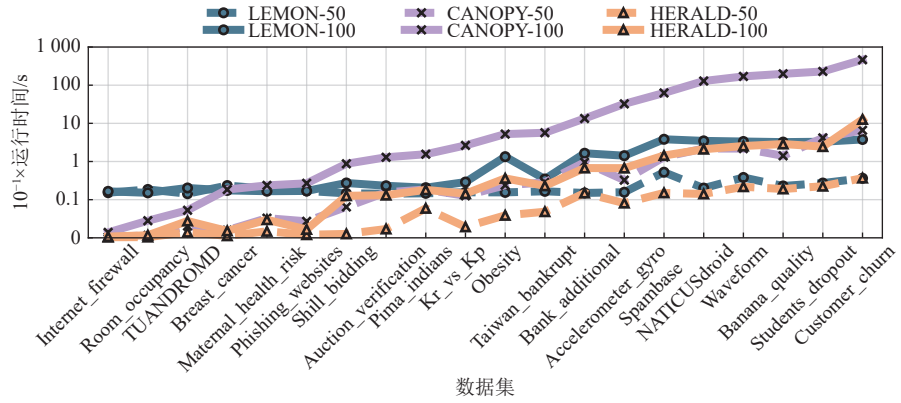


Fig. 6 Comparison of required running time for LEMON and CANOPY reduction

图 6 LEMON 和 CANOPY 约简所需运行时间的比较

K、迭代轮数等)配置下，额外测试了 GPT-4o 与 Qwen3-Max 两种主流模型。为了更清晰地展示，表 4 中仅列出了在部分数据集上初始模型池规模 50 时的结果，其余设置下结果相同。对比表 2 和表 3，实验结果表明，所有测试 LLM 均能准确理解真值表逻辑约束并稳定输出高质量候选子集，充分证明 LEMON 的核心优化机制不依赖于特定基座模型的架构。该设计赋予了框架高度的可替换性与工程灵活性，使其能够无缝适配未来不断演进的 LLM 生态，进一步提升了方法的学术严谨性与实际部署可行性。

5)各个模块对 LEMON 性能的影响。表 5 评估了 LEMON 各核心组件相对于基线 LLM(零样本提示)的增量贡献。实验选取初始模型池规模为 50 的部分数据集进行对比分析，结果表明：①基线 LLM 因缺乏示例引导，生成解的随机性较高，导致约简后模型规模偏大且分类准确率下降；②移除历史解库后，尽管评估器仍可筛选单次最优解，但由于缺失少样本上下文，LLM 探索能力受限，模型规模与准确率均显著退化，验证了基于历史解的少样本学习在引导高质量解空间搜索中的关键作用；③移除评估器时，算法退化为每轮随机抽取 5 个子集构建少样本提示，并直接输出历史解中规模最小者，因缺乏基于性能的排序机制，分类准确率明显下降，但少样本引导仍有效抑制了模型规模的膨胀；④关闭迭代优化(仅执行单轮生成)时，因评估器与历史解库已参与初始构建，最终性能与完整流程的差异甚微。该现象表明 LEMON 具备快速收敛特性，仅需极少迭代轮次即可实现高效约简，进一步印证了其算法效率。

Table 4 Accuracy Comparison of LEMON Ensemble Reduction Using Different LLM Agencies

表 4 使用不同 LLM 代理时 LEMON 集成约简准确率对比

数据集	GPT-4o		Qwen3-Max	
	准确率/%	数量	准确率/%	数量
Spambase	95.070±1.423	4.31	90.997±1.521	3.12
Waveform	86.114±2.683	1.84	93.500±3.642	2.25
Obesity	90.658±4.496	3.30	92.428±2.326	4.01
kr_vs_Kp	98.105±2.176	3.47	97.532±1.277	2.97

注：“±”前数值表示平均值，“±”后数值表示标准差。

Table 5 Accuracy Improved by Each Module of LEMON

表 5 LEMON 各个模块带来的准确率提升

数据集	基线 LLM	准确率/%			
		完整框架	缺少历史解	缺少评估器	缺少迭代
Spambase	83.74	99.20	85.31	80.67	92.20
Waveform	79.10	99.07	82.52	79.21	87.15
Obesity	89.31	98.94	90.27	79.59	85.64
kr_vs_Kp	89.42	98.72	87.63	83.37	95.29
约简后决策树数量/颗	30.16	3.27	31.58	4.19	2.83

4.3 硬件部署

本文对比了 RF、CANOPY 和 LEMON 三种方法所生成集成模型硬件实现后的开销，所有 Verilog HDL 代码均基于 LEMON 框架集成的 LLM 技术生成。为确保比较的公平性，所有集成模型统一采用投票机制进行预测，并将初始模型池规模设定为 50 棵决策树(此时 LEMON-vote 性能最佳)。硬件实现均基于流水线架构，旨在投票阶段高效聚合每棵树的输出，逻辑综合过程将这些流水线结构映射至由触发器构成的寄存器阵列。所有生成代码通过 Vivado 进行功能仿真，将仿真输出与原始 Python 模型在相同输入下的预测结果逐样本比对以确保功能正确，且未发现时序违例或组合逻辑振荡等硬件异常。

尽管表 3 的实验结果已表明 LEMON 能有效压

缩集成模型的规模,但模型规模的缩减并不直接等同于硬件开销的降低。为此,本文进一步对比了3种方法所生成集成模型的实际硬件资源消耗。为确保对比的公平性,所有模型均统一采用LEMON的部署方案进行硬件映射,以排除部署策略差异带来的干扰。表6展示了4种方法在硬件部署后详细的资源开销。实验结果表明,LEMON在保持与RF相当或更高准确率的同时(在大多数数据集上),实现了超过64.7%的功耗降低和超过92.6%的整体硬件利用率提升。平均而言,LEMON所需的寄存器数量比RF少92.2%。

此外,图7展示了在Banana_quality数据集上,RF、CANOPY和LEMON三种方法之间的硬件利用率对比分析。通过布局视图中的青色高亮区域,直观呈现了底层逻辑门与布线网络的物理实现细节。实验结果表明,LEMON展现出卓越的硬件利用率,能够在硬件资源消耗不足RF模型10%的情况下保持相当的分类准确率。

Table 6 Comparison of Hardware Resource Consumption of Different Methods

表6 不同方法硬件资源消耗比较

数据集	Maternal_health_risk				Breast_cancer				Pima_indians				Phishing_websites			
	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W
RF	780	77	14	0.252	460	77	14	0.233	2 760	77	16	0.747	460	77	22	0.175
CANOPY	113	10	1	0.113	54	6	3	0.108	755	16	6	0.224	**40**	**6**	**0**	0.104
LEMON	**6**	**6**	**1**	**0.100**	**48**	**6**	**1**	**0.100**	**186**	**6**	**0**	**0.119**	44	6	4	**0.103**

数据集	Auction_verification				Obesity				kr_vs_Kp				Bank_additional			
	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W
RF	758	77	53	0.189	3 558	77	89	0.747	995	77	8	0.559	2 971	77	88	0.747
CANOPY	106	11	**2**	**0.103**	529	11	5	0.223	57	6	0	0.116	1 479	40	18	0.621
LEMON	**47**	**6**	5	0.108	**272**	**6**	**0**	**0.163**	**50**	**6**	**0**	**0.109**	**187**	**6**	**3**	**0.147**

数据集	Students_dropout				TUANDROMD				Spambase				Waveform			
	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W
RF	3 953	77	70	0.747	446	77	22	0.280	4 992	77	86	0.747	5 561	77	285	0.747
CANOPY	2 137	36	49	0.747	**33**	**6**	1	**0.109**	1 629	20	6	0.747	2 259	24	113	0.747
LEMON	**256**	**6**	**9**	**0.198**	40	8	**0**	0.111	**310**	**6**	**1**	**0.476**	**320**	**5**	**3**	**0.222**

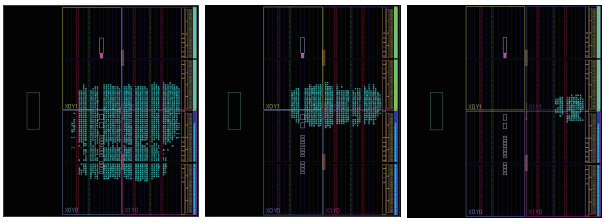
数据集	Shill_bidding				Taiwan_bankrupt				Customer_churn				Banana_quality			
	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W
RF	1879	77	1	0.747	6 632	77	61	0.747	3 177	76	74	0.747	5 812	77	255	0.747
CANOPY	**152**	**6**	**0**	**0.122**	2 215	18	10	0.747	2 649	62	55	0.747	2 277	28	84	0.747
LEMON	289	8	4	0.129	**484**	**6**	**12**	**0.747**	**281**	**6**	**2**	**0.143**	**437**	**6**	**11**	**0.163**

数据集	Room_occupancy				NATICUSdroid				Accelerometer_gyro				Internet_firewall			
	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W	LUT	FF	Mux	功耗/W
RF	1 509	76	6	0.43	585	77	34	0.263	3 777	76	45	0.747	1 776	77	1	0.424
CANOPY	**98**	**6**	1	0.127	281	37	34	0.147	1 017	16	9	0.279	**41**	**2**	**0**	**0.107**
LEMON	111	6	**0**	**0.123**	**37**	**3**	**7**	**0.101**	**329**	**6**	**0**	**0.132**	120	6	0	0.120

注:黑体数值为指标最优值。LUT表示查找表,FF表示触发器,Mux表示多路选择器。

5 结 论

本文提出LEMON,一种面向高效硬件实现的LLM驱动的集成约简与Verilog HDL生成框架。该框架能够在保持甚至提升预测准确率的同时,显著压缩集成模型的规模。该目标主要通过结合上下文学习与少样本学习技术的高效约简策略得以实现。针对约简后模型中存在的复杂逻辑,LEMON进一步引入多级逻辑最小化方法,在保持准确率的前提下优化硬件资源开销与运行效率。此外,LEMON还具备基于LLM的自动化硬件代码生成能力。依托



(a) RF (b) CANOPY (c) LEMON

Fig. 7 Layout view of resource utilization on Banana_quality dataset

图 7 Banana_quality 数据集上资源利用率的布局视图

LLM 在代码理解与生成方面的优势，用户仅需提供自然语言提示词与模型结构描述，即可自动将软件端的分类树集成模型转换为可直接部署于 FPGA 的 Verilog HDL 代码。该方法显著降低了从 HLL 到 HDL 的转换复杂度，使硬件部署流程更加高效便捷。本研究工作的代码开源地址为“https://github.com/TechnologyAiGroup/LEMON”。

作者贡献声明：皇甫璟轩负责算法实现、实验验证及论文撰写；王虹飞提供整体方案与算法设计、实验设计指导并修改论文。

参 考 文 献

[1] Elnawawy M, Sagahyroon A, Shanableh T. FPGA-based network traffic classification using machine learning[J]. IEEE Access, 2020, 8: 175637−175650

[2] Lin Xiang, Blanton R D S, Thomas D E. Random forest architectures on FPGA for multiple applications[C]//Proc of the Great Lakes Symp on VLSI. New York: ACM, 2017: 415−418

[3] Zhou Zhihua. Ensemble Learning[M]//Machine learning. Singapore: Springer Singapore, 2021: 181−210

[4] Mienye I D, Sun Yanxia. A survey of ensemble learning: Concepts, algorithms, applications, and prospects[J]. IEEE Access, 2022, 10: 99129−99149

[5] Wang Hongfei, Blanton R D. Ensemble reduction via logic minimization[J]. ACM Transactions on Design Automation of Electronic Systems, 2016, 21(4): 1−17

[6] Wang Hongfei, He Kun. Improving test and diagnosis efficiency through ensemble reduction and learning[J]. ACM Transactions on Design Automation of Electronic Systems, 2019, 24(5): 1−26

[7] Wang Hongfei, Li Jianwen, He Kun. Hierarchical ensemble reduction and learning for resource-constrained computing[J]. ACM Transactions on Design Automation of Electronic Systems, 2019, 25(1): 1−21

[8] Ibragimov B, Gusev G. Learn together stop apart: An inclusive approach to ensemble pruning[C]//Proc of the 30th ACM SIGKDD Conf on Knowledge Discovery and Data Mining. New York: ACM, 2024, 1166–1176

[9] Bian Yijun, Wang Yijun, Yao Yaqiang, et al. Ensemble pruning based on objection maximization with a general distributed framework[J]. IEEE Transactions on Neural Networks and Learning Systems, 2019, 31(9): 3766−3774

[10] Huang Sen, Yang Kaixiang, Qi Sheng, et al. When large language model meets optimization[J]. Swarm and Evolutionary Computation, 2024, 90: 101663

[11] Martínez-Muñoz G, Suárez A. Pruning in ordered bagging ensembles[C]//Proc of the 23rd Int Conf on Machine Learning. Piscataway, NJ: IEEE, 2006: 609−616

[12] Guo Huaping, Liu Hongbing, Li Ran, et al. Margin & diversity based ordering ensemble pruning[J]. Neurocomputing, 2018, 275: 237−246

[13] Lu Zhenyu, Wu Xindong, Zhu Xingquan, et al. Ensemble pruning via individual contribution ordering[C]//Proc of the 16th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2010: 871−880.

[14] Lazarevic A, Obradovic Z. Effective pruning of neural network classifier ensembles[C]//Proc of the 2001 Int Joint Conf on Neural Networks. Piscataway, NJ: IEEE, 2001, 2: 796−801

[15] Onan A, Korukoğlu S, Bulut H. A hybrid ensemble pruning approach based on consensus clustering and multi-objective evolutionary algorithm for sentiment classification[J]. Information Processing & Management, 2017, 53(4): 814−833

[16] Tamon C, Xiang Jie. On the boosting pruning problem[C]//Proc of European Conf on Machine Learning. Berlin: Springer, 2000: 404−412

[17] Zhang Yi, Burer S, Nick Street W, et al. Ensemble pruning via semi-definite programming[J]. Journal of Machine Learning Research, 2006, 7(7): 1315−1338

[18] Hu Ruihan, Zhou Songbin, Liu Yisen, et al. Margin-based pareto ensemble pruning: An Ensemble pruning algorithm that learns to search optimized ensembles[J]. Computational Intelligence and Neuroscience, 2019, 2019(1): 7560872

[19] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in Neural Information Processing Systems, 2020, 33: 1877−1901

[20] Liu Aixin, Feng Bei, Xue Bing, et al. DeepSeek-V3 technical report[J]. arXiv preprint arXiv: 2412.19437, 2024

[21] Chang Kaiyan, Wang Kun, Yang Nan, et al. Data is all you need: Finetuning LLMs for chip design via an automated design-data augmentation framework[C]//Proc of the 61st ACM/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2024: 1−6

[22] Tsai Yun-Da, Liu Mingjie, Ren Haoxing. RTLFixer: Automatically fixing RTL syntax errors with large language model[C]//Proc of the 61st ACM/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2024: 1−6

[23] Wu Haoyuan, He Zhuolun, Zhang Xinyun, et al. ChatEDA: A large language model powered autonomous agent for EDA[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2024, 43(10): 3184−3197

[24] Guo Peifu, Chen Y, Tsai Y, et al. Towards optimizing with large language models[C]//Proc of the 4th Workshop on Knowledge-

infused Learning. New York: ACM, 2024: 1−11

[25] Wu Xingyu, Wu Shenghao, Wu Jibin, et al. Evolutionary computation in the era of large language model: Survey and roadmap[J]. IEEE Transactions on Evolutionary Computation, 2024, 29(2): 534−554

[26] Pluhacek M, Kazikova A, Kadavy T, et al. Leveraging large language models for the generation of novel metaheuristic optimization algorithms[C]//Proc of the Companion Conf on Genetic and Evolutionary Computation. New York: ACM, 2023: 1812−1820

[27] AhmadiTeshnizi A, Gao W, Udell M. OptiMUS: Optimization modeling using MIP solvers and large language models[J]. arXiv preprint arXiv: 2310.06116, 2023

[28] Lehman J, Gordon J, Jain S, et al. Evolution Through Large Models[M]//Handbook of Evolutionary Machine Learning. Singapore: Springer Nature Singapore, 2023: 331−366

[29] Liu Fei, Tong Xialiang, Yuan Mingxuan, et al. Evolution of heuristics: Towards efficient automatic algorithm design using large language model[C]//Proc of the 41st Int Conf on Machine Learning. New York: ICML, 2024: 1−23

[30] Ye Haoran, Wang Jiarui, Cao Zhiguang, et al. ReEvo: Large language models as hyper-heuristics with reflective evolution[C]//Proc of the 38th Int Conf on Neural Information Processing Systems. New York: Curran Associates Inc., 2024: 1−38

[31] Yang Chengrun, Wang Xuezhi, Lu Yifeng, et al. Large language models as optimizers[C]//Proc of the 12th Int Conf on Learning Representation. Washington, DC: ICLR, 2023: 1−41

[32] Liu Shengcai, Chen Caishun, Qu Xinghua, et al. Large language models as evolutionary optimizers[C]//Proc of the 2024 IEEE Congress on Evolutionary Computation. Piscataway, NJ: IEEE, 2024: 1−8

[33] Lange R, Tian Yingtao, Tang Yujin. Large language models as evolution strategies[C]//Proc of the Genetic and Evolutionary Computation Conf Companion. New York: ACM, 2024: 579−582

[34] Jiang Shuo, Xie Min, Luo Jianxi. Large language models for combinatorial optimization of design structure matrix[J]. Proceedings of the Design Society, 2025, 5: 2201−2210

[35] Liu Fei, Lin Xi, Yao Shunyu, et al. Large language model for multiobjective evolutionary optimization[C]//Proc of the 2025 Int Conf on Evolutionary Multi-Criterion Optimization. Singapore: Springer Nature Singapore, 2025: 178−191

[36] Khataei A, Bazargan K. TreeLUT: An efficient alternative to deep neural networks for inference acceleration using gradient boosted decision trees[C]//Proc of the ACM/SIGDA Int Symp on Field Programmable Gate Arrays. New York: ACM, 2025: 14−24

[37] Wang Chao, Gong Lei, Li Xi, et al. A ubiquitous machine learning accelerator with automatic parallelization on FPGA[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(10): 2346−2359

[38] Alsharari M, Mai S T, Woods R, et al. Efficient integer-only-inference of gradient boosting decision trees on low-power devices[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2024, 72(1): 241−253

[39] Martínek T, Kořenek J, Čejka T. LGBM2VHDL: Mapping of LightGBM models to FPGA[C]//Proc of the IEEE 32nd Annual Int Symp on Field-Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2024: 97−103

[40] Dinh T P, Pham-Quoc C, Thinh T N, et al. A flexible and efficient FPGA-based random forest architecture for IoT applications[J]. Internet of Things, 2023, 22: 100813

[41] Daigneault M A, David J P. Automated synthesis of streaming transfer level hardware designs[J]. ACM Transactions on Reconfigurable Technology and Systems, 2018, 11(2): 1−22

[42] Zeng Shulin, Liu Jun, Dai Guohao, et al. FlightLLM: Efficient large language model inference with a complete mapping flow on FPGAs[C]//Proc of the ACM/SIGDA Int Symp on Field Programmable Gate Arrays. New York: ACM, 2024: 223−234

[43] Liu Jun, Zeng Shulin, Ding Li, et al. FlightVGM: Efficient video generation model inference with online sparsification and hybrid precision on FPGAs[C]//Proc of the ACM/SIGDA Int Symp on Field Programmable Gate Arrays. New York: ACM, 2025: 2−13

[44] Brayton R, Mishchenko A. ABC: An academic industrial-strength verification tool[C]//Proc of Int Conf on Computer Aided Verification. Berlin: Springer, 2010: 24−40



Huangfu Jingxuan, born in 2001. PhD candidate. His main research interests include EDA and hardware security.

皇甫璟轩, 2001 年生。博士研究生。主要研究方向为 EDA、硬件安全。



Wang Hongfei, born in 1985. PhD, associate professor, PhD supervisor. His main research interests include artificial intelligence theory and algorithm design, IC test and diagnosis, EDA, and hardware security.

王虹飞, 1985 年生。博士，副教授，博士生导师。主要研究方向为人工智能理论算法设计、芯片测试与诊断、EDA、硬件安全。