CPU-Eval：面向中央处理器硬件代码生成的大语言模型评估

Yuisei Iwata*、Yutaka Masuda†和Tohru Ishihara†
*名古屋大学信息学部，日本名古屋
†名古屋大学信息学研究生院，日本名古屋 电子邮件：
yiwata@nagoya-u.jp， {masuda、ishihara}@ertl.jp

arXiv:2611.32405v1 [cs.AR] 24 Nov 2026

摘要——**VerilogEval 等基准测试推动了大语言模型在硬件描述语言（HDL）生成方面的快速进展。然而，近期的 HDL 生成系统在 VerilogEval 上的通过率已超过 90%，留给进一步展现进展的空间有限。为进一步推动这一具有实际重要性的硬件开发任务，我们推出 CPU-Eval，这是一个由六款中央处理器构建而成、包含 339 个 HDL 生成问题的基准测试。这些问题取材于实际的 CPU 设计，包括已流片制造的系列，涵盖从小型嵌入式内核到支持 Linux 的乱序内核等各类 CPU，并涉及以 SystemVerilog、Chisel 和 SpinalHDL 编写的多种 CPU 功能。每个问题都要求大语言模型执行真实的 CPU 开发任务，即生成 HDL。在七个最先进的大语言模型中，CPU-Eval 上的 pass@1 均值比 VerilogEval v2 Spec-to-RTL 低 45.14 个百分点，揭示了被 VerilogEval 高通过率掩盖的能力差异。CPU-Eval 还表明，源代码行数更多的子模块，其 pass@1 更低，这凸显了细粒度模块化设计对大语言模型辅助硬件生成的重要性。我们还发现，对于使用较不常见 HDL 编写的 CPU，pass@1 更低，因此建议设计人员在部署前确认大语言模型足以胜任其所用 HDL 的代码生成工作。**

I. 引言

VerilogEval [1], [2] 等基准测试推动了大语言模型（LLM）硬件描述语言（HDL）生成能力的快速发展。基准测试使研究人员能够评估新的大语言模型和生成方法是否能够生成在功能上更加正确的硬件描述语言；而编译和功能测试失败则为改进模型及开发生成技术提供了具体反馈。在训练阶段，VeriReason通过加入推理步骤和测试平台扩展了RTLCoder数据集，将测试平台结果用作强化学习的奖励，并使用VerilogEval表明，由此得到的模型能够生成在功能上更加正确的RTL [3], [4]。在推理阶段，RTLFixer利用VerilogEval问题中的编译器错误，迭代修复语法错误 [5],；而VerilogCoder则利用VerilogEval测试平台失败结果和波形跟踪来定位并修复功能错误 [6]。因此，基准测试为分析失败原因以及开发模型和生成方法提供了共同基础，而不仅仅是用于比较它们。

要使这一研究方向持续推进，基准测试必须能够揭示不同硬件描述语言生成方法之间可测量的差异，并提供具有挑战性的案例，以便分析和修复失败。然而，近期系统在多个 VerilogEval 问题集 [7]–[9],上的通过率已超过 90%，留下的待进一步研究的未解决案例越来越少。一些基准测试通过采用来自开源硬件设计 [10], [11] 的高难度问题来应对这一问题。例如，RealBench 要求大语言模型根据四个开源硬件 IP 生成模块，并报告称 o1-preview 的 pass@1 为 13.3%，而其在 VerilogEval v2 Spec-to-RTL (S2R) 上的成绩为 70.5% [10]。不过，现有基准测试未能充分衡量大语言模型在完整CPU中实现子模块的能力，因为它们涵盖的 CPU 设计种类有限，也没有将生成的子模块作为这些 CPU 的一部分进行评估。

为弥补这一空白，我们推出了 CPU-Eval，这是一个包含 339 题的基准测试，用于生成六种完整且实用的中央处理器中的子模块。得益于广泛的中央处理器覆盖范围，CPU-Eval 为大语言模型辅助处理器设计提供了切实可行的见解，详见第 III 节。CPU-Eval 的任务反映了设计人员的常见做法：基于现有中央处理器进行构建，并仅设计特定用途所需的额外子模块，而非从零开始设计整个中央处理器 [12], [13]。

我们的贡献如下：

1) 我们从六种实用的中央处理器实现中构建了 339 个 HDL 生成问题，其中包括已在硅上制造的 CPU 系列。这些问题涵盖的范围从小型嵌入式内核到支持 Linux 的乱序内核，涉及数据通路和复杂控制逻辑中的各类 CPU 功能。CPU-Eval[1] 涵盖三种硬件描述语言：RTL 级的 SystemVerilog，以及更高级的硬件构建语言 Chisel 和 SpinalHDL。2) 我们为大语言模型定义了一项基准测试任务，将子模块级 HDL 生成与 CPU 级功能评估相结合，且不依赖商业EDA工具。CPU-Eval 会将生成的每个子模块集成到其 CPU 中，在该 CPU 上执行测试程序，并

将中央处理器的执行跟踪与参考ISA轨迹进行比较，以检测其是否偏离预期的CPU行为。3）通过在 CPU-Eval 上评估七种最先进的大语言模型，我们揭示了 VerilogEval 高通过率所掩盖的模型能力差异；评估结果显示，均值 pass@1 比 VerilogEval v2 S2R 低 45.14 个百分点。CPU-Eval 表明，子模块的源代码行越多，pass@1 越低；而采用较少使用的硬件描述语言编写的中央处理器，其 pass@1 也更低。

[1]CPU-Eval 可通过 https://github.com/next-edge-ai-ishihara/cpu-eval 获取。

我们的发现对大语言模型辅助的硬件设计有两点启示。首先，源代码长度趋势表明，应采用更精细的模块化设计，为大语言模型提供更小的子模块。其次，不同硬件描述语言之间的差异促使设计人员使用 CPU-Eval 等实用基准测试，在部署前验证大语言模型是否足以胜任以所用硬件描述语言生成代码的任务。

II. CPU-Eval基准测试设计

*A. HDL*生成与评估示例

每个 CPU-Eval问题都将一条指定某个 CPU 子模块的提示，与对生成实现进行的功能测试配对。CPU-Eval 衡量大语言模型生成的 HDL 能否在完整CPU中正常运行；如果不能，则记录程序执行在何处偏离参考指令集架构模型。

图 1 展示了大语言模型如何解决 339 个问题之一，以及 CPU-Eval 如何对其进行评估。图 1(a) 中的提示指定了 Ibex CPU 的算术逻辑单元（ALU），大语言模型 Gemma 4 31B 则生成了图 1(b) 所示的 ibex_alu 子模块。生成的硬件描述语言在计算除法器所用结果之前，错误地丢弃了除数和被除数各自的最高位。在每个除法步骤中，Ibex 都使用该结果来决定是否减去除数并设置相应的商位，因此截断操作数可能会改变商。

CPU-Eval 将这段硬件描述语言集成到 Ibex 中，并构建一个 RTL 模型。随后，它在该中央处理器的 RTL 模型和 Spike 指令集架构模型上运行图 1(c) 所示的测试程序 busy_work.S。图 1(d) 对比了两者的执行轨迹。在执行除法指令之前，两条轨迹都显示 x28 的值为 0x77ab6f90 (2,007,723,920)，x7 的值为 0xef56df20 (−279，519，456)，因此正确的有符号商为 0xfffffff9 (−7)。随后，Spike 将该值写入 x29，而操作数截断导致 RTL 模型写入 0xfffffffa (−6)。因此，这些轨迹指出了暴露功能错误的具体指令和架构状态更新，而不仅仅是报告生成的硬件描述语言未能通过测试。研究人员可以分析这些具体故障，以改进大语言模型和硬件描述语言生成方法。

B. 规模与多样性

CPU-Eval 包含 339 个 HDL 生成问题，涉及六款 RISC-V CPU，涵盖广泛的 CPU 设计。我们从 RISC-V International 维护的 RISC-V Exchange 的 Core(s) 类别中， [14], [15], 选取了可再分发 HDL 且具有文档化验证流程的 CPU,

不使用商业EDA工具。如表 I 所总结，这些 CPU 的范围从小型嵌入式内核到支持 Linux 的乱序内核，其硬件源代码采用 SystemVerilog、Chisel 和 SpinalHDL 编写。CPU-Eval 面向用于构建实体处理器的实用 CPU 系列，而非

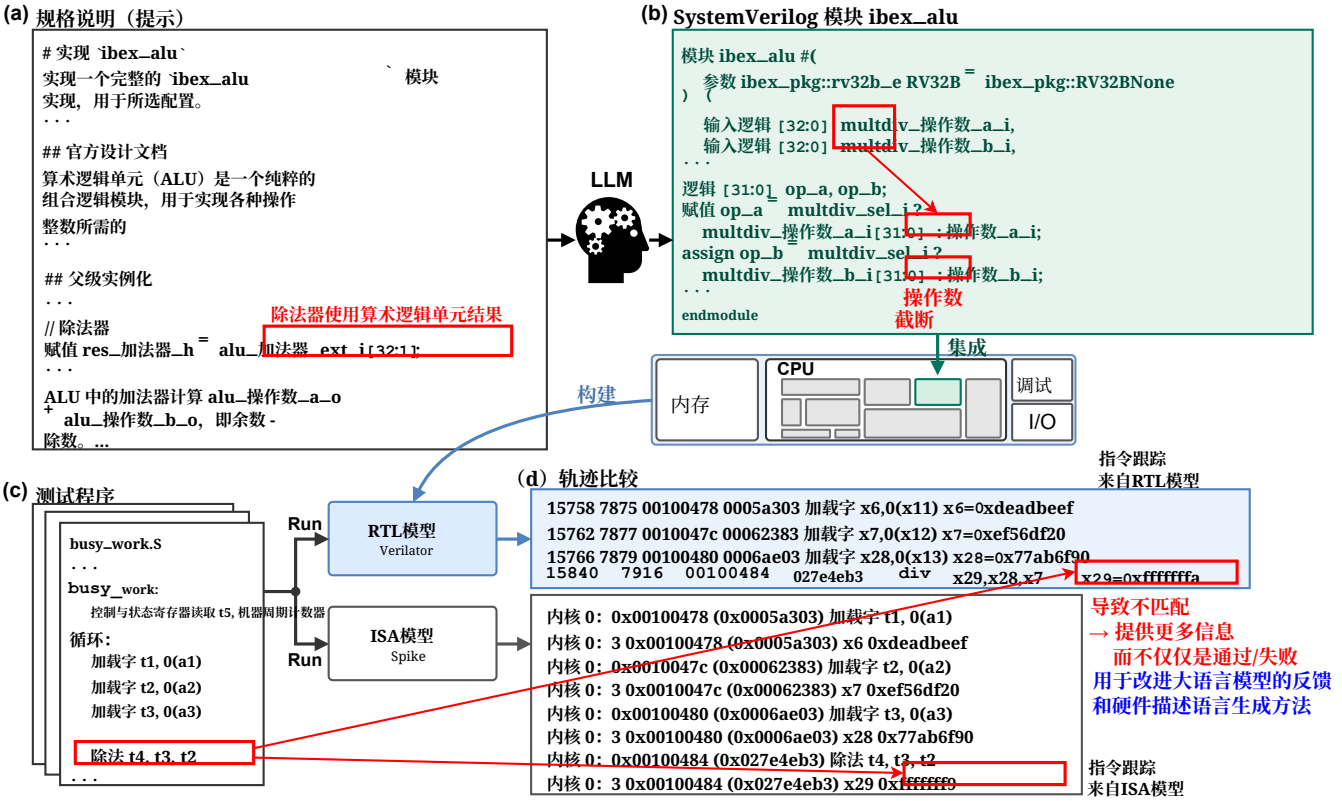

图 1。339 个 CPU-Eval 问题之一的生成与评估流程：(a) CPU-Eval 将 Ibex CPU 的算术逻辑单元（ALU）提示提供给大语言模型（LLM），(b) 大语言模型生成 ibex_alu，CPU-Eval 将其集成到 Ibex 中，(c) CPU-Eval 执行测试程序，以及 (d) 轨迹比较定位功能错误。

表 I CPU-Eval 中的六个开源 RISC-V CPU 及其特征。

CPU	描述	HDL	数量	源代码行		
			子模块	中位数	Mean	总计
BOOM	采用3路译码的64位乱序执行内核，配备96项重排序缓冲区，并支持Linux	Chisel	91	82	171.0	15,557
CVA6	采用6级流水线的32/64位顺序执行内核，并提供支持Linux的配置	SystemVerilog	93	152	368.0	34,224
Rift2Core	采用9级流水线，支持机器模式、监管者模式和用户模式的64位乱序执行内核	Chisel	48	38.5	75.2	3,610
VeeR EL2	32 位顺序执行核心，采用a4级流水线，具有分支预测功能，并支持机器模式/用户模式	SystemVerilog	71	122	328.2	23,299
VexRiscv	针对现场可编程门阵列优化的 32 位顺序执行核心，可选配缓存、内存管理单元并支持 Linux	SpinalHDL	16	170	275.0	4,400
Ibex	采用两级流水线的 32 位顺序执行核心，适用于嵌入式控制应用	SystemVerilog	20	679.5	816.8	16,336
总体			339	106	287.4	97,426

表II BOOM 子模块的 91 个 HDL 生成问题及其源代码行数。INSN. 表示执行轨迹之间比较的指令数量。† 表示在这些轨迹中未记录的额外检查。

问题	行 指令 问题	行 指令 问题	行 指令 问题	行 指令 问题	行 指令
算术逻辑单元 5014.6k 算术逻辑单元 2254.24k 算术逻辑单元 18618.1k 原子内存操作算术逻辑单元 563.62k 仲裁器 173.58k 基本分发器 195.48k	内存单元	取消队列 88 2.26k 分支掩码生成逻辑 763.17k† 分支预测器 2783.17k 断点单元 46 †	取指缓冲区 160 595 取指目标队列 2681.27k† 融合乘加译码器 31 1.27k 旁路年龄逻辑 44 243 浮点流水线 249 1.36k	循环分支预测器列 141 † 加载存储单元 1451 4.45k 最大周期斐波那契LBSK 48 †内存地址计算单元 736.2k†	队列 892.03k† RecFN到RecFN转换器 46 536 寄存器文件可综合 74 6.29k 寄存器读取 186 6.29k 寄存器读取解码 51 7.77k
BIM分支预测器组 106 3.17k†	BTB分支预测器组 176 2.13k†	浮点到浮点转换 59 1.34k	54位乘法器 43 274 乘加递归浮点流水线 621.27k		
Boom核心 14304.45k Boom复制数据阵列 23 1.21k	BundleBridgeNe xus 37 †	浮点到整数转换 73 4.03k	乘加递归浮点至原始_后乘法 126 652		重命名忙碌表 48 6.29k 重命名空闲列表 781.57k†
Boom前端模块 6702.98k Boom输入输出未命中状态保持寄存器 821.53k Boom未命中状态保持寄存器 3547.77k Boom未命中状态保持寄存器文件 2657.77k Boom非阻塞数据缓存模块 50211.6k Boom探测单元 112 †	CompareRecFN 41 700	FPU 81 2.31k	乘加递归浮点至原始_前乘法 96 652		重命名映射表 87 6.66k 重命名阶段 1951.57k†
		浮点执行单元 136 1.75k	乘除法单元 1444.78k		重排序缓冲区 6672.22k†
		浮点融合乘加流水线 36 1.27k	NBDTLB 341 †空局部分支预测器组 5 †空预取器 5 †流水线乘法器 272.94k		RoundAnyRawFNToRecFN 0 342 30 RoundRawFNToRecFN 326 300 RVC扩展器 15 391 TAGE分支预测器组 1631.11k† TAGE表 1601.11k†
		浮点单元 21 641 HellaCacheArbi ter 79 缓存模块 250 † 整数到浮点转换 44 1.96k 整数到浮点转换单元 45 536 整数交叉开关 18 † 发射槽 227 1.39k IssueUnitCol			
		ComposedBranch PredictorBank 35 † 控制状态寄存器文件 1212 1.64k 译码单元 128 8.45k DivSqrtRecF64 42 274 DivSqrtRecF64_mulAddZ31 66 274 DivSqrtRecF64ToRaw_mulAd664 274 dZ31			
BoomRAS 22 †	除法单元 25 469	循环中 100 †	流水线乘法单元 18 2.94k		TLB 341 †
Boom瓦片模块 Imp 98 2.42k	FAMicroBTBBranchPredicto 131 rBank †	L1元数据阵列 26 †	物理内存保护检查器 48 †预测重命名阶段 49 290† PTW 476		TL交叉开关 47 †微操作码浮点除法译码器 或 38 535 微操作码浮点运算单元译码器 91 6.64k
Boom写回单元 120 † 分支译码 913.17k†	浮点除法平方根单元 145 535	分支 循环分支预测器组 195 †			

仅作为基准测试练习创建的设计。例如，Ibex 已完成多次流片 [16]，CVA6 的前身 Ariane 采用 22 纳米全耗尽绝缘体上硅工艺制造 [17]，基于 BOOM 的内核采用 28 纳米互补金属氧化物半导体工艺制造 [18]，VexRiscv 则用于完成流片的 Baochip 1x [19]。CPU-Eval 公开可用，使研究人员无需访问商业EDA工具即可评估和开发 HDL 生成方法。

对于每个中央处理器，我们枚举了所选配置中构成该中央处理器的所有子模块。表II以一个示例列出了全部 91 个 BOOM 问题，展示了这些子模块所覆盖的中央处理器功能范围之广，包括算术单元、指令解码、流水线控制、旁路、分支预测、缓存和内存管理。

表 I报告了每个中央处理器的每个子模块的源代码行数的中位数和均值，以及所有子模块的总计。在表II中，行数列报告了每个 BOOM 子模块的源代码长度。指令数列报告了在各测试程序中 RTL 与 ISA模型之间进行比较的已退役指令数量，其中“k”表示 1,000。

匕首符号标记了 CPU-Eval 会检查未记录在执行跟踪中的 CPU行为的问题。例如，LoopBranchPredictorBank 的测试程序会将其测得的周期数写入 x20，CPU-Eval 再将其与预期值进行比较。页表遍历器（PTW）在 TLB 未命中后遍历页表，它通过监管者级程序进行评估，这些程序会检查脏位处理以及页面重新映射后的指令取指。BoomRAS 的定向测试会将返回地址写入栈中，按索引读取这些地址，并将每次读取的_addr输出与预期地址进行比较。这些额外检查使 CPU-Eval 能够涵盖全部 91 个 BOOM 子模块，包括分支预测和虚拟内存转换等重要功能，而这些功能无法仅通过已提交指令轨迹进行评估。

C. 与现有基准测试的比较机制

表 III 从问题数量、所包含的 CPU 设计、生成任务和评估方法等方面，将 CPU-Eval 与现有基准测试进行了比较。不同于 VerilogEval v2 [2] 中源自教学练习的独立模块，以及 RTLLM 2.0 [21]，中手工制作的独立设计，CPU-Eval 基于完整且实用的 CPU 实现中的子模块构建问题，其中包括在硅上制造的 CPU 系

表 III
问题数量 ，包括 D CPU 设计、任务以及 CPU-Eval 和相关硬件代码基准测试中的评估流程 S.

基准测试	问题数量	包含的 CPU 设计	大语言模型的功能	基准测试如何评估输出
VerilogEval v2 [2]	156	无。小型独立模块 来自教学练习	根据一份 规格说明	使用提供的测试平台进行仿真，适用于 每个问题
RTLLM [20]	30	一个包含 518 行的简化版 RISC CPU	为一个设计生成全部 RTL 根据规格说明	使用为每个问题提供的测试平台进行仿真 每个问题
RTLLM 2.0 [21]	50	小型独立模块，例如 一个 32 位 MIPS 算术逻辑单元	根据一份规格说明生成一个 RTL 模块 规格说明	使用为每个问题提供的测试平台进行仿真 每个问题
RealBench [10]	64	一个 32 位嵌入式 RISC-V CPU	生成一个模块或完整的 设计，基于规格说明	使用测试平台仿真和形式 等价性检查
ChipBench [11]	265	来自开源项目的九个模块 CPUs	生成或调试 Verilog、 Python、SystemC 或 CXXRTL	将生成的输出行为与 黄金 Verilog 实现进行比较
ChipVerilog [22]	64	两个小型 RISC 内核：OR1200 和一个五级 MIPS-16 内核	根据规格说明生成一个模块或整个 IP/内核	使用测试平台仿真或形式 等价性检查
HWE-Bench [23]	417	任务中的五个 CPU：Ibex、CVA6、 Rocket、VeeR EL2 和香山	修复硬件代码仓库中报告的 缺陷	应用生成的补丁并运行 特定任务测试
CPU-Eval	**339**	**个中央处理器，从嵌入式到 支持 Linux 的乱序内核**	**生成一个中央处理器子模块 根据规格说明**	**将生成的子模块集成到 CPU 中，并在 CPU 上执行程序**

列。CPU-Eval 的问题集规模是 VerilogEvalv2 S2R 的两倍以上，并且其中问题所涉及的子模块包含的源代码行数显著更多：中位数为 106 行，均值为 287 行，而 VerilogEvalv2 S2R 分别为 16 行和 19 行。

RTLLM、RealBench、ChipBench 和 ChipVerilog 都包含 CPU 设计，但它们的任务要求 LLM 生成一个完整的小型 CPU 或单个模块，并且其评估不会将生成的子模块集成到完整CPU中，也不会在其上执行程序 [10], [11], [20]–[22]。CPU-Eval 则要求 LLM 生成实用 CPU 的一个子模块，并涵盖六种 CPU，范围从小型嵌入式内核到支持 Linux 的乱序内核。

CPU-Eval 和 HWE-Bench 都涵盖实用的 CPU 设计，但二者的任务不同：HWE-Bench 要求大语言模型智能体修复 GitHub 仓库中报告的缺陷 [23], ，而 CPU-Eval 要求 LLM 根据子模块的规格说明为其生成硬件描述语言（HDL）。

D. 问题质量检查

仅凭较低的通过率，无法证明 HDL 生成问题很难，因为不完整的提示和有缺陷的测试也可能导致生成的 HDL 失败。例如，RTL-BenchMT 发现，VerilogEval 的某些提示遗漏了测试所要求的寄存器初始化 [24]。STG 发现，有缺陷的 VerilogEval 测试平台可能会拒绝正确的 HDL [25]。因此，在将 LLM 失败解释为问题难度的证据之前，我们会检查每个 CPU-Eval 提示是否缺少要求，并检查每项测试是否会给出错误的判定。

1）检查提示中的缺失规格说明：为防止规格说明不完整，我们通过检查子模块的源代码以及中央处理器与其连接的位置，枚举每个问题中子模块的接口和外部要求的行为。随后，我们将每一项内容对应到设计文档、指令集与接口规范或周边硬件描述语言中的逐字摘录。最终生成的提示包含人类开发者实现该子模块时所需的设计信息。

2）检查测试结论：作为功能评估的可靠起点，CPU-Eval 使用中央处理器开发者为验证其完整CPU而运行的测试程序。对于 CVA6，CPU-Eval 使用 RISC-V架构测试，这些测试用于检查 RISC-V 规格说明的实现，并且 CVA6 开发者将其纳入回归测试 [26]–[27]。VeeR EL2 的评估使用由 Linux基金会托管的 CHIPS 联盟维护、并由中央处理器开发者在持续集成中运行的 riscv-dv 回归测试 [28]–[30]。Rift2Core 和 BOOM 使用两个项目共同采用的 riscv-tests 处理器测试套件进行测试 [31]–[33]。VexRiscv 的测试程序来自该项目的 TestIndividualFeatures回归测试套件 [34]。对于 Ibex，CPU-Eval 使用简单的_系统程序，这些程序由 Ibex 项目在持续集成中运行 [16]。

我们不假设，复用这些测试程序就能保证对每个 CPU-Eval 问题都给出正确判定。在纳入某个问题之前，我们首先确认其测试能够接受 CPU 仓库中该子模块的源代码。随后，我们准备多个硬件描述语言实现，每个实现都包含不同的功能错误，但仍能成功构建并完成仿真。由于这些实现的执行结果与预期的 CPU行为不同，测试必须拒绝每个实现。如果某个错误实现通过了测试，我们会判断是程序未能触发该错误，还是比较机制遗漏了它的影响，并据此强化测试程序或比较机制。每次修改后，我们都会重复这些检查。

例如，在 CVA6 store_unit 问题中，一个为每次存储都写入零的实现最初通过了测试，因为比较机制记录了指令和寄存器更新，却遗漏了内存更新。因此，我们扩展了比较机制，纳入内存写入信息，例如地址和数据值。扩展后的比较机制会拒绝那个为每次存储都写入零的实现，以及另一个抑制所有内存写入的实现，同时仍接受 CVA6 代码库中的 store_unit 源代码。通过这种方式，我们提高了 CPU-Eval 测试的可靠性。

表 IV $k \in \{1, 10\}$ 在 CPU-Eval 总体、各中央处理器以及 VerilogEval v2 S2R 上的 PASS@k (%)。最后一行中的带符号差值为 CPU-EVAL 减去 VerilogEval v2，单位为百分点。

	Gemma 4 31B		gpt-oss-120B		Qwen3.5-397B		DeepSeek-V4.1-Flash		Kimi K2.7 Code		GPT-5.6 Sol		Claude Opus 5		均值（7 个大语言模型）	
问题集	$k=1$	$k=10$	$k=1$	$k=10$	$k=1$	$k=10$	$k=1$	$k=10$	$k=1$	$k=10$	$k=1$	$k=10$	$k=1$	$k=10$	$k=1$	$k=10$
VerilogEval v2	69.23	76.92	69.55	83.97	69.17	82.05	74.74	85.26	69.81	84.62	90.19	94.23	92.69	95.51	76.48	86.08
CVA6	32.04	45.16	29.78	45.16	25.59	43.01	34.73	56.99	36.13	55.91	56.34	69.89	73.87	90.32	41.21	58.06
Ibex	24.50	30.00	22.50	30.00	16.50	30.00	26.00	40.00	25.00	35.00	47.50	60.00	68.00	90.00	32.86	45.00
VeeR EL2	37.32	57.75	38.87	57.75	35.77	50.70	37.89	63.38	39.72	60.56	62.39	78.87	69.72	85.92	45.96	64.99
Rift2Core	9.58	16.67	8.33	27.08	10.83	18.75	18.33	37.50	18.96	39.58	52.08	70.83	56.04	79.17	24.88	41.37
VexRiscv	0.00	0.00	1.25	6.25	0.00	0.00	2.50	12.50	5.00	6.25	10.63	18.75	14.37	56.25	4.82	14.29
BOOM	2.75	6.59	5.93	16.48	6.26	14.29	10.66	20.88	9.67	21.98	30.77	49.45	57.25	78.02	17.61	29.67
所有 CPU-Eval	20.15	30.38	20.47	34.81	18.70	30.68	24.57	42.77	25.22	41.89	47.46	63.42	62.86	82.89	31.35	46.69
差值（百分点）	−49.08	−46.54	−49.08	−49.17	−50.46	−51.37	−50.17	−42.48	−44.59	−42.73	−42.73	−30.81	−29.83	−12.62	**−45.14**	−39.39

III. 实验

A. 实验设置

为了发掘大语言模型辅助处理器设计的实际见解，我们在 CPU-Eval 的 339 个问题和 VerilogEval v2 S2R 上评估七个最先进的大语言模型。我们评估了由 Ollama Cloud 提供服务的五个开放权重大语言模型：Gemma 4 31B、gpt-oss-120B、Qwen3.5-397B、DeepSeek-V4.1-Flash 和 Kimi K2.7 Code。我们还评估了两个旗舰级商用大语言模型：GPT-5.6 Sol 和 Claude Opus 5。对于每个问题，我们向每个大语言模型提供提示，并独立采样 10 个硬件描述语言实现。对于五个 Ollama Cloud 模型，我们将温度设为 0.8，将 top-p设为 0.95，与 VerilogEval 的采样设置 [1]保持一致。对于这五个模型，我们设置了 65,536 个令牌的上限；该上限足以生成每个 CPU-Eval 问题所需的硬件描述语言，同时还能停止无限重复输出的缺陷生成结果。我们将 gpt-oss-120B 的思考模式设为 low，将另外四个开放权重模型的思考模式设为 false，并将 GPT-5.6 Sol 和 Claude Opus 5 的推理强度设为 low。

我们报告了 $k \in \{1, 10\}$ 的 pass@k 。pass@k 衡量的是：对于一个问题， k 个生成的硬件描述语言实现中至少有一个通过的概率。遵循 Chen 等人 [35] 的做法，VerilogEval [1], 我们将针对每个问题生成的 $n = 10$ 硬件描述语言实现中通过测试的结果估计为 pass@k ，其定义如下

$$\text{pass@}k = \frac{1}{|P|} \sum_{p \in P} \left(1 - \frac{\binom{n-c_p}{k}}{\binom{n}{k}} \right),$$

其中， P 为问题集， c_p 为针对问题 p生成的且通过测试的硬件描述语言实现数量。

B. CPU-Eval 揭示的能力差异

如表 IV 所示，七个大语言模型在 CPU-Eval 上的均值 pass@1 比在 VerilogEval v2 S2R 上低 45.14 个百分点。Claude Opus 5 在 VerilogEvalv2 S2R 上的 pass@1 达到 92.69%，但在 CPU-Eval 上仅为 62.86%，这表明在 VerilogEval 上取得较高通过率，并不意味着具备同等能力，能够为实际 CPU 中的子模块生成硬件描述语言。如下文所述，CPU-Eval 揭示了被 VerilogEval 的高通过率所掩盖的模式。

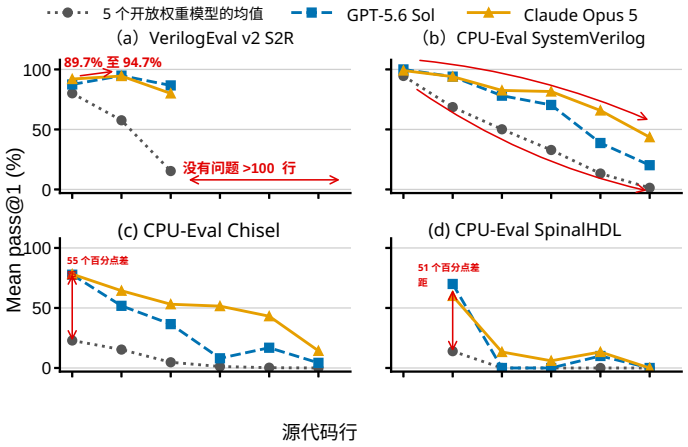

图2。按源代码长度划分的 pass@1 均值。（a）展示了 VerilogEval v2 S2R 上两个商业模型的均值几乎达到饱和。（b）说明随着 CPU-Eval 中 SystemVerilog 子模块变长，各模型的 pass@1 会降低。（c）和（d）突出展示了五个开放权重模型与两个商业模型在 Chisel 和 SpinalHDL 子模块上的 pass@1 均值差异。

1）对源代码长度的敏感性：VerilogEval v2 S2R 上接近饱和的分数，掩盖了 pass@1 随源代码长度变化的情况。如图2(a)所示，尽管源代码长度有所增加，两个商业模型的均值 pass@1 仍从 1–20 行时的 89.74% 上升到 21–50 行时的 94.66%。这两个分数都已接近饱和，而在 1–20 行范围内剩余的大多数失败，源于第 II-D 节所述类型的有缺陷的提示或测试。因此，这种表面上的反转并不能充分反映模型对源代码长度的敏感性。

图2(b)和(c)显示，随着源代码长度增加，CPU-Eval 的 SystemVerilog 和 Chisel 子模块的均值 pass@1 总体呈下降趋势。图2(b)中超过500行的范围包含 CPU-Eval 最长的子模块，共3,176行；相比之下，VerilogEval v2 S2R 问题所提供的最长解决方案仅包含78行。在两个面板中，每个系列在其最长的非空范围内的数值都显著低于其最短的非空范围。

图2(b)显示，随着 SystemVerilog

子模块的源代码行数增加，不同模型的 pass@1 降幅各不相同。五个开放权重模型的均值从 1–20 行范围内的 94.55% 几乎跌至 500 行以上时的 1.38%，下降了 93.17 个百分点。GPT-5.6 Sol 和 Claude Opus 5 的降幅较小，分别为 79.79 和 55.55 个百分点，在 500 行以上时仍分别保持 20.21% 和 43.54% 的 pass@1。由此，CPU-Eval 不仅能区分模型的整体 pass@1，还能区分它们生成源代码行数更多的子模块的能力。

ChipVerilog 报告称，随着源代码长度增加，大语言模型在多个硬件设计的模块（包括两款小型中央处理器 [22]）中生成符合功能要求的硬件描述语言代码的概率会降低。CPU-Eval 在仅取自六个完整CPU的子模块中也观察到了相同的总体趋势，并进一步揭示，这种下降程度因大语言模型和硬件描述语言而异。

这种对源代码长度的敏感性对 LLM辅助硬件设计具有重要意义。模块化设计通过将复杂系统分解为较小的组件，使人类开发者能够更轻松地管理这些系统，并能独立开发和测试各个组件 [36]。我们的结果表明，细粒度模块化设计中的较小子模块，同样可以让大语言模型更轻松地完成硬件生成任务。

2) 硬件描述语言之间的差异：图 2(b)–(d) 显示，即使按源代码长度对各子模块分组，不同硬件描述语言编写的中央处理器之间仍存在显著差异。在三种硬件描述语言共有的每个源代码长度区间内，七种模型的均值 pass@1 在 SystemVerilog 中最高，在 Chisel 和 SpinalHDL 中则显著较低。例如，在包含 51–100 行代码的子模块中，两种商业模型在 SystemVerilog 中的平均得分为 80.31%，在 Chisel 中为 44.83%，在 SpinalHDL 中为 6.67%。因此，在 SystemVerilog 子模块上的成功表现，并不能代表模型为采用 Chisel 或 SpinalHDL 编写的中央处理器生成子模块的能力。

图 2(c) 和 (d) 还表明，这种难度在不同模型之间存在显著差异。对于包含 1–20 行的 Chisel 子模块，两个商业模型比五个开放权重模型高出 55 个百分点。对于包含 21–50 行的 SpinalHDL 子模块，差距为 51 个百分点。

一种假设是，模型能够从有关某种硬件描述语言的公共信息中受益。截至 2026 年 9 月 19 日，Stack Overflow 这一面向程序员的公共问答网站，
包含 3,672 个标记为 SystemVerilog、695 个标记为 Chisel 以及 13 个标记为 SpinalHDL 的问题。这些问题数量与图 2(b)–(d) 中三种硬件描述语言的总体均值 pass@1 排名顺序一致。这些结果表明，使用范围较小的硬件描述语言会对基于大语言模型的硬件设计构成采用障碍。大语言模型或许能够克服这一障碍：商业模型在 Chisel 和 SpinalHDL 上的表现均显著优于开放权重模型。

3) 使用相同硬件描述语言编写的中央处理器之间的差异：图 3 对比了使用 Chisel 编写的两个乱序中央处理器 BOOM 和 Rift2Core。图 3(a) 显示，与 Rift2Core 相比，BOOM 中长度超过 50 行的问题占比更高，而包含 1–20 行的问题占比更低。对于图 3(b) 中的五个

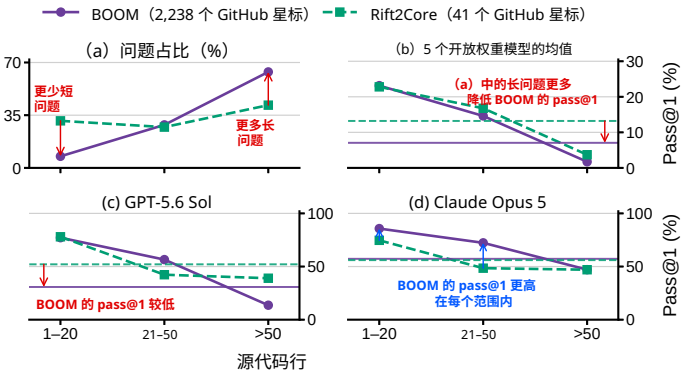

图 3. BOOM 和 Rift2Core 按源代码长度划分的结果。(a) 显示各范围内问题所占的比例。(b)–(d) 显示各范围内的均值 pass@1，水平线表示每个中央处理器所有问题的总体 pass@1。

开放权重模型，在每个源代码长度范围内，两个中央处理器的差异最多为 3 个百分点，但 BOOM 的总体 pass@1 低 6.15 个百分点。因此，BOOM 中较长问题的占比较高，而这类问题的 pass@1 较低，从而导致了这一总体差异。

图 3(c) 和 (d) 展现了两种商业模型的不同模式。与五种开放权重模型一样，GPT-5.6 Sol 在 BOOM 上的总体 pass@1 低于在 Rift2Core 上的表现。Claude Opus 5 则有所不同：它在每个源代码长度范围内于 BOOM 上的 pass@1 都更高，并且在两个中央处理器上的总体 pass@1 几乎相等。截至 2026年9月19日，BOOM 拥有 2,238 个 GitHub 星标和 529 个复刻，而 Rift2Core 拥有 41 个星标和 14 个复刻。因此，Claude Opus 5 可能接触过更多 BOOM 特有的代码，这或许可以解释其在每个范围内于 BOOM 上获得更高 pass@1 的原因。

IV. 结论

本文介绍了 CPU-Eval。该基准测试从六种实用 CPU 中提取了 339 个 HDL 生成问题，将每个生成的子模块集成到相应的 CPU 中并执行软件，以此进行评估。在七个大语言模型中，CPU-Eval 上的均值 pass@1 比 VerilogEval v2 S2R 低 45.14 个百分点，揭示了 VerilogEval 的高通过率所掩盖的能力差异。我们的评估显示，随着子模块中源代码行数量的增加，pass@1 会随之下降，这促使我们采用更小的子模块作为大语言模型的生成单元。对不同硬件描述语言的比较还显示，使用较不广泛的硬件描述语言编写的 CPU，其 pass@1 更低。这一结果凸显了 CPU-Eval 等实用的硬件代码生成基准测试的必要性：在部署大语言模型之前，应评估其使用硬件描述语言设计者所采用的硬件描述语言生成代码的能力。

致谢

研究中使用了 OpenAI Codex 和 Anthropic Claude，协助构建基准测试、开展实验并编辑稿件。作者审阅并核实了所有经 AI 辅助生成的内容。本研究部分由JST 下一代边缘人工智能半导体研究与开发计划资助，资助编号为 JPMJES2511。

参考文献

[1] M. Liu、N. Pinckney、B. Khailany 和 H. Ren，“特邀论文：Verilo- gEval：评估用于Verilog代码生成的大语言模型”，载于*IEEE/ACM*国际计算机辅助设计会议（*ICCAD*）论文集，2023年，第1–8页。[2] N. Pinckney、C. Batten、M. Liu、H. Ren 和 B. Khailany，“重新审视VerilogEval：硬件代码生成大语言模型的一年改进”，*ACM*电子系统设计自动化汇刊，第30卷，第6期，第91:1–91:20页，2025年。[3] 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，第44卷，第4期，第1448–1461页，2025年。[4] Y. Wang、G. Sun、W. Ye、G. Qu 和 A. Li，“VeriReason：利用测试平台反馈进行强化学习，增强Verilog生成的推理能力”，载于大湖区超大规模集成电路研讨会（*GLSVLSI*）论文集，2026年，第948–953页。[5] Y. Tsai、M. Liu 和 H. Ren，“RTLFixer：使用大语言模型自动修复 RTL 语法错误 ”，载于*ACM/IEEE* 设计自动化会议（*DAC*）论文集，2024年，第1–6页。[6] C.-T. Ho、H. Ren 和 B. Khailany，“VerilogCoder：采用基于图的规划和抽象语法树（AST）波形追踪工具的自主Verilog编码智能体”，*AAAI*人工智能会议论文集，第39卷，第1期，第300–307页，2025年。[7] 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*ACM/IEEE* 设计自动化会议（*DAC*）论文集，2025年，第1–7页。[8] A. Abdollahi、S. Shokoufa、N. Ashrafi、M. Kamal 和 M. Pedram，“HDLFORGE：一种采用自适应模型升级的两阶段多智能体框架，可高效生成Verilog代码”，arXiv:2603 04646，2026年。。[9] X. Kong、T. Tabassum、M. Abdelwahab 和 H. Zheng，“使用大语言模型生成可综合RTL设计的渐进式方法”，arXiv:2607.18519，2026年。[10] P. Jin、D. Huang、C. Li、S. Cheng、Y. Zhao、X. Zheng、J. Zhu、S. Xing、B. Dou、R. Zhang、Z. Du、Q. Guo 和 X. Hu，“RealBench：使用真实世界IP设计对Verilog生成模型进行基准测试”，arXiv:2507 16200，2025年。。[11] Z. Yu、C. Zhou、Y. Lin、H. Zhang、H. Ye、J. Cui、Z. Pan、J. Zhao 和 Y. Ding，“ChipBench：用于评估大语言模型在人工智能辅助芯片设计中性能的下一代基准测试”，arXiv:2601.21448，2026年。[12] K. Atasu、L. Pozzi 和 P. Ienne，“微体系结构约束下的自动化专用指令集扩展”，国际并行编程期刊，第31卷，第6期，第411–428页，2003年12月。[13] P. Yu 和 T. Mitra，“面向指令集可扩展处理器的可扩展自定义指令识别”，载于嵌入式系统编译器、体系结构与综合国际会议（*CASES*）论文集，2004年，第69–78页。[14] RISC-V International，“RISC-V Exchange：Core档案”，无日期，访问日期：2026-09-21。 [在线]。网址：https://riscv.org/blog/exchange categor y/core/ [15] ——，“RISC-V Exchange：Cores档案”，无日期，访问日期：2026-09-21。[在线]。网址：https://riscv.org/blog/exchange category/cores/ [16] lowRISC贡献者，“Ibex RISC-V核心”，2026年。 [在线]。网址：– h ttps://github.co – m/lowRISC/ibex/tree/405c6d1d8220a18b2f9196141167a 5875422dee4 [17] F. Zaruba 和 L. Benini，“应用级处理的成本：22纳米全耗尽绝缘体上硅技术中支持Linux的1.7 GHz 64位RISC-V核心的能耗与性能分析”，*IEEE*超大规模集成系统汇刊，第27卷，第11期，第2629–2640页，2019年。

[18] C. Celio、P.-F. Chiu、K. Asanovi´c、B. Nikoli´c 和 D. Patterson，“BROOM：一种开源乱序处理器，支持在 28 纳米互补金属氧化物半导体中进行具有弹性的低电压运行”，IEEE Micro，第 39 卷第 2 期，第 52–60 页，2019 年。[19] Baochip 贡献者，“Baochip 1x 芯片”，https://github.com/baochip / baochip-1x，未注明日期，访问日期：2026-09-15。[20] Y. Lu、S. Liu、Q. Zhang 和 Z. Xie，“RTLLM：一种用于大语言模型设计 RTL 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）论文集，2024 年，第 722–727 页。[21] S. Liu、Y. Lu、W. Fang、M. Li 和 Z. Xie，“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 IEEE/ACM 国际计算机辅助设计会议（ICCAD）论文集，2024 年，第 1–9 页。[22] Y. Tan、J. Du、X. Meng 和 Y. Lyu，“ChipVerilog：一种源自 OpenCores 的大规模大语言模型 Verilog RTL 生成基准测试”，arXiv:2607.13079，2026 年，发表于 ACM/IEEE 机器学习辅助 CAD 国际研讨会（MLCAD）。[23] F. Cui、H. Hou、Z. Luo、C. Yin 和 Y. Liang，“HWE-Bench：对大语言模型智能体执行现实世界硬件错误修复任务的基准测试”，arXiv:2604.14709，2026 年。[24] J. Wang、S. Liu、H. Zhou 和 Z. Xie，“RTL-BenchMT：通过智能体辅助分析和修订动态维护 RTL 生成基准测试”，载于 ACM/IEEE 设计自动化会议（DAC）论文集，2026 年，第 1–7 页。[25] E.-M. Huang、Y.-H. Kao、R.-H. Deng、W.-P. Hsin、Y.-T. Hsieh、C. Liang、H.-Y. Tsou、M.-C. Chen、Y.-K. Hung、S.-C. Ho、P.-H. Huang、S.-H. Hung 和 H. T. Kung，“面向大语言模型驱动 HDL 设计的结构化测试平台生成与面向验证的数据整理”，arXiv:2606.12983，2026 年。[26] RISC-V International，“riscv-arch-test：RISC-V 架构测试框架”，2023 年。 [在线]。可访问：https://github.com/riscv/ris cv-arch-test/tree/a5a49fc9f244192649e57fe61b4513d9bc39b1e3[27] OpenHW 基金会，“CVA6 RISC-V CPU”，2026 年。 [在线]。可访问：https://github.com/openhwfoundation/cva6/tree/6348e9e684678f83b40d0520028513d06b36e7b2[28] CHIPS 联盟，“我们是谁”，未注明日期，访问日期：2026-09-21。 [在线]。可访问：https://www.chipsalliance.org/about/who-we-are/ [29] ——，“riscv-dv：用于 RISC-V 处理器验证的随机指令生成器 ”，未注明日期，访问日期：2026-09-21。 [在线]。可访问：https: //github.com/chipsalliance/riscv-dv [30] ——，“Cores-VeeR-EL2”，2026 年。 [在线]。可访问：https://github.com / chipsalliance/Cores-VeeR-EL2/tree/aefefae20b77fa61ba306bff36a72f42ae4cc028[31] RISC-V 软件，“riscv-tests：RISC-V 处理器的单元测试”，未注明日期，访问日期：2026-09-21。 [在线]。可访问：https://github.com/riscv-software-src/riscv-tests[32] R. Lee，“Rift2Core”，2024 年。 [在线]。可访问：https://github.com/whu tddk/Rift2Core/tree/64b660fb70e0cab491049bdeea87c27936dcbf1e[33] RISC-V BOOM 开发者，“伯克利乱序机器”，2022 年。[在线]。可访问：https://github.com/riscv-boom/riscv-boom/tree/efe2b35cc2224edf3a417381d437cfdcb37e02a8[34] SpinalHDL 贡献者，“VexRiscv”，2026 年。 [在线]。可访问：https: // github.com/SpinalHDL/VexRiscv/tree/c4b2a55b22f46afb760446cd69f3ca7c36eef778[35] M. Chen 等人，“评估经过代码训练的大语言模型”，arXiv:2107.03374，2021 年。[36] A. Kravchuk-Kirilyuk、F. Graciolli 和 N. Amin，“模块化命令式范式：重新思考面向可维护软件的大语言模型”，载于《第一届 ACM SIGPLAN 语言模型与编程语言国际研讨会》论文集，2025 年，第 106–111 页。