arXiv:2603.08745v1 [cs.AR] 4 Mar 2026

# ChatNeuroSim：一种用于自动化存内计算加速器部署与优化的大语言模型智能体框架

MING-YEN LEE，佐治亚理工学院，美国
SHIMENG YU，佐治亚理工学院，美国

存算一体（CIM）架构通过降低存储器与计算单元之间的数据传输开销，已被广泛研究用于深度神经网络（DNN）加速。在传统存算一体设计流程中，系统级存算一体模拟器（如 NeuroSim）被用于跨不同硬件配置与深度神经网络工作负载开展设计空间探索（DSE）。然而，存算一体设计人员需要投入大量精力来解读模拟器手册并理解复杂的参数依赖关系。此外，通常需要进行大量设计–仿真迭代，才能在硬件约束下确定最优 CIM 配置。这些挑战严重延长了设计空间探索周期，并阻碍了存内计算的快速部署。为应对这些挑战，本文提出 ChatNeuroSim，一种基于大语言模型（LLM）的智能体框架，用于自动化存内计算加速器部署与优化。ChatNeuroSim 可自动化整个存算一体工作流，包括任务调度、请求解析与调整、参数依赖检查、脚本生成以及仿真执行。它还集成了所提出的、采用设计空间剪枝的 CIM优化器，从而能够针对不同深度神经网络工作负载快速确定最优配置。ChatNeuroSim 在大量请求级测试平台上进行了评估，并展现出正确的仿真与优化行为，验证了其在自动请求解析和任务执行方面的有效性。此外，与无剪枝基线相比，所提出的设计空间剪枝技术加快了 CIM 优化过程。在 22纳米工艺下优化 Swin Transformer Tiny 的案例研究中，所提出的 CIM优化器相较于未采用设计空间剪枝的同一优化算法，实现了 0.42×–0.79× 的平均运行时间缩短。

CCS 概念： • **硬件** → **方法学 E** **DA**.

附加关键词与短语：存内计算、多智能体、优化、设计空间探索、大语言模型

## 1 引言

为满足现代人工智能工作负载的性能和能效要求，采用脉动阵列以及数字/模拟存算一体架构的专用硬件加速器日益受到需求 [20, 22, 33]。为缩短产品上市时间并降低设计成本，系统级脉动阵列人工智能硬件模拟器 [29], [27], 以及数字/模拟存内计算加速器模拟器 [1, 16, 30] 已被用于在芯片流片前高效开展设计空间探索 [24]。

尽管这些模拟器效果显著，但也带来了诸多可用性挑战。设计人员需要投入大量精力，通过广泛查阅手册和文档，了解模拟器功能、复杂的参数依赖关系以及异构的输入–输出格式。将高层设计意图转换为有效的模拟器参数、调用仿真并解析输出结果，通常需要谨慎的人工干预。如图 1 所示，传统的存内计算设计空间探索工作流需要耗费大量时间来了解模拟器行为（步骤1）、手动将请求转换为参数并执行仿真（步骤2），以及对仿真输出进行后处理（步骤3）。这一迭代过程造成人力成本高昂，并显著延长了存算一体设计周期。

近年来，基于大语言模型（LLM）的多智能体框架已成为自动化电子设计自动化（EDA）工作流程的有前景的解决方案，能够解读文档、将用户请求转换为模拟器输入，并管理任务执行 [37, 38]。例如，ChatEDA [37] 集成了基于大语言模型的智能体，以

本研究部分由 PRISM（SRC/DARPA JUMP2.0 中心之一）资助。作者联系方式：Ming-Yen Lee，mlee838@gatech.edu，美国佐治亚州亚特兰大的佐治亚理工学院；Shimeng Yu，shimeng.yu@ ece.gatech.edu，美国佐治亚州亚特兰大的佐治亚理工学院。

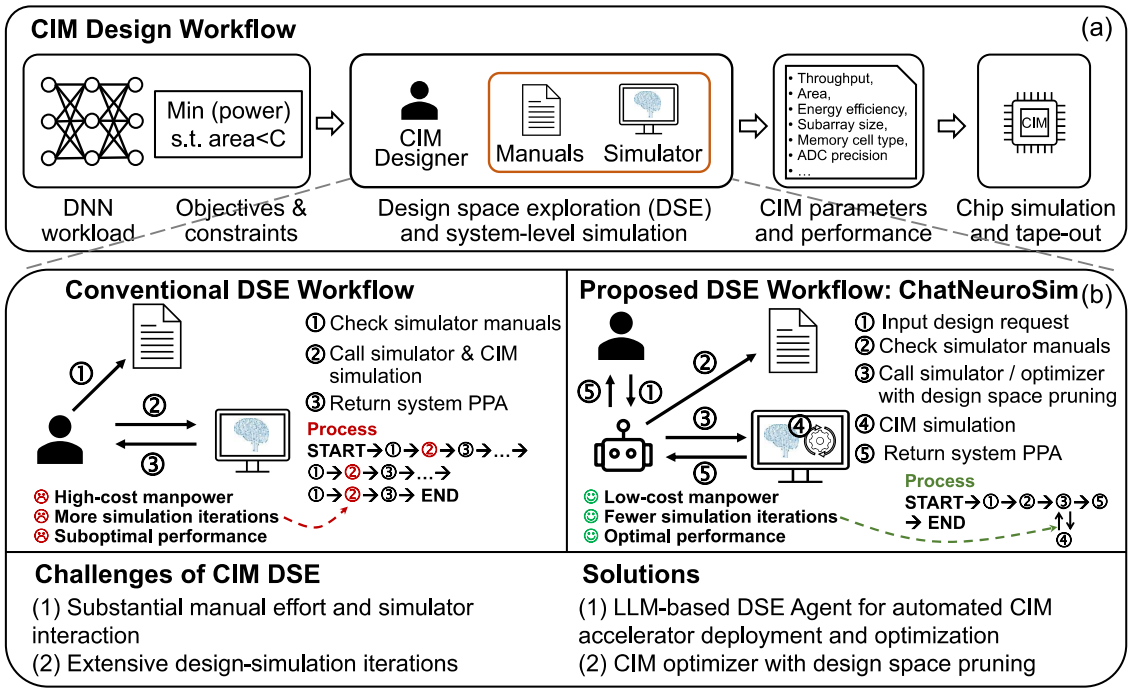



图 1。（a）存算一体（CIM）设计工作流程；（b）传统设计空间探索（DSE）工作流程和所提出的基于大语言模型的 DSE 工作流程。

自动化从寄存器传输级（RTL）到图形数据系统（GDS）的设计流程，而 ChatArch [38] 则提出了一种由大语言模型驱动的框架，用于自动化处理器架构优化。然而，尽管已有多种存内计算模拟器 [1, 16, 30], ，基于大语言模型的智能体框架在自动化存内计算仿真和部署方面的应用仍未得到探索。

除了易用性方面的挑战外，存算一体设计人员还面临着如何高效探索庞大而复杂的设计空间这一挑战。即使熟练使用模拟器，在多种资源约束下确定最优硬件配置仍然十分困难。设计人员通常依靠丰富的领域专业知识和反复仿真来优化候选配置，但最终设计仍可能不是最优的。为缓解这一挑战，人们引入了基于机器学习（ML）的优化算法，例如遗传算法、贝叶斯优化和强化学习（RL），以减少设计与仿真之间的迭代次数并加快设计空间探索 [12, 13, 15, 32, 39]。例如， [12] 采用强化学习和遗传算法，为目标深度神经网络工作负载平衡计算与存储资源；而 [39] 提出了一种结合迁移学习的强化学习辅助架构探索框架，以减少所需的迭代次数。

在存内计算设计空间探索领域，已有研究 [18]提出了一种基于强化学习的自动化优化框架，该框架融合了迁移学习与设计空间剪枝，主要针对 ResNet 和密集连接网络等传统卷积神经网络。然而，仍有若干重要方向尚未得到充分探索，包括针对视觉 Transformer 等更复杂工作负载的优化，以及系统评估设计空间剪枝在不同优化目标和约束下的通用性与有效性。

为应对这些挑战，本文提出 ChatNeuroSim，这是一个构建于 NeuroSim 之上的基于大语言模型的智能体框架，用于自动化存内计算加速器部署与优化。ChatNeuroSim 通过解析用户请求、查阅模拟器文档、生成执行脚本、调用模拟器或优化器，并返回系统级准确率、性能、功耗和面积（PPA）指标，实现存内计算设计空间探索工作流的自动化。此外，我们开发了一种基于启发式机器学习算法并结合设计空间剪枝的 CIM优化器，在先前工作 [18] 的基础上进行扩展，以支持跨多种深度神经网络工作负载的快速优化，包括残差网络、Swin Transformer [21],和视觉 Transformer [5]。本文的主要贡献总结如下：

- **用于自动化存内计算部署与优化的基于大语言模型的智能体框架：**所提出的框架利用基于大语言模型的智能体自动执行端到端存算一体工作流，从解析用户请求到调用存内计算模拟器（即 NeuroSim [30]）或优化器，并返回最终准确率以及功耗、性能和面积结果。

- **面向高效存内计算优化的设计空间剪枝：**我们提出了一种基于启发式机器学习的存内计算优化器，通过设计空间剪枝实现高效优化。该方法利用从先前工作负载中迁移的知识初始化探索空间，并在优化过程中对其进行迭代细化。与传统启发式机器学习基线相比，该方法提高了搜索效率并缩短了优化运行时间。
- **全面的智能体评估与优化分析：**我们使用包含 40 个测试用例的定制存内计算请求数据集对所提出的智能体框架进行评估，在脚本生成和仿真结果方面实现了 100% 的正确率。我们还在 22 纳米工艺节点上，使用最先进的新兴非易失性存储器，对 Swin Transformer Tiny 上的存内计算优化器进行了验证。与不同优化目标下的基线无剪枝算法相比，采用设计空间剪枝的优化器可使平均运行时间缩短至 0.42 倍–0.79 倍，并使 P95 运行时间缩短至 0.29 倍–0.69 倍。我们还分析了设计空间剪枝在不同性能指标、硬件约束和剪枝超参数下的通用性，并将所得见解集成到 ChatNeuroSim 中，以便为用户提供指导。

本文其余部分安排如下：第 2 节介绍存内计算仿真工具、基于机器学习的加速器优化，以及基于大语言模型的电子设计自动化智能体的背景。第 3 节介绍 ChatNeuroSim 框架，包括其智能体工作流，以及结合设计空间剪枝的优化过程。第 4 节使用多种存内计算请求和基准工作负载评估 ChatNeuroSim。第 5 节总结本文。附录展示 ChatNeuroSim 的用户界面（UI）和对话示例，并提供 GitHub 链接（https://github.com/neurosim/ ChatNeuroSim），供感兴趣的用户试用。

## 2 背景

### 2.1 存算一体（CIM）与存内计算模拟器

存算一体（CIM）是一种新兴的计算范式，矩阵乘法中的乘加运算（MAC）可直接在存储阵列内执行。与传统的冯·诺依曼架构相比，存算一体架构通过消除存储器与计算单元之间频繁的数据传输，显著降低了数据搬移开销。如图2所示，典型的CIM系统采用层次化组织结构，分为芯片、图块、处理单元（PE）和子阵列层级。深度神经网络权重存储在子阵列的存储单元中，而输入激活值则施加到字线（WL）上。乘加运算在存储阵列内自然完成；在模拟存内计算（ACIM）子阵列中，运算结果由模数转换器（ADC）进行数字化，而在数字存内计算（DCIM）子阵列中，结果则通过数字加法树累加。

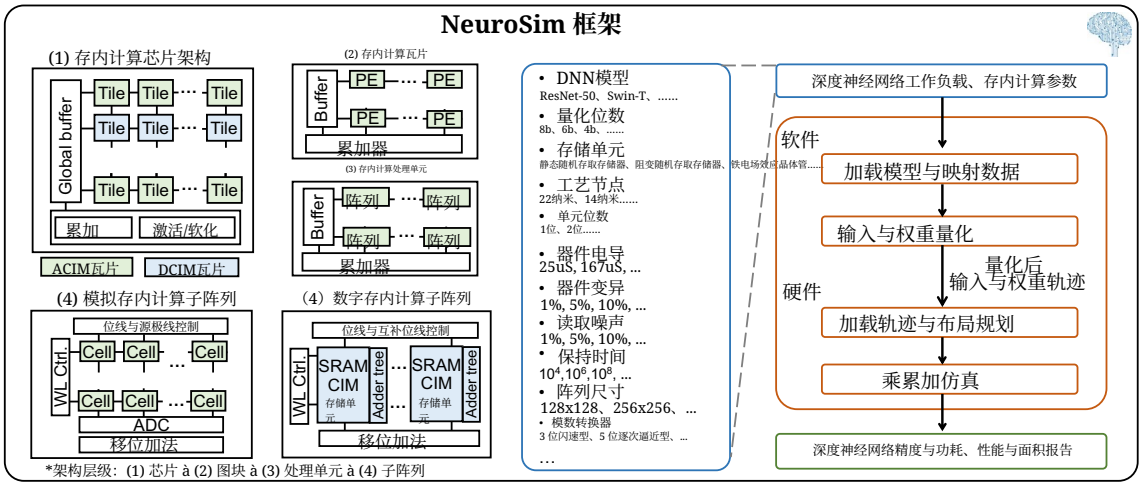



图 2. 存内计算（CIM）模拟器：NeuroSim 框架。

在多种硬件约束和性能目标下设计完整的CIM系统，仍是一项复杂且耗时的任务。为促进CIM架构设计和系统级优化，研究人员提出了多种CIM仿真框架 [1, 16, 30]。通过联合建模深度神经网络工作负载和硬件特性，与晶体管级仿真相比，这些模拟器减少了设计空间探索（DSE）过程中的设计工作量和迭代时间。

在这些框架中，NeuroSim [17, 28, 30] 已成为一个全面的CIM仿真平台，通过统一的用户可配置接口，集成了先进工艺节点下DNN工作负载、存内计算架构、存储器件特性和外围逻辑电路的建模。如图 2 所示，接收用户定义的参数后，NeuroSim首先加载DNN模型和架构配置。输入和权重经过量化，并映射到CIM子阵列上，以进行准确率仿真，同时对器件变异和模数转换器（ADC）量化误差等硬件非理想因素进行建模。随后，将生成的量化数据轨迹传递给硬件级模型，以估算系统级PPA指标。最新的NeuroSim V1.5 [30] 支持在存内计算架构上部署卷积神经网络（CNN）和视觉Transformer（ViT），用于DNN推理。CNN工作负载采用ACIM子阵列实现，而ViT则采用混合ACIM–DCIM架构，其中ACIM用于线性运算（例如Q、K、V投影和MLP层），DCIM用于注意力分数计算（$QK^T$）和加权聚合。这种软硬件协同建模能够对存内计算系统进行端到端评估，并在流片前提供关键的性能洞察。

### 2.2 存内计算设计空间探索（设计空间探索）

尽管与晶体管级仿真和优化相比，存内计算模拟器能够加快评估速度，但探索规模庞大且高度耦合的CIM设计空间仍然充满挑战。以图2中的设计空间示例为例，可调硬件参数的数量很容易使设计空间规模达到 $10^3$–$10^4$ 量级。要高效探索这一空间，需要丰富的领域专业知识和手动调优。同时，还必须筛除违反面积、功耗或性能约束的不可行配置。因此，手动试错和调优方法

随着设计复杂度的提高，效率会越来越低。自动且高效的存内计算优化方法对于加快设计空间探索进程、快速识别高性能硬件配置至关重要。

### 2.3 用于人工智能加速器设计空间探索的机器学习算法

为应对设计空间探索的可扩展性挑战，研究人员提出了多种机器学习算法，用于自动优化人工智能加速器。模拟退火 [14] 是一种启发式搜索算法，它基于接受“温度”概率性地接受性能较差的设计点，从而避免搜索过程陷入局部最优解。随着“温度”降低，搜索会逐步聚焦于设计空间中潜力较大的区域。遗传算法 [12]是另一种启发式搜索算法，它将设计点表示为基因序列，并通过多代迭代中的变异和交叉操作来演化候选解。贝叶斯优化构建代理模型，以估算目标函数并识别预期性能较高的潜在配置。树结构帕岑估计器（TPE） [2] 是一种贝叶斯优化技术，它根据观测到的性能对参数分布进行建模，并通过高效采样获取高质量设计点。

强化学习（RL）通过将设计空间探索过程建模为智能体—环境交互，提供了另一种有效范式。在这一过程中，动作对应于硬件参数的选择，奖励则反映性能目标。基于策略的强化学习方法 [31, 34] 非常适用于连续或高维参数空间，而基于价值的方法 [23, 35, 36]，（如Q学习）则通过迭代价值估计，特别适合离散设计空间。这些由机器学习驱动的方法减少了对手动调优的依赖，显著提升了人工智能加速器设计空间探索的效率。

### 2.4 基于大语言模型（LLM）的电子设计自动化智能体框架

近年来，大语言模型（LLM）在对话交互、代码合成和文档理解等自然语言处理应用中展现出卓越能力 [8, 37, 38]。这些能力推动了人们对将大语言模型应用于电子设计自动化工作流的日益浓厚兴趣。VerilogCoder [8] 利用大语言模型实现自动化Verilog代码生成，并进行语法检查和功能验证。ChatEDA [37] 引入了基于大语言模型的智能体接口，可自动完成从寄存器传输级（RTL）生成到图形数据系统II（GDSII）实现的电路设计流程。通过将用户请求分解为结构化子任务，这些多智能体框架能够以极少的人工干预自动完成任务调度、脚本生成、语法验证和工具调用。

提示工程 [3] 和检索增强生成（RAG） [19]是构建高效基于大语言模型的多智能体系统的两项关键技术，无需进行模型再训练或微调。提示工程提供结构化的输入输出模板、推理指南和执行约束，在任务处理过程中引导大语言模型的行为。经过精心设计的提示能够实现对用户意图的一致解析、可靠的参数提取以及确定性的工具调用。RAG则通过在生成响应前从外部知识库中检索相关领域知识（例如模拟器手册和参数规范），进一步提升可靠性。通过将大语言模型的推理建立在检索到的权威信息之上，RAG能够显著提高响应准确率，并减少复杂电子设计自动化任务中的幻觉。

鉴于基于大语言模型的多智能体系统在电子设计自动化领域的优势，为存内计算开发专用的大语言模型驱动的智能体框架引起了研究界的广泛关注。此类框架能够减少人工工作量、提高设计生产力，并加快存内计算部署。

## 3 ChatNeuroSim：基于大语言模型的自动化存内计算部署与优化智能体框架 3.1 总体框架

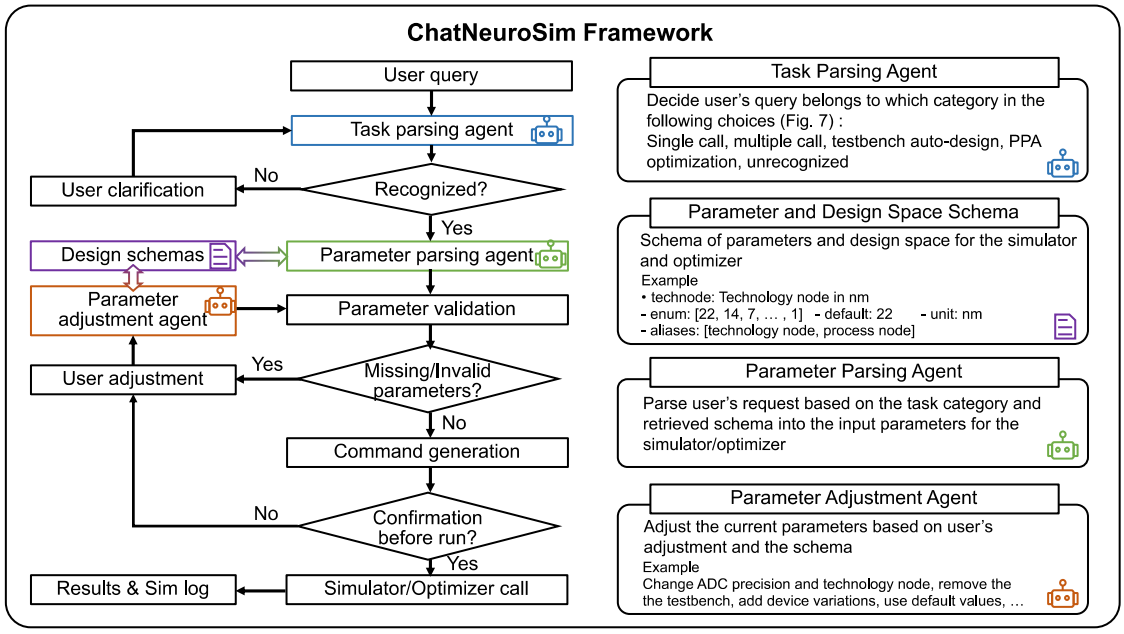



图3. ChatNeuroSim：总体框架。

图3展示了 ChatNeuroSim 的总体框架和工作流程，其中包含三个基于大语言模型的智能体，以及一组用于参数和设计空间检索的模式。ChatNeuroSim 接受用户的存内计算查询作为初始输入（例如图5），并首先调用任务解析智能体，将请求归类为某一执行类别。

ChatNeuroSim 支持四种典型请求类别：单次调用、多次调用、测试平台自动设计和 PPA 优化。当请求无法分类时，也会处理这一特殊情况。表1总结了这些类别的定义和代表性查询示例。单次调用请求用于评估特定硬件配置的准确率和/或 PPA 指标。多次调用请求指定多个明确的参数集，从而针对不同测试平台进行多次模拟器调用。测试平台自动设计类别同样涉及多次模拟器调用，但不同之处在于参数值并未明确指定。相反，用户会表达较高层次的意图（例如，分析 SRAM 工艺缩放对 PPA 趋势的影响），参数解析智能体则根据设计空间模式自动确定合适的扫描取值。最后一类 PPA 优化旨在用户定义的约束下确定最佳硬件配置。在这种情况下，ChatNeuroSim 会调用 CIM 优化器（第3.3节）；该优化器在完成参数解析和验证后，会与存内计算模拟器进行内部交互。如果任务解析智能体无法有把握地确定请求类别，就会提示用户澄清，并重新提交调整后的请求。

确定请求类别后，参数解析智能体会检索模拟器或优化器输入所对应的模式。每个模式条目都规定了参数属性，例如支持的值、默认

表1. 存内计算请求类别

| 查询类别 | 描述 | 示例 |
|---|---|---|
| 单次调用 | 单个测试平台和一次仿真-在一组固定参数下进行仿真。 | 我想在存内计算架构上使用 VGG8 对 CIFAR-10 进行仿真，输入和权重均采用 8位量化。我只想获得功耗、性能和面积估计。存储器件为 22纳米阻变存储器，单元精度为 1位。状态电导为 25uS 和无变化时为 167uS。子阵列大小为 128x128，且 ADC 精度为 7bit。 |
| 多次调用 | 多个测试平台和仿真-用户使用不同的指定参数值。 | 我想在存内计算架构上仿真 CIFAR-10 上的 VGG8 输入和权重均采用 8位量化。我只想获取 22纳米、14纳米和 7纳米工艺节点下的功耗、性能和面积估计。存储器件为静态随机存取存储器。子阵列大小为 128x128，且 ADC 精度为 7 位。 |
| 测试平台自动设计 | 多个测试平台和仿真，使用用户指定的参数名称，但未指定参数值。 | 检查不同情况下 CIFAR-100 上残差网络-18 的准确率下降输入和权重量化精度。仅仿真准确率，不考虑功耗、性能和面积。运行 5 批次，每批次大小为 100。使用 1b 单元，子阵列大小为 128x128，模数转换器为 7b。 |
| PPA优化 | PPA优化请求，用于寻找最佳存内计算配置——在约束条件下进行部署。 | 我想设计一款基于 22 纳米工艺节点的存内计算芯片，面积不超过 3600mm$^2$ 运行 ImageNet 上的 ResNet-50，并将功耗降至最低。使用模拟退火，在 20 轮内找出器件类型、ADC 类型、ADC 精度、子阵列大小和子阵列多路复用器的最佳参数组合。 |

数值、单位、工艺节点和别名。基于这些定义，参数从用户的自然语言查询中提取并转换为结构化输入。针对不同类别设计的系统提示旨在确保参数解析一致，并正确格式化以用于下游仿真或优化任务。

用户请求可能会遗漏必需参数或包含无效值。为此，ChatNeuroSim会根据模式验证所有提取的参数，并在检测到缺失或无效条目时向用户提供反馈。用户随后可以发送调整请求，由参数调整智能体处理，以按照模式和用户意图插入缺失参数或修改现有值。所有参数通过有效性检查后，ChatNeuroSim会生成执行脚本，并在执行前请求用户确认。确认后，该框架调用模拟器或优化器，并返回PPA指标和执行日志。

在整个工作流程中，用户使用标准存内计算术语与ChatNeuroSim交互，无需手动查阅模拟器文档或参数依赖关系。该框架自动将高层用户请求映射为有效的模拟器或优化器输入，支持对多个参数和仿真进行动态调整，并可灵活添加或移除测试平台。通过自动化从用户查询到仿真结果的整个流水线，ChatNeuroSim显著减少了设计和优化高性能存内计算系统所需的时间和人工工作量。

### 3.2 智能体的提示设计

ChatNeuroSim 中的每个智能体都配有专用的系统提示，以确保其能够正确、一致且可靠地响应用户输入。图 4 展示了 ChatNeuroSim 中三个智能体的系统提示精简版，

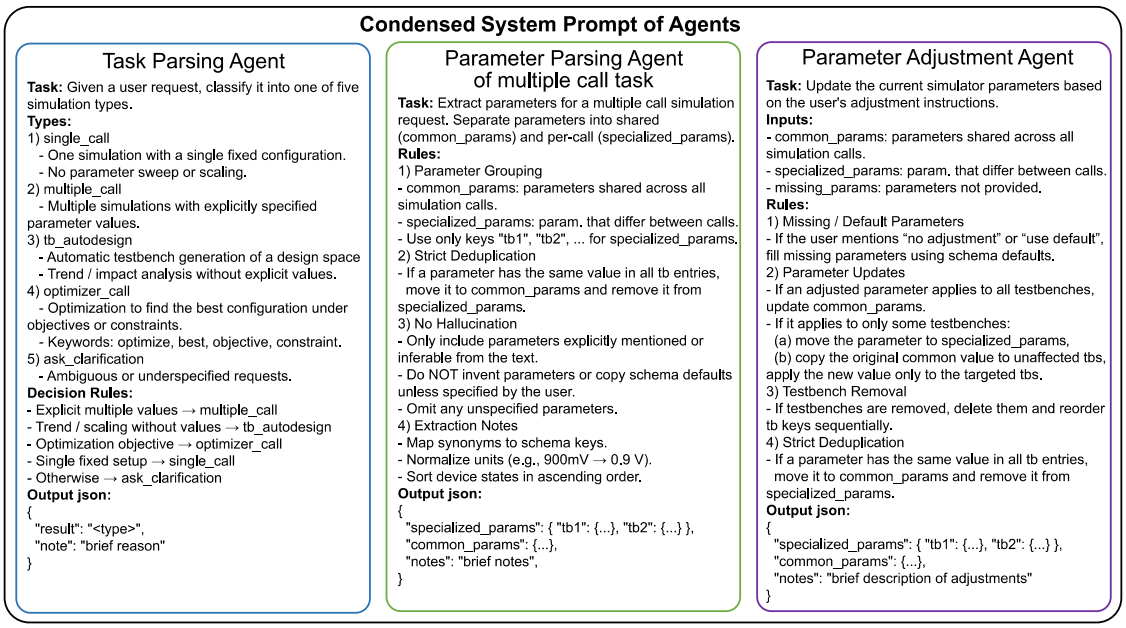



图 4. ChatNeuroSim：面向智能体的提示设计。

重点呈现智能体有效运行所需的核心组成部分。每个精简提示都由三个关键要素构成：任务描述、规则规范和输出结构模板。下面总结了三个智能体的提示设计流程。

• **任务解析智能体：**任务解析智能体将用户请求分类为预定义的执行类别。如表1所总结的，每个类别均由一组独特的语义特征定义。**类型**部分在提示中明确规定了这些定义，而区分性特征则编码在**决策规则**中。基于这些结构化描述，该智能体确定最合适的请求类别。为增强鲁棒性，提示还定义了异常处理机制：如果无法将请求自信地匹配到任何预定义类别，智能体会请求用户提供进一步说明。智能体同时输出分类结果及其决策依据。

• **参数解析智能体：**确定请求类别后，参数解析智能体会获取特定类别的提示，以从用户查询中提取结构化参数。这种类型感知提示设计确保智能体能够适应不同任务中的异构参数格式和模式定义。例如，在多调用任务中，智能体会提取 NeuroSim 模式中定义的模拟器输入参数；而在 PPA 优化任务中，它会获取优化器特定参数，例如设计空间和优化算法配置。每个提示由三个组件组成：任务描述、参数提取规则和结构化输出模板。如图 4 所示，提取出的参数被组织为两组：专用参数 (specialized_params)，这些参数因测试平台而异；以及通用参数 (common_params)，这些参数在所有测试平台之间共享。智能体遵循一组明确规则，包括分离

专用参数和通用参数、严格禁止幻觉的约束以及格式规范化要求。输出包括解析后的参数和提取过程中的关键说明。

- **参数调整智能体：** 参数调整智能体负责处理用户提出的修改、添加或删除仿真参数的请求。对于缺失的参数，智能体会根据用户明确提供的输入或模式中定义的默认值为其赋值。此阶段的一项重要原则是正确管理专用参数和通用参数，尤其是在涉及多个测试平台的场景中。调整后，参数可以在专用范围和通用范围之间相互转换。此外，添加或删除测试平台时，智能体需要重新排列测试平台索引，以保持一致性。这些规则直接编码在系统提示中，以确保可靠地更新参数。智能体会输出修订后的参数集，并附上详细的调整说明。

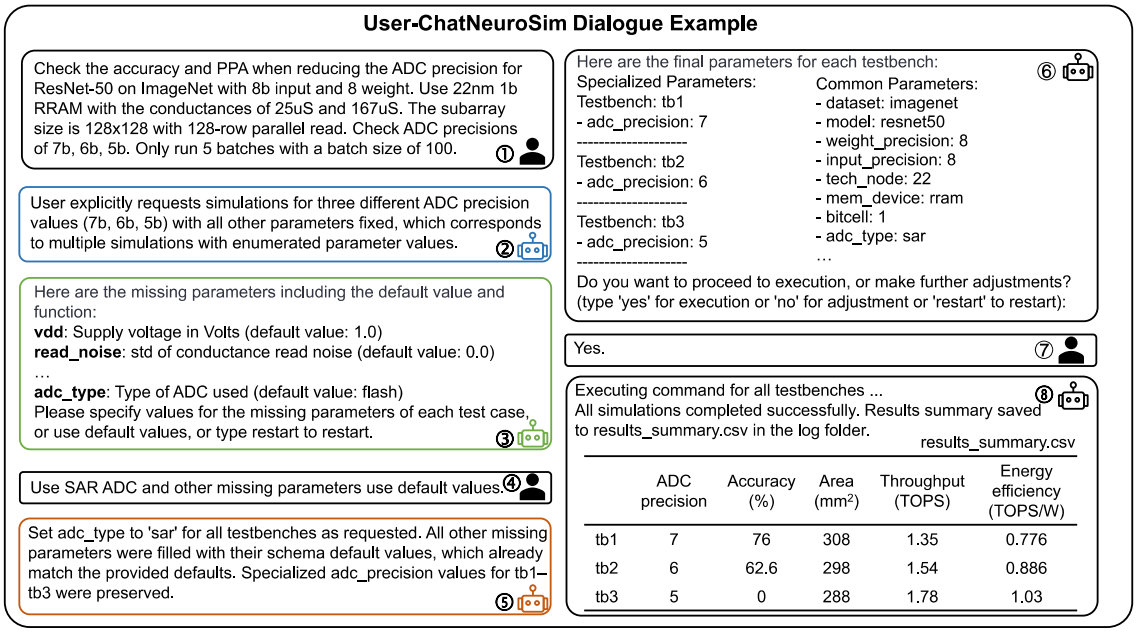



图5. 用户与 ChatNeuroSim 之间的对话示例。

图5展示了用户与 ChatNeuroSim 之间具有代表性的对话，演示了从提交请求、调整参数、智能体响应到最终执行仿真的端到端交互。在此示例中，用户提出了多次调用请求，要求使用三种不同的模数转换器精度评估 ResNet-50 的准确率和功耗、性能和面积。任务解析智能体根据提示中定义的标准正确识别了请求类别。随后，参数解析智能体提取仿真参数，并报告执行所需的缺失项。用户提供默认参数、模数转换器精度等缺失值后，参数调整智能体会相应地更新参数集。如图5步骤6所示，三个测试平台的最终配置按专用参数组和通用参数组进行整理。

用户确认仿真执行后，ChatNeuroSim 会生成执行脚本、调用 NeuroSim，并记录仿真结果和日志。在交互过程中，用户只需输入标准的存内计算术语

无需手动查阅模拟器文档或构建执行命令。通过将用户的高层意图自动映射为有效的参数配置和可执行脚本，ChatNeuroSim 可无缝完成仿真和结果检查。这种自动化的智能体驱动工作流显著缩短了设计周转时间，减轻了存内计算系统探索与优化中的人工负担。

### 3.3 采用设计空间剪枝的存内计算优化器

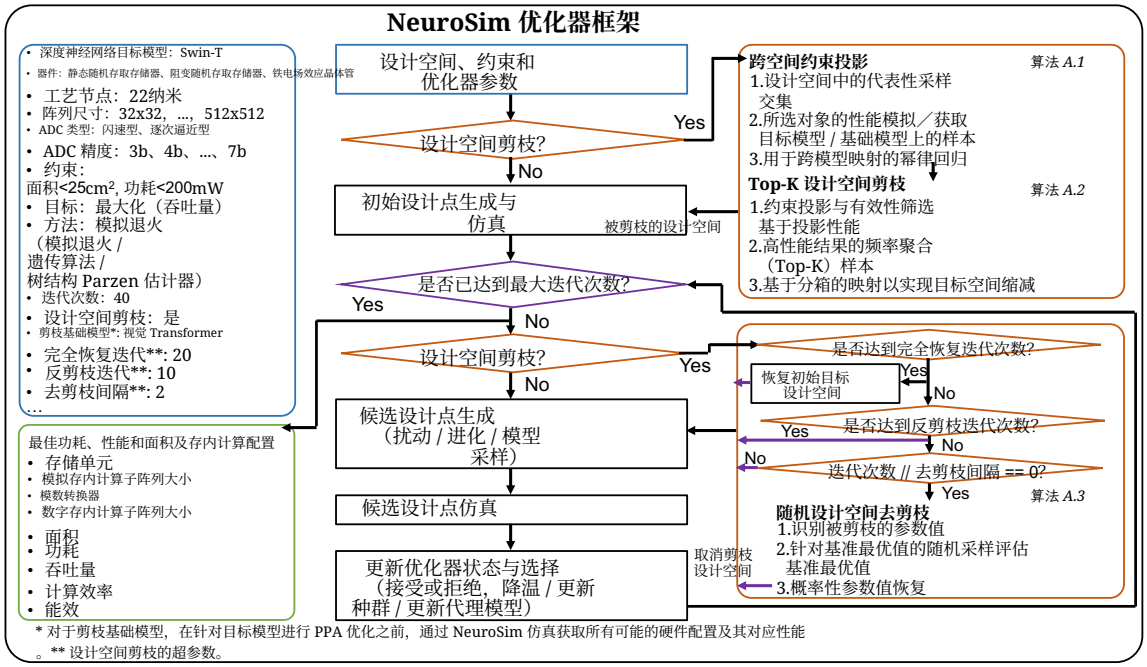



图6. ChatNeuroSim：存内计算优化器框架。

除智能体框架外，ChatNeuroSim 还集成了存内计算优化器，可通过高效的设计空间搜索实现自动化 PPA 优化。如图6所示，优化器接受以下输入：(1) 目标深度神经网络工作负载的优化目标；(2) 硬件设计空间规范（例如，存储器件类型、子阵列尺寸、模数转换器配置）；(3) 功耗和面积等硬件约束；(4) 所选的启发式优化算法；以及 (5) 用于控制所提出的设计空间剪枝算法的超参数。在优化过程中，优化器通过仿真候选存内计算配置与 NeuroSim 反复交互，最终报告满足约束条件的最佳 PPA 及相应的硬件配置。

**不进行设计空间剪枝的基线优化**

收到用户输入后，优化器会初始化设计点，并迭代地优化候选配置。在基线模式下，用户选择一种标准启发式算法（即模拟退火（SA）、遗传算法（GA）或树结构 Parzen 估计器（TPE）），且不进行设计空间剪枝。尽管这些算法的采样策略各不相同，但每次迭代都可以归纳为三个共同步骤。

- **候选生成：** SA 通过随机扰动当前表现出色的配置来生成新的候选项。GA 对一组表现出色的“父代”配置进行变异和交叉操作，从而构建新一代。TPE 根据学习得到的代理分布采样候选项；该代理分布基于此前观测到的“好”样本与“坏”样本的参数统计信息构建。

- **仿真与记录：** 对于每个候选项，优化器都会调用存内计算模拟器获取其功耗、性能和面积，然后将结果记录在历史缓冲区中。
- **状态更新：** 优化器会根据更新后的历史缓冲区更新其内部状态及表现最佳的配置集合。例如，SA 会根据候选项的目标改进情况以及由温度控制的概率来决定接受或拒绝该候选项；GA 会更新下一代所用的父代池；TPE 则会更新其代理模型使用的参数分布。

在达到 最大迭代次数后，优化器会返回最优配置及其功耗、性能和面积 .

**设计空间剪枝的动机**

虽然这些启发式算法具备有效的搜索能力，但当设计空间较大且每次 PPA 评估成本较高时，优化运行时间仍可能很长。尤其是，与主要探索存内计算参数的卷积神经网络（例如 ResNet-50）优化相比，复杂的 Transformer 工作负载通常包含更大的设计空间。对 ViT-Base（ViT-B）和 Swin Transformer Tiny（Swin-T）等视觉 Transformer 进行优化时，通常需要引入与数字存内计算相关的配置，从而扩大搜索空间，并增加 NeuroSim 仿真延迟（第 4.3 节）。为进一步提高搜索效率，我们提出了一种由迁移驱动的设计空间剪枝策略，该策略利用此前针对其他工作负载（称为基础模型）进行优化时获得的知识，加速对目标模型的优化。

**通过跨空间约束投影进行设计空间剪枝**

Fig. 第 6 节概述了集成到存内计算优化器中的三种与剪枝相关的算法，详细伪代码见附录 A。前两个过程（即跨空间约束投影和 Top-$K$ 剪枝）在第一次优化迭代之前执行。

**跨空间约束投影**（算法1）：将知识从基础模型迁移到目标模型时，一个关键挑战是，不同设计空间中的硬件约束（例如功耗和面积）可行性通常无法直接比较。因此，我们提出一种跨空间投影方法，将约束值从基准空间映射到目标空间。受硬件性能趋势中常见的幂律缩放现象启发，我们将基准空间指标 $X$ 与相应的目标空间指标 $Y$ 之间的关系建模为：

$$Y = \alpha X^{\beta}, \tag{1}$$

其中，$\alpha$ 和 $\beta$ 表示特定于空间的缩放效应。应用 $log()$函数后，投影可表示为双对数域中的线性回归：

$$\ln(Y^{m,c}) = a_1^{m,c} \ln(X^{m,c}) + a_0^{m,c}, \tag{2}$$

其中，$X^{m、c}$和 $Y^{m、c}$分别表示基准设计空间和目标设计空间中存储器类型 $m$ 对应的约束 $c$ 值。系数（$a^{m、}{}_0{}^{c}$、$a^{m、}{}_1{}^{c}$）通过普通最小二乘法（OLS）估计，如算法1所示。

算法1的流程如下。首先，识别基准空间与目标空间在一致参数下的交集设计空间 I。针对每种存储器类型，我们选取关键参数的边界值 V（例如子阵列尺寸、ADC分辨率、多路复用器配置），并从这些边界值的组合中随机抽取候选点，以构建拟合集。在目标空间中对这些候选点进行仿真，以获得 $Y^{m,\ c}$。对于基准空间，我们假设 PPA 值已在先前的优化运行中完成仿真并存储，因此可通过查找表而非重新仿真来获取 $X^{m,\ c}$

。利用成对样本（$X^{m,\ i\ c}$， $Y^{m,\ i\ c}$），我们进行普通最小二乘拟合，以求得投影系数。

**Top-K 设计空间剪枝**（算法 2）：利用前一算法中的投影函数，算法 2 在优化迭代开始前对目标设计空间进行剪枝。我们首先将交集空间中各设计点的约束值从基础设计空间投影到目标设计空间，并筛除预测会违反目标约束的配置。此过程无需在目标设计空间中运行仿真。随后，我们统计基础模型在交集设计空间中的Top-K高性能样本里各参数的值频率。为增强剪枝的鲁棒性并处理离散值粒度问题，我们将参数值分为小、中和大分箱。我们按分箱聚合频率，并保留超过预设阈值的分箱，以此构建高性能区域。由此得到剪枝后的目标设计空间，其中优先保留与投影后的基础设计空间中高性能配置相关的值。

**用于恢复和提升鲁棒性的随机去剪枝**（算法3）：由于剪枝是以牺牲探索为代价来提高效率，全局最优配置可能不在初始剪枝后的空间中。为降低这一风险，我们引入一种随机去剪枝机制（算法3），在优化过程中逐步恢复被排除的值，在利用剪枝后空间的同时，受控地向完整设计空间重新扩展，从而取得平衡。

算法3由去剪枝间隔和完全恢复阈值控制。每次触发去剪枝事件时（即每经过一个去剪枝间隔），算法都会识别上一次剪枝中移除的参数值，并通过随机采样对其进行评估。具体而言，它将高性能基线配置中的缺失值替换为候选值，构成候选设计；随后将这些候选设计的性能与选定的基线进行比较，并计算每个候选值的胜率。然后将胜率用作概率评分，以决定是否将该值恢复到当前搜索空间中。此过程会定期重复，直至优化达到完全恢复阈值，之后恢复完整的原始设计空间，以进行全面搜索。

总之，跨空间约束投影、Top-K 剪枝和随机去剪枝为复杂的深度神经网络工作负载提供了高效且稳健的存内计算优化。这种集成方法降低了早期探索阶段的仿真开销，同时保留了恢复被排除区域并获得接近全局最优功耗、性能和面积解决方案的能力。第4节将详细分析这些技术在不同目标、约束和剪枝超参数下何时能带来最大收益。

## 4 评估与案例研究

### 4.1 实验设置

ChatNeuroSim 使用 Python 实现，智能体工作流基于 LangGraph [9]开发。为评估其在不同用户意图下的正确性和鲁棒性，我们构建了一个包含 40 个 CIM 查询的请求级基准，表1中列出的每个任务类别各有 10 个请求。对于智能体后端 API，我们使用 GPT-5.1 [26],GPT-5-mini [25], DeepSeekV3.2-Chat 和 DeepSeekV3.2-Reasoner [4]来验证智能体功能。所有 ChatNeuroSim 智能体执行均在 Intel(R) Xeon(R) Gold 5317 CPU 上运行。NeuroSim V1.5 仿真在 NVIDIA RTX A6000 GPU 和 Intel(R) Xeon(R) Gold 5317 CPU 上运行。我们还基于 Streamlit Python 库实现了一个类似聊天机器人的用户界面，以便进行交互式使用 [10]。附录 B 提供了一个用户界面对话示例及其对应的仿真输出。

为了评估 CIM优化器，我们在不同的设计空间、优化算法、硬件约束和性能目标下，测试其在卷积神经网络和视觉 Transformer 工作负载上的优化效果。我们进一步比较

表2. CIM优化器：设计空间探索参数

| 参数 | 范围/取值 |
|---|---|
| 模型 | ResNet-50 ，Swin-T ，ViT-B |
| 数据集 | ImageNet |
| 输入/权重精度 | 8b / 8b |
| 工艺节点 | 22nm |
| 器件 | SRAM, RRAM, 铁电场效应晶体管 |
| 模拟存内计算子阵列列数 / 行数[a] | 32、64、128、256、512 |
| ADC 类型 | 闪速型，逐次逼近型 |
| ADC 精度[b] | 3b - 7b |
| 每个 ADC 的列数 | 4, 8, 16, 32 |
| 权重复制[c] | 0, 1 |
| DCIM 子阵列列数/行数[a] | 32, 64, 128, 256 |

*a*。ResNet-50 仅使用 ACIM 实现，Swin-T 和 ViT-B 的线性运算使用 ACIM 实现，注意力分数计算使用 DCIM 实现。*b*。并行读取等于 2 模数转换器精度。ACIM 子阵列行数小于 2 模数转换器精度被视为无效设计点。*c*。权重复制仅应用于 ResNet-50 的卷积运算。0 表示流水线不进行权重复制（气泡长、面积小），1 表示流水线进行权重复制（气泡短、面积大）。*d*。总设计空间：ResNet-50：5280，Swin-T：42240，ViT-B：42240。

采用和未采用所提出的设计空间剪枝时的运行时间及优化结果，并在后续案例研究中总结剪枝在何种情况下最有益的实践见解。这些见解会作为优化请求中的建议集成到 ChatNeuroSim 中。

表 2 总结了优化基准测试中使用的深度神经网络工作负载和存内计算参数。我们以三个 ImageNet 模型的推理功耗、性能和面积为目标：ResNet-50 [7], Swin Transformer Tiny（Swin-T） [21], 以及 Vision Transformer Base（ViT-B） [5]。由于 ResNet-50 的设计空间小于 Swin-T 和 ViT-B，且 NeuroSim 运行时间更短，案例研究主要以 Swin-T 作为代表性视觉 Transformer 工作负载。在位串行计算模式下，输入精度和权重精度均为 8 位。一次操作定义为一次 8 位加法或一次 8 位乘法。我们评估三种 22 纳米存储器技术：静态随机存取存储器（SRAM）、阻变随机存取存储器（RRAM）[11], 以及铁电场效应晶体管（FeFET） [6]。模数转换器选项包括闪速型和逐次逼近寄存器（SAR）架构，精度为 3–7 位，列共享设置为 4–32。为探索性能与面积权衡，设计空间进一步包含权重复制选项，可在层流水线过程中降低流水线气泡开销 [28]。

运行时间和所能达到的最佳性能对于实际优化都至关重要。在运行时间评估中，CIM优化器和 NeuroSim PPA仿真均在配备 48 个核心的 Intel(R) Xeon(R) Gold 5317 CPU 上执行。我们分配 32 个物理核心用于并行仿真，并将其余核心留作操作系统开销、输入/输出和模拟器内部线程使用。

为确定优化迭代中合适的批次大小，本文探究了批次大小（16、32 和 48）对搜索效率与运行时间的影响。如表 3 所示，达到目标最优功耗、性能和面积所需的样本数量在不同批次大小下相对一致。结果基于每种算法的 50 次独立运行计算得出。在 32 核并行下，每次迭代的平均仿真时间随批次大小从

16 扩大至 48 而近似线性增加。在针对 Swin-T 进行优值优化时所评估的配置中，综合考虑三种启发式算法，批次大小 32 的平均运行时间最低。因此，在后续优化案例研究中，采用批次大小 32 与 32 核并行。

表 3. 不同策略和批次大小下 Swin-T FoM 优化的平均仿真样本数

| 批次 Size | GA （样本） | SA （样本） | TPE （样本） | 平均批次 运行时间（分钟）* | 平均总 运行时间（分钟）** |
|---|---|---|---|---|---|
| 16 | 1277 | 719 | 1862 | 5.9 | 472 |
| 32 | 1048 | 622 | 1660 | 7.2 | **250** |
| 48 | 1459 | 598 | 1831 | 9.9 | 266 |

GA：遗传算法，SA：模拟退火，TPE：树结构 Parzen 估计器。FoM =能效 x 计算效率，单位：$TOPS^2/W/mm^2$，面积约束：25 $cm^2$，功耗约束：200 mW * 平均批次运行时间基于 30 次独立仿真试验并采用 32 核并行进行表征。** 计算公式为 $\frac{1}{3}\sum_{alg\in\{GA,SA,TPE\}}\left(\frac{S_{alg}}{n}\times T_{batch}\right)$，其中 $S_{alg}$ 为平均仿真样本，$n$ 为批次大小，$T_{batch}$ 为平均批次运行时间。

为了对设计空间探索运行时间进行切合实际的估计，我们采用基于表征的成本模型。由于 NeuroSim 评估每个批次大小约需 7 分钟的运行时间开销，我们首先通过对批次大小 $n \in 1, 8, 16, 24, 32$ 运行 30 次独立试验来表征模拟器运行时间。除 $n = 32$ 外，我们还纳入了更小的批次大小，因为所提出的剪枝算法在反剪枝过程中会引入验证样本，并且部分候选方案可能已是先前迭代中缓存的仿真结果，从而导致仿真批次大小低于 32。

在优化过程中，我们记录每次迭代中评估的唯一设计点数量 $i$，记为 $n_i$。总运行时间估算如下：

$$\mathcal{T}total = \sum_{i=1}^{M} \left(T_{logic,i} + T_{batch}(n_i)\right), \tag{3}$$

其中，$M$ 为优化迭代次数，$T_{logic,\ i}$ 表示优化器的计算开销（与仿真时间相比可以忽略不计），而 $T_{batch}(n_i)$ 是评估 $n_i$ 个设计点时测得的 NeuroSim 运行时间。对于未明确表征的 $n_i$ 取值，我们在相应区间内采用线性插值。基于此模型，我们报告 50 次独立优化运行的平均运行时间和 $P95$ 运行时间，以反映算法的变化性和硬件约束下的仿真成本。除非另有说明，以下实验中的所有运行时间和性能结果均基于 50 次独立运行。

### 4.2 案例研究 1：ChatNeuroSim 请求数据集的准确率

第一个案例研究评估 ChatNeuroSim 处理不同用户请求的正确性。评估数据集包含 40 个 CIM 查询，表1所定义的每种任务类别各有 10 个请求。当且仅当 ChatNeuroSim 生成有效且可运行的执行脚本，成功完成运行并产生仿真结果时，才认为请求处理正确。在本研究中，我们排除参数调整阶段，重点评估 ChatNeuroSim 一次性脚本生成的能力。我们使用四种大语言模型 API 后端评估请求处理准确率：GPT-5。1 [26],GPT-5-mini [25],DeepSeekV3.2-Chat 和 DeepSeekV3.2-Reasoner [4]。表 4 汇总了各模型的总体准确率及各类别的细分结果。

如表 4 所示，使用 GPT-5.1 时，ChatNeuroSim 在全部 40 项请求中均达到 100% 准确率，并为所有任务类别成功生成正确脚本。使用 GPT-5-mini 时，总体准确率降至 90%，其中

表 4. ChatNeuroSim：所提请求数据集的准确率 集

| 大语言模型 / 准确率 (%) | 单次调用 | 多次调用 | 测试平台自动设计 | PPA优化 | 总计准确率 |
|---|---|---|---|---|---|
| GPT-5.1 | 100 | 100 | 100 | 100 | 100 |
| GPT-5 mini | 100 | 90 | 80 | 90 | 90 |
| DeepSeekV3.2-Reasoner | 90 | 100 | 100 | 90 | 95 |
| DeepSeekV3.2-Chat | 80 | 90 | 60 | 80 | 77.5 |

大多数错误出现在需要进行比单次调用请求更复杂的参数提取的任务类别中。对于 DeepSeekV3.2 系列，DeepSeekV3.2-Reasoner 的总体准确率保持在 95%，而 DeepSeekV3.2-Chat 的准确率为 77.5%。这些结果表明，多次调用、测试平台自动设计和 PPA优化等更复杂的请求类型，对参数解析保真度和规则遵循提出了更高要求。

对于GPT-5.1以外的大语言模型后端，可通过完善智能体提示词、扩大规则覆盖范围，或引入额外的验证与校准智能体来交叉检查调度决策和提取的参数，从而实现进一步改进。近期研究表明，采用更结构化、更细粒度工作流的多智能体框架，能够显著提升复杂EDA任务中的鲁棒性与可靠性 [8, 40]。在本工作中，我们迈出了对基于大语言模型的智能体进行CIM工作流系统评估的第一步，并证明在结合GPT-5.1时，ChatNeuroSim能够可靠地处理所有已评估的请求。降低智能体执行成本并提升其他大语言模型后端下的效率，留待未来工作。

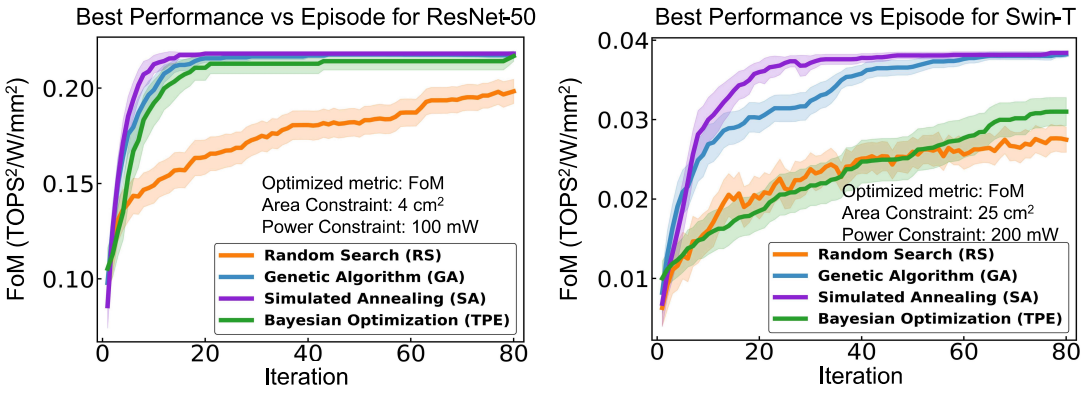
### 4.3 案例研究2：在不进行设计空间剪枝的情况下对ResNet-50和Swin-T进行FoM优化 g

在案例研究2中，我们评估所提出的CIM优化器在不进行设计空间剪枝时的优化性能与运行时间。目标是建立基线并回答以下问题：传统启发式优化算法是否足以针对不同的深度神经网络工作负载，快速收敛并找出最优功耗、性能和面积配置？

我们在功耗和面积约束下，对 ResNet-50 和 Swin Transformer Tiny（Swin-T）进行品质因数（FoM）优化，采用四种优化算法：随机搜索（RS）、遗传算法（GA）、模拟退火（SA）和基于树结构 Parzen 估计器（TPE）的贝叶斯优化。FoM 定义为能效与计算效率的乘积。每项实验运行 80 次优化迭代，并进行 50 次独立运行。

图 7 展示了各算法达到的平均最佳 FoM，以及相应的运行时间统计。对于 ResNet-50，三种启发式算法（GA、SA 和 TPE）的收敛速度均快于随机搜索。其中，SA 始终收敛最快，平均 FoM 性能更优，且平均运行时间和 P95 运行时间均最低。此处，运行时间定义为优化器首次找到性能最佳配置时所处迭代的实际运行时间。所有运行时间数值均使用第 4.1 节所述的基于表征的成本模型进行估算。P95 运行时间根据 50 次运行的经验分布计算，表示 95% 的运行可在该时间内完成，可用于估算接近最坏情况的性能。

如图 7 所示，使用遗传算法、模拟退火或树结构 Parzen 估计器优化 ResNet-50，约一小时即可完成。相比之下，Swin-T 优化的运行时间显著更长，这是因为 NeuroSim 仿真延迟增加且设计空间更大。即使使用模拟退火这一评估方法中效率最高的算法，优化器




| Model | ResNet-50 | | | | Swin transformer tiny | | | |
|---|---|---|---|---|---|---|---|---|
| Methods | RS | GA | SA | TPE | RS | GA | SA | TPE |
| Avg. Runtime (min) | 59 | 18 | 11 | 19 | 253 | 229 | 140 | 367 |
| P95 Runtime (min) | 118 | 43 | 20 | 53 | 530 | 458 | 382 | 541 |
| Avg. FoM Performance* | 0.1983 | 0.2176 | 0.2181 | 0.2168 | 0.02732 | 0.03230 | 0.03840 | 0.03100 |

* FoM = Energy efficiency x Compute efficiency, unit: TOPS²/W/mm²

图 7. 案例研究 2：在不进行设计空间剪枝的情况下，使用不同优化策略时，(a) ResNet-50 和 (b) Swin-T 的最佳 FoM 随迭代次数的变化。阴影区域表示 50 次运行的 95% 自助法置信区间。

平均也需要数小时，最糟的 5% 运行中甚至需要 6–7 小时，才能找到 FoM 最优配置。这些结果表明，现有启发式算法虽然适用于基于卷积神经网络的工作负载，但其运行时间开销对于视觉变换器优化而言已难以接受。这一观察促使我们引入专为基于变换器的存内计算优化设计的设计空间剪枝技术，并在后续案例研究中对其进行评估。

- 关键要点：视觉 Transformer 工作负载（例如 Swin-T）由于设计空间更大、仿真延迟更长，其优化运行时间比卷积神经网络（例如 ResNet-50）更长。要提高 Transformer 的 PPA 优化收敛速度，必须采用标准启发式优化之外的算法级改进。

### 4.4 案例研究 3：不同算法下有无设计空间剪枝时对 Swin-T 的优化

在本案例研究中，我们评估了所提出的设计空间剪枝策略对 Swin Transformer Tiny（Swin-T）优化的有效性。我们比较了采用和未采用设计空间剪枝时，使用三种启发式算法（即遗传算法、模拟退火和树结构 Parzen 估计器）进行 FoM 优化和能效优化的运行时间与性能。应用剪枝时，我们考虑了两个基础模型：ResNet-50 和 Vision Transformer Base（ViT-B）。对于每个基础模型，我们假设对应设计空间的功耗、性能和面积（PPA）已完成预仿真并存储在数据库中。在初始剪枝阶段（算法 2），优化器会分析基础模型的性能分布，并在优化开始前据此对 Swin-T 设计空间进行剪枝。

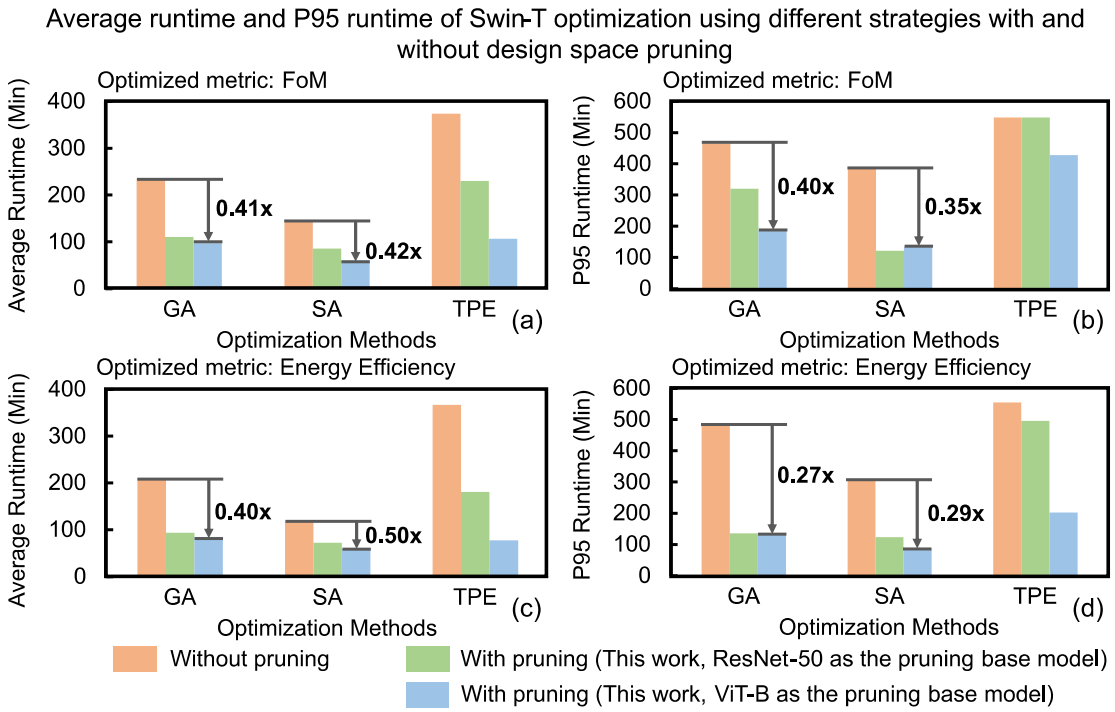



图 8. 案例研究 3：在有无设计空间剪枝条件下，采用不同策略进行 Swin-T 优化的平均运行时间 (a,c) 与 P95 运行时间 (b,d)。（面积约束：25 $cm^2$, 功耗约束：200 mW，优化迭代：80）

图 8 展示了 FoM 优化和能效优化的运行时间。对于 FoM 优化，与未剪枝的基线相比，基于 ViT-B 的设计空间剪枝策略使用模拟退火时，平均运行时间降低了 0.42 倍，P95 运行时间降低了 0.35 倍。类似地，对于能效优化，同一剪枝策略使平均运行时间降低了 0.50 倍，P95 运行时间降低了 0.29 倍。在所有启发式算法中，无论是否采用设计空间剪枝，模拟退火始终收敛最快，在平均运行时间和 *P95* 运行时间方面均优于遗传算法和树结构 Parzen 估计器。比较两种基础模型可见，对于两个优化目标，基于 ViT-B 的设计空间剪枝在三种启发式算法中均能稳定缩短运行时间。相比之下，基于 ResNet-50 的剪枝策略相较于无剪枝基线，仅能有限地缩短树结构 Parzen 估计器的运行时间。这种现象可能源于：（1）不同算法采用的候选采样策略存在差异；（2）在被剪枝的设计空间中，算法早期收敛后于高性能区域出现局部停滞。

我们进一步评估在固定迭代预算下优化后性能指标的质量。FoM 值经过归一化，以全局最优值为基准。我们报告平均 FoM 和 P5 FoM，其中 P5 FoM 表示 95% 的优化运行所获得的 FoM 值高于该阈值。如图 9 所示，无论基于 ResNet-50 还是 ViT-B 进行设计空间剪枝，在所有算法—迭代配置下，平均 FoM 和 P5 FoM 均始终高于不剪枝的基线。在三种算法中，模拟退火在 20 次和 40 次迭代限制下实现了最高的优化性能。值得注意的是，尽管采用设计空间剪枝的树结构 Parzen 估计器并不总能显著缩短运行时间（图 8），但与

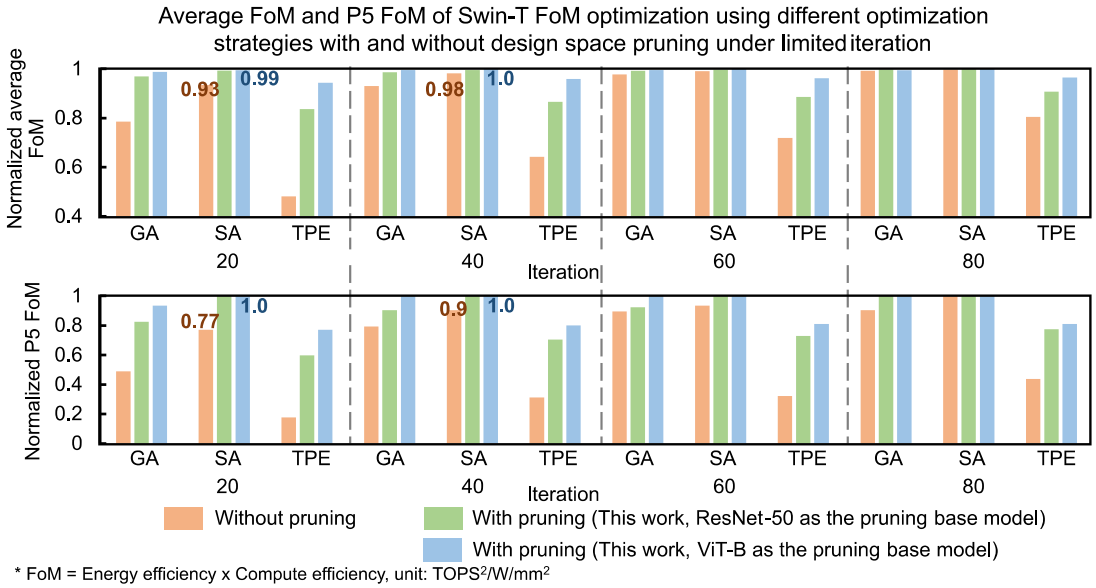



图 9. 案例研究 3：在有无设计空间剪枝的情况下，采用不同优化策略对 Swin-T 进行 FoM 优化的平均 FoM 和 P5 FoM t
设计空间剪枝条件下的表现，迭代次数有限。（面积约束：25 $cm^2$, 功耗约束：200 mW）

相应的无剪枝基线相比，它始终能找到质量更高的解。这些结果表明，在这些情况下，设计空间剪枝不仅能加快优化速度，还能在优化迭代次数受限时提升解的质量。本案例研究的主要结论如下：

- 无论是否应用剪枝，在 *FoM* 优化和能效优化中， *P*95 模拟退火的平均值和运行时间始终低于遗传算法和树结构 *Parzen* 估计器。
- 与不进行剪枝的优化相比，基于视觉 *Transformer-B*的设计空间剪枝可在所有评估的算法和目标下持续缩短运行时间。
- 在迭代预算有限的情况下，基于 *ResNet-50* 和视觉 *Transformer-B* 的设计空间剪枝可提升所有三种启发式算法的平均 *FoM* 和 *P5 FoM*。

### 4.5 案例研究 4：不同硬件约束下采用和不采用设计空间剪枝的 Swin-T FoM优化

由于模拟退火（SA）的搜索效率高于遗传算法（GA）和树结构 Parzen 估计器（TPE），因此在接下来的三个案例研究中，我们采用 SA 作为优化算法。案例研究 4 评估了所提出的设计空间剪枝策略在硬件约束范围扩展时的普适性。图 10 报告了在五种不同的功耗和面积约束配置下，Swin-T 的 FoM 优化平均运行时间和 P95 运行时间。当硬件约束放宽时，不进行剪枝的基线优化收敛更快，因为设计空间中更大比例的区域变为可行区域，而且更容易找到高性能配置。

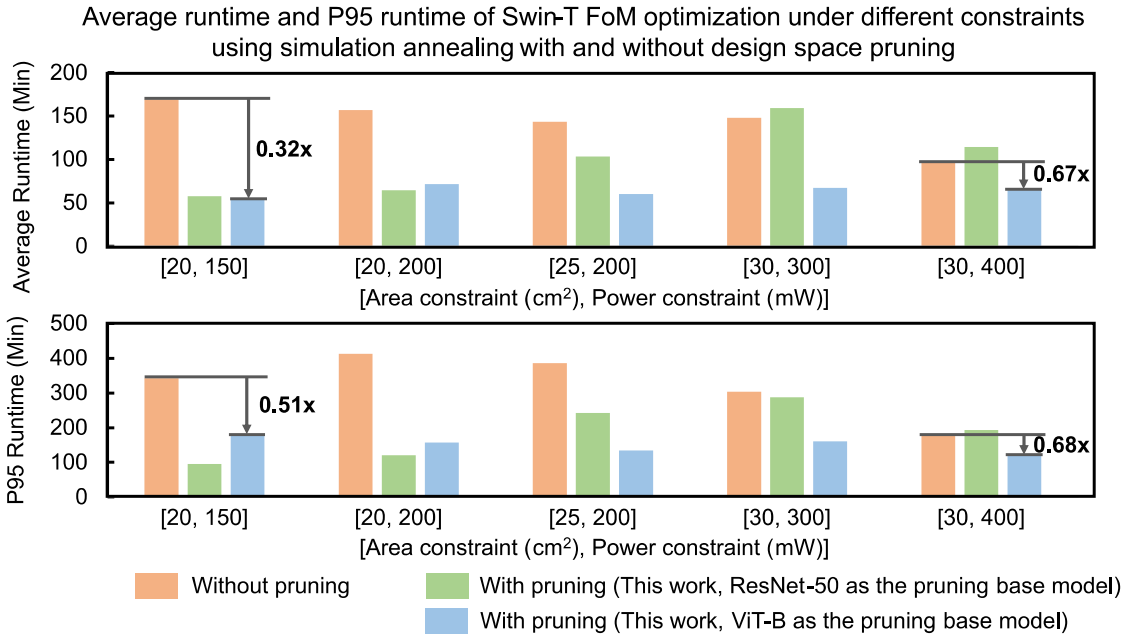



图 10. 案例研究 4：在不同约束下，使用模拟退火进行 Swin-T FoM优化时的平均运行时间和 P95 运行时间，比较有无设计空间剪枝的情况。（优化迭代：80）

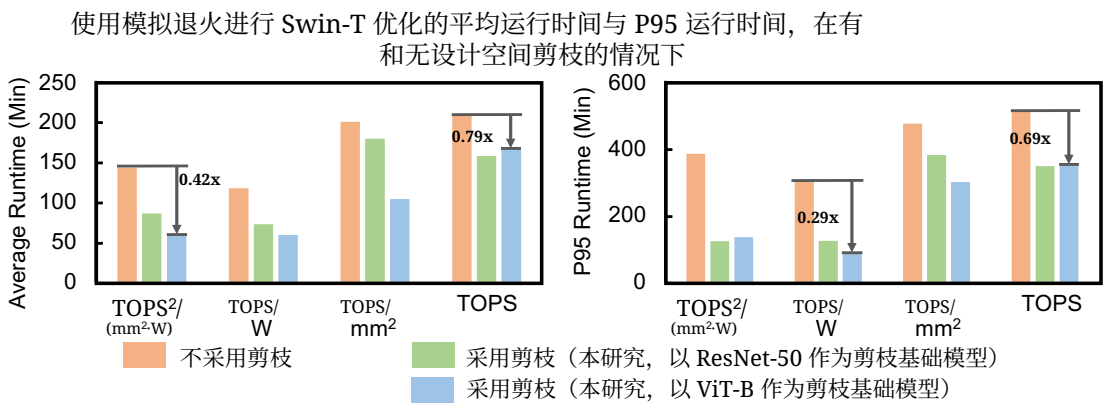
基于 ResNet-50 进行设计空间剪枝时，在约束较严格的情况下可以观察到运行时间有所改善。然而，在约束较宽松时，剪枝带来的收益可能有限，甚至会降低运行时间表现。这种现象可归因于以下几个因素：（1）将约束投影外推至显著放宽的约束时，准确率下降；（2）在被剪枝的设计空间中的局部高性能区域过早停滞；（3）优化过程中，随机反剪枝机制（算法3）带来了额外的验证开销。

相比之下，基于 ViT-B 的设计空间剪枝在所有评估的约束配置下均能缩短运行时间。这一观察表明，Swin-T 与 ViT-B 中与 FoM 相关的设计空间具有高度相似性。与无剪枝基线相比，在五种约束配置下，基于 ViT-B 的设计空间剪枝可使平均运行时间缩短 0.32x–0.67x，并使 P95 运行时间缩短 0.51x–0.68x。

- 核心要点： 基于 ViT-B 的设计空间剪枝在广泛的硬件约束下，始终能够加速 Swin-T FoM优化，体现出相较于不进行剪枝的优化方法更强的通用性和鲁棒性。

### 4.6 Case 研究 5：采用和不采用设计空间剪枝时对 Swin-T 进行不同指标的优化

在本案例研究中，我们评估了所提出的设计空间剪枝策略在不同优化目标下的通用性。具体而言，我们考察了存内计算系统中四种常用的性能指标：FoM、能效、计算效率和吞吐量。所有实验均基于 Swin Transformer Tiny（Swin-T）开展，并分别采用带设计空间剪枝和不带设计空间剪枝的模拟退火方法。




* 面积约束：25 cm², 功耗约束：200 mW，优化迭代：80
** 指标及单位：面积：cm²；功耗：mW，吞吐量：TOPS；计算效率：TOPS/mm²；能效：每瓦万亿次操作，FoM=计算效率×能效：TOPS²/(mm²·W)

Swin-T 22纳米CIM 性能优化总结

| 指标单位 | 最优配置 | cm² | mW | TOPS | TOPS/mm² | TOPS/W |
|---|---|---|---|---|---|---|
| TOPS²/(mm²·W) | 铁电场效应晶体管，64x256，3b 逐次逼近型，16 列/模数转换器，256x64 | 14.7 | 178 | 3.51 | 2.39e-3 | 16.1 |
| 每瓦万亿次操作 | 铁电场效应晶体管，64x256，3b 逐次逼近型，4 列/模数转换器，256x64 | 21.1 | 167 | 3.70 | 1.75e-3 | 18.2 |
| TOPS/mm² | 铁电场效应晶体管，32x128，3b 逐次逼近型，32 列/模数转换器，128x128 | 10.4 | 197 | 2.71 | 2.62e-3 | 11.3 |
| TOPS | 静态随机存取存储器，64x128，3b 逐次逼近型，4 列/模数转换器，128x64 | 23.5 | 196 | 3.96 | 1.68e-3 | 16.6 |

* 配置：单元、模拟存内计算子阵列大小、模数转换器、每个 ADC 的列数、数字存内计算子阵列大小
** 操作次数由 8bx8b 乘加运算定义。

图 11. 案例研究5：采用模拟退火优化 Swin-T 时，在有无设计空间剪枝两种情况下的平均运行时间和 P95 运行时间，以及 Swin-T 22纳米CIM 性能优化总结。

图 11 报告了各项指标的优化运行时间。对于全部四个优化目标，基于 ResNet-50 或 Swin-T 集成设计空间剪枝后，确定最优配置所需的运行时间均持续减少。具体而言，与无剪枝基线相比，基于 ViT-B 的设计空间剪枝使平均运行时间降低至原来的 0.42–0.79 倍，使 P95 运行时间降低至原来的 0.29–0.69 倍。图 11 的汇总结果还列出了每个优化目标下针对 Swin-T 获得的最优 22 纳米存内计算配置及其相应的性能指标。这些结果表明，所提出的剪枝框架能够有效推广到单一指标优化之外，从而依据多样化的存内计算性能标准实现高效的设计空间探索。

- 核心要点：基于 ResNet-50 或 ViT-B 进行设计空间剪枝，均能在多个性能目标下持续加速 Swin-T 优化，证明了所提出优化框架在指标层面的普适性。

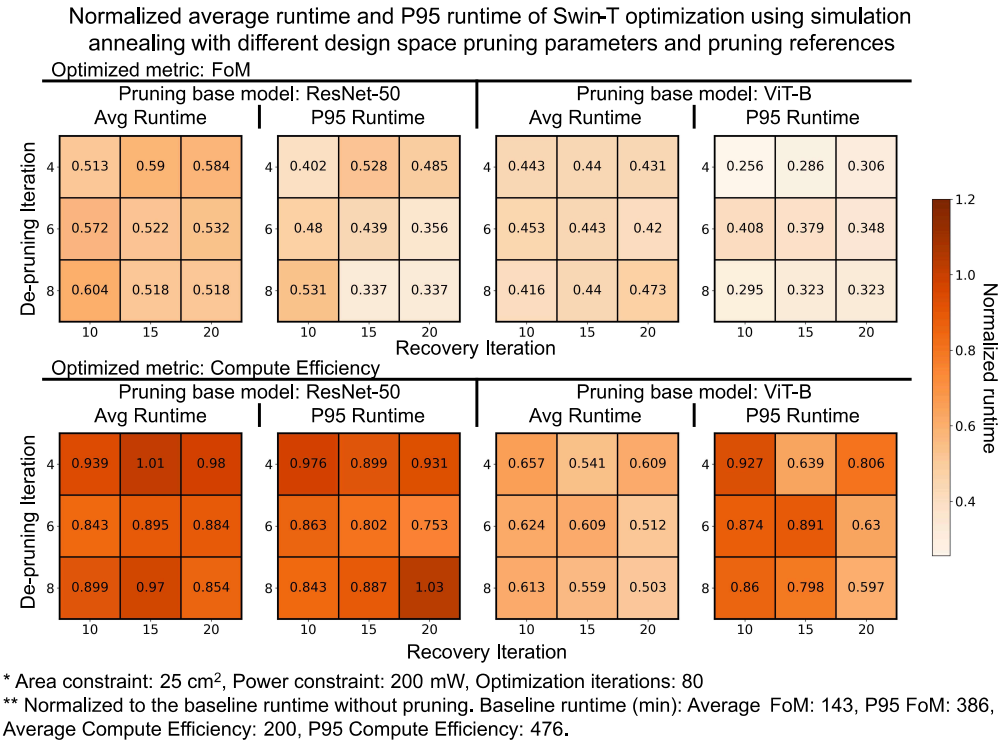



图 12. 案例研究 6：使用模拟退火对 Swin-T 进行优化时，在采用不同设计空间剪枝参数和剪枝基础模型的情况下，归一化平均运行时间和 P95 运行时间。

### 4.7 案例研究 6：设计空间剪枝的超参数设计空间

在最后一个案例研究中，我们分析了所提出的设计空间剪枝框架中超参数的选择对优化运行时间的影响。图 12报告了 Swin-T 的 FoM 和计算效率优化的平均运行时间与 P95 运行时间，并将其归一化为相应的未剪枝基线运行时间。我们重点关注两个关键超参数：反剪枝迭代和恢复迭代次数（图6）。反剪枝迭代指定随机去剪枝终止的迭代次数。在此之前，反剪枝算法会按固定间隔调用（例如，案例研究2–6中每两次迭代调用一次），以逐步恢复被排除的参数值。反剪枝停止后，优化器会继续在最近一次反剪枝后的设计空间内搜索，直到达到恢复迭代次数，此时便恢复完整的原始设计空间。

对于基于 ResNet-50 进行设计空间剪枝的 FoM 优化，在恢复迭代次数为 10 的情况下，将反剪枝迭代从 4 次增加到 8 次会导致运行时间延长。这一现象表明，优化器在反剪枝阶段无法在剪枝后的空间内确定最优配置，并且由于剪枝后的空间搜索不足，很快转而搜索完整设计空间。此外，反剪枝阶段的延长还会带来额外的验证开销，这是因为反剪枝过程引入了更多仿真。

同样地，在反剪枝迭代次数为 4 的情况下，将恢复迭代次数从 10 增加到 15 会导致运行时间延长，因为优化器会在较长时间内继续探索一个排除了最优配置的被剪枝空间。在

较大的反剪枝迭代次数（6 或 8）下则相反，增加恢复迭代次数会缩短运行时间，因为被剪枝的设计空间已在反剪枝过程中得到充分验证和调整。

在所有基于 ResNet-50 评估的配置中，4 次反剪枝迭代和 10 次恢复迭代的组合，相较于无剪枝基线实现了最大的平均运行时间缩短。该配置对应一种激进的剪枝策略：优化器在早期迭代中充分利用从基础模型迁移的知识；如果迁移的经验证明无效，则迅速恢复完整设计空间。相比之下，8 次反剪枝迭代和 20 次恢复迭代的配置实现了最低的 P95 运行时间。这种保守策略强调在进入被剪枝的设计空间进行长期探索之前，充分验证迁移的知识，从而降低接近最坏情况的运行时间。

在优化计算效率时，设计空间剪枝带来的收益不如优化 FoM 时显著。以 ResNet-50 作为基础模型时，平均运行时间缩短最大为 0.843x，P95 运行时间缩短最大为 0.802x。在某些配置中，运行时间甚至超过了无剪枝基线。这一现象表明，对于计算效率而言，由 ResNet-50 得到的被剪枝的设计空间通常不包含最优的 Swin-T 配置，这是因为两者的搜索空间特征存在差异。此外，反剪枝引入的验证开销进一步削弱了潜在的运行时间收益。

相比之下，以 ViT-B 作为基础模型可带来更多改进，实现 0.503 倍的平均运行时间缩短和 0.597 倍的 P95 运行时间缩短。当恢复迭代次数设为 20、反剪枝迭代设为 6 或 8 时，与无剪枝基线相比，运行时间可得到有效缩减。这些结果表明，ViT-B 与 Swin-T 之间具有更高程度的设计空间相似性，从而能够在不同性能目标之间实现有效的知识迁移。

### 4.8 案例研究 2-6 与设计指南总结

基于案例研究 2-6 的实验结果，我们总结了在 CIM 优化中应用设计空间剪枝的以下实用指南：

- 对于卷积神经网络（例如 *ResNet-50*），优化运行时间通常较短，设计空间剪枝并非严格必要。对于视觉 *Transformer*（例如 *Swin-T*），其涉及更大的设计空间和更长的仿真时间，设计空间剪枝在加速优化方面非常有效。
- 无论是否采用设计空间剪枝，模拟退火的收敛速度始终快于遗传算法和树结构 *Parzen* 估计器。

- 根据实证评估，在 *32* 核并行条件下，建议将优化预算设为约 *40* 次迭代。请注意，该数值可根据可用计算资源和时间约束进行调整。
- 对于视觉变换器优化，选择架构特征相似的基础模型（例如，为 *Swin-T* 选择 *ViT-B*），对于在不同硬件约束和优化目标下有效地进行设计空间剪枝至关重要。

- 反剪枝和恢复迭代次数会显著影响优化运行时间。应根据探索效率与鲁棒性之间所需的权衡，谨慎设定这些参数。

所有这些指南都已集成到 ChatNeuroSim 中，旨在帮助用户在硬件约束下针对存内计算功耗、性能和面积选择合适的优化策略。通过自动化仿真器交互、参数管理以及结合自适应设计空间剪枝的优化，ChatNeuroSim 大幅减少了存内计算设计工作量，并能够快速识别高性能硬件配置。

## 5 结论

本研究提出了 ChatNeuroSim，这是一种用于自动化存内计算加速器部署与优化的基于大语言模型的智能体框架。该框架通过任务解析、参数解析和参数调整智能体协同构成的工作流，支持四种类型的存内计算请求。我们使用定制的存内计算请求数据集验证了智能体系统的正确性和鲁棒性，一次性脚本生成的通过率达到 100%。此外，我们提出了一种采用自适应设计空间剪枝的 CIM优化器，可在优化新目标工作负载的功耗、性能和面积时，迁移并验证从先前基础模型中获得的优化经验。在大量案例研究中，与不采用设计空间剪枝的基线优化相比，所提出的设计空间剪枝策略将平均优化运行时间降低了 0.42x–0.79x，并将 P95 运行时间降低了 0.29x–0.69x。我们还分析了优化目标、硬件约束、基础模型选择和剪枝超参数的影响，并将由此得出的设计指南整合到 ChatNeuroSim 中。

通过自动化仿真器交互、参数依赖处理和 PPA 优化，ChatNeuroSim 减少了解读模拟器文档和管理复杂设计配置所需的人工工作。所提出的设计空间剪枝技术进一步提升了搜索效率，在不牺牲解的质量的前提下加快了设计空间探索。ChatNeuroSim 框架及其优化方法显著减少了存内计算设计空间探索所需的人力和时间，助力高性能存内计算加速器的快速、高效开发。ChatNeuroSim 用户界面已向研究社区开源，并为感兴趣的用户提供试用机会（https://github.com/neurosim/ChatNeuroSim）。

## 参考文献

[1] Tanner Andrulis、Joel S Emer 和 Vivienne Sze。2024。CiMLoop：一种灵活、准确且快速的存内计算建模工具。发表于 *2024IEEEInternational Symposium on Performance Analysis of Systemsand Software (ISPASS)*。IEEE，10–23。[2] James Bergstra、Rémi Bardenet、约书亚·本吉奥和 Balázs Kégl。2011。超参数优化算法。神经信息处理系统进展 24（2011）。[3] Tom B. Brown、Benja min Mann、Nick Ryder、Melanie Subbiah、Jared Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell 等。2020。语言模型是少样本学习者。发表于神经信息处理系统进展，第33卷。1877–1901。[4] DeepSeek-AI。2025。DeepSeek-V3.2：推动开放大语言模型的前沿。arXiv:2512.02556 [cs.CL] https://arxiv.org/abs/2512.02556[5] Alexey Dosovitskiy。2020。一张图像胜过 16×16 个词：用于大规模图像识别的 Transformer。*arXiv* 预印本 *arXiv:2010.11929*（2020）。[6] Stefan Dünkel、Martin Trentzsch、Regina Richter、Patrick Moll、Christine Fuchs、Oliver Gehring、Mateusz Majer、Stefan Wittek、B Müller、Thomas Melde 等。2017。一种基于铁电场效应晶体管的超低功耗、超高速嵌入式非易失性存储器技术，适用于22纳米全耗尽绝缘体上硅及更先进工艺。发表于*2017 IEEE InternationalElectronDevicesMeeting (IEDM)*。IEEE，19–7。[7] 何恺明、张祥雨、任少卿和孙剑。2016。用于图像识别的深度残差学习。发表于*IEEE*计算机视觉与模式识别会议论文集。770–778。[8] Chia-Tung Ho、Haoxing Ren 和 Brucek Khailany。2025。VerilogCoder：具备基于图规划和基于抽象语法树（AST）的波形追踪工具的自主 Verilog 编码智能体。发表于*AAAI*人工智能会议论文集，第39卷。300–307。[9] LangChain Inc.。2024。*LangGraph*：使用大语言模型构建有状态的多智能体应用。https://github.com/langchain-ai/langgraph[10] Streamlit Inc.。2024。*Streamlit*：构建和共享数据应用的最快方式。https://github.com/streamlit/streamlit[11] Pulkit Jain、Umut Arslan、Meenakshi Sekhar、Blake C Lin、Liqiong Wei、Tanaya Sahu、Juan Alzate-Vinasco、Ajay Vangapaty、Mesut Meterelliyoz、Nathan Strutt 等。2019。13.2 一种采用自适应成形/置位/复位方案的 3.6 Mb、10.1 Mb/mm² 嵌入式非易失性阻变存储器宏，采用22纳米鳍式场效应晶体管技术，工作电压低至0.5 V，在0.7 V下感测时间为5 ns。发表于*2019 IEEEInternational Solid-State Circuits Conference-(ISSCC)*。IEEE，212–214。[12] Sheng-Chun Kao、Geonhwa Jeong 和 Tushar Krishna。2020。Confuciux：使用强化学习为深度神经网络加速器自主分配硬件资源。发表于*202053rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*。IEEE，622–636。[13] Sheng-Chun Kao 和 Tushar Krishna。2020。Gamma：通过遗传算法自动完成加速器上的深度神经网络模型硬件映射。发表于第*39*届计算机辅助设计国际会议论文集。1–9。[14] Scott Kirkpatrick、C Daniel Gelatt Jr. 和 Mario P Vecchi。1983。通过模拟退火进行优化。《科学》220，4598（1983），671–680。[15] Aviral Kumar、Amir Yazdanbakhsh、Milad Hashemi、Kevin Swersky 和 Sergey Levine。2021。面向硬件加速器架构设计的数据驱动离线优化。*arXiv* 预印本 *arXiv:2110.11346*（2021）。

[16] Manuel Le Gallo、Corey Lammie、Julian Büchel、Fabio Carta、Omobayode Fagbohungbe、Charles Mackin、Hsinyu Tsai、Vijay Narayanan、Abu Sebastian、Kaoutar El Maghraoui 等。2023。使用 IBM 模拟存内硬件加速套件进行神经网络训练与推理。《APL Machine Learning》1，4（2023）。[17] Junmo Lee、Anni Lu、Wantong Li 和 SHIMENG YU。2024。NeuroSim v1.4：将数字存内计算的技术支持扩展至1纳米节点。《IEEE电路与系统汇刊I：常规论文》（2024）。[18] MING-YEN LEE 和 SHIMENG YU。2025。NeuroSim Agent：采用可迁移强化学习与动态设计空间剪枝的存内计算加速器自动部署。发表于2025年ACM/IEEE第7届计算机辅助设计机器学习研讨会（MLCAD）。IEEE，1–7。[19] Patrick Lewis、Ethan Perez、Aleksandra Piktus、Fabio Petroni、Vladimir Karpukhin、Naman Goyal、Heinrich Küttler、Mike Lewis、Wen-tau Yih、Tim Rocktäschel、Sebastian Riedel 和 Douwe Kiela。2020。面向知识密集型自然语言处理任务的检索增强生成。发表于《神经信息处理系统进展》，第33卷。9459–9474。[20] Wantong Li、Xiaoyu Sun、Shanshi Huang、Hongwu Jiang 和 SHIMENG YU。2022。一种40纳米多层单元阻变存储器存内计算宏，具备稀疏性控制、片上写入校验和温度无关的模数转换器参考电压。《IEEE固态电路期刊》57，9（2022），2868–2877。[21] Ze Liu、Han Hu、Yutong Lin、Zhuliang Yao、Zhenda Xie、Yixuan Wei、Jia Ning、Yue Cao、Zheng Zhang、Li Dong 等。2022。Swin Transformer v2：扩展容量与分辨率。发表于《IEEE/CVF计算机视觉与模式识别会议论文集》。12009–12019。[22] Anni Lu、Junmo Lee、Tae-Hyeon Kim、Muhammed Ahosan Ul Karim、Rebecca Sejung Park、Harsono Simka 和 SHIMENG YU。2024。面向人工智能硬件加速器的高速新兴存储器。《Nature Reviews Electrical Engineering》1，1（2024），24–34。[23] Volodymyr Mnih、Koray Kavukcuoglu、David Silver、Alex Graves、Ioannis Antonoglou、Daan Wierstra 和 Martin Riedmiller。2013。使用深度强化学习玩 Atari 游戏。arXiv 预印本 arXiv:1312.5602（2013）。[24] Luigi Nardi、David Koeplinger 和 Kunle Olukotun。2019。实用设计空间探索。发表于2019年IEEE第27届计算机与通信系统建模、分析与仿真国际研讨会（MASCOTS）。IEEE，347–358。[25] OpenAI。2025。GPT-5 mini。https://openai.com/index/gpt-5-mini/。[26] OpenAI。2025。GPT-5.1。https://openai.com/index/gpt-5-1/。[27] Angshuman Parashar、Priyanka Raina、Yakun Sophia Shao、Yu-Hsin Chen、Victor A Ying、Anurag Mukkara、Rangharajan Venkatesan、Brucek Khailany、Stephen W Keckler 和 Joel Emer。2019。Timeloop：一种系统化的深度神经网络加速器评估方法。发表于2019年IEEE系统与软件性能分析国际研讨会（ISPASS）。IEEE，304–315。[28] Xiaochen Peng、Shanshi Huang、Yandong Luo、Xiaoyu Sun 和 SHIMENG YU。2019。深度神经网络+ NeuroSim：一种支持多种器件技术的存内计算加速器端到端基准测试框架。发表于2019年IEEE国际电子器件会议（IEDM）。IEEE，32–5。[29] Ritik Raj、Sarbartha Banerjee、Nikhil Chandra、Zishen Wan、Jianming Tong、Ananda Samajdhar 和 Tushar Krishna。2025。SCALE-Sim v3：一种用于端到端系统分析的模块化、周期精确脉动阵列加速器模拟器。发表于2025年IEEE系统与软件性能分析国际研讨会（ISPASS）。IEEE，186–200。[30] James Read、MING-YEN LEE、Wei-Hsing Huang、Yuan-Chun Luo、Anni Lu 和 SHIMENG YU。2025。NeuroSim V1.5：用于对具有器件和电路级非理想特性的存内计算加速器进行基准测试的改进型软件基础。arXiv 预印本 arXiv:2505.02314（2025）。[31] John Schulman、Filip Wolski、Prafulla Dhariwal、Alec Radford 和 Oleg Klimov。2017。近端策略优化算法。arXiv 预印本 arXiv:1707.06347（2017）。[32] Jamin Seo、Akshat Ramachandran、Yu-Chuan Chuang、Anirudh Itagi 和 Tushar Krishna。2025。AIRCHITECT v2：通过统一表示学习硬件加速器设计空间。arXiv 预印本 arXiv:2501.09954（2025）。[33] Janak Sharda、Po-Kai Hsu 和 SHIMENG YU。2024。采用三维堆叠无电容DRAM为大语言模型设计加速器。发表于2024年IEEE第6届人工智能电路与系统国际会议（AICAS）。IEEE，487–491。[34] Richard S Sutton、David McAllester、Satinder Singh 和 Yishay Mansour。1999。基于函数逼近的强化学习策略梯度方法。《神经信息处理系统进展》12（1999）。[35] Hado Van Hasselt、Arthur Guez 和 David Silver。2016。基于双Q学习的深度强化学习。发表于《AAAI人工智能会议论文集》，第30卷。[36] Christopher JCH Watkins 和 Peter Dayan。1992。Q学习。《机器学习》8（1992），279–292。[37] Haoyuan Wu、Zhuolun He、Xinyun Zhang、Xufeng Yao、Su Zheng、Haisheng Zheng 和 Bei Yu。2024。ChatEDA：一种由大语言模型驱动的电子设计自动化自主智能体。《IEEE集成电路与系统计算机辅助设计汇刊》43，10（2024），3184–3197。[38] Zheng Wu、Zhuochu Yang、Zhuoyuan Yang、Zihao Chen、Li Shang 和 Fang Yan。2025。ChatArch：一种面向处理器架构优化的知识驱动思维图大语言模型框架。《ACM电子系统设计自动化汇刊》31，2（2025），1–26。[39] Amir Yazdanbakhsh、Christof Angermueller、Berkin Akin、Yanqi Zhou、Albin Jones、Milad Hashemi、Kevin Swersky、Satrajit Chatterjee、Ravi Narayanaswami 和 James Laudon。2021。Apollo：可迁移的架构探索。arXiv 预印本 arXiv:2102.01723（2021）。[40] Yujie Zhao、Hejia Zhang、Hanxian Huang、Zhongming Yu 和 Jishen Zhao。2025。Mage：用于自动生成寄存器传输级代码的多智能体引擎。发表于2025年ACM/IEEE第62届设计自动化会议（DAC）。IEEE，1–7。

**附录**

**A 设计空间剪枝算法**

三种算法的工作流程如图6所示，实现细节则在本节中给出。

**A.1 算法：基于幂律拟合的跨空间约束投影**

**算法1：** 基于幂律拟合的跨空间约束投影

**输入：** 基础设计空间 $\mathrm{D}_b$，目标设计空间 $\mathrm{D}_t$；基础数据集 $\mathrm{S}_b$；约束集 C；采样预算 $N$。
**输出：** 设计空间交集 I；所有存储器类型 $m \in \mathrm{M}$ 和约束的拟合系数 $\{(a^{m}_{,0}{}^{c}, a^{m}_{,1}{}^{c})\}$ 约束 $c \in \mathrm{C}$。

$\mathrm{I} \leftarrow$ 交集($\mathrm{D}_b, \mathrm{D}_t$)；// 步骤 1：选择共享参数空间
$\mathrm{M} \leftarrow \mathrm{I}$[存储单元类型]；// 常见存储单元类型
// 提取代表性采样的边界值
$\mathrm{V} \leftarrow$ {最小最大值(I[$k$]) | $k \in$ {ACIM行, ACIM列, ADC类型, ADC级别, ADC多路复用器列, DCIM行, DCIM列}};
**对于** $m \in \mathcal{M}$ **执行**
  // 步骤 2：生成候选样本集 $\Omega_m$ 基于 V 中的所有参数组合
  $\Omega_m \leftarrow \{(m, r^a, c^a, t, l, u, r^d, c^d) \mid r \in \{V[rowACIM]_{min}, V[rowACIM]_{max}\} \ldots\}$;
  $\mathrm{T}_m$，$\mathrm{B}_m \leftarrow$ 选择样本($\Omega_m$，$N$)；// 若 $|\Omega_m| > N$，则获取随机唯一子集
  // 步骤3：执行性能仿真
  $\mathrm{Perf}_t \leftarrow$仿真($\mathrm{T}_m$);
  $\mathrm{Perf}_b \leftarrow$ 获取($\mathrm{S}_b$，$\mathrm{B}_m$);
  **对每个** 约束 $c \in C$ **执行**
    // 构造用于拟合的成对数据（$X_i$，$Y_i$）
    $\mathbf{X}, \mathbf{Y} \leftarrow \emptyset$;
    **从** $i = 1$ **到** $|\mathcal{T}_m|$ **执行**
      $X_i \leftarrow$ 性能$_b$[$\mathrm{B}_m[i]$][$c$];
      $Y_i \leftarrow$ 性能$_t$[$\mathrm{T}_m[i]$][$c$];
      $\mathbf{X} \leftarrow \mathbf{X} \cup \{X_i\}, \mathbf{Y} \leftarrow \mathbf{Y} \cup \{Y_i\}$;
    **end**
    // 步骤 4：在对数-对数空间中进行线性回归拟合
    ($a^{m}_{,0}{}^{c}$，$a^{m}_{,1}{}^{c}$) $\leftarrow$最小二乘法(log(X)，log(Y));
  **end**
**结束**
**返回** $\mathcal{I}$，$\{(a^{m}_{,0}{}^{c}, a^{m}_{,1}{}^{c})\}$;

算法1旨在解决存在架构差异时，将性能约束从基础设计空间 $\mathrm{D}_b$映射到目标设计空间 $\mathrm{D}_t$ 这一挑战。通过建立解析跨模型映射，该算法将基础设计空间的约束（例如功耗和面积）投影到目标设计空间。因此，基础模型中投影性能违反用户定义约束的设计点将被筛除。其余设计点则会进入下一阶段的设计空间剪枝，进行进一步分析。

算法首先调用交集函数，以识别两个空间之间的共享参数维度（例如，存储单元类型、阵列维度、ADC分辨率），并将其记为交集空间 I。为确保拟合模型在整个设计范围内的鲁棒性，算法提取每个维度的边界值（最小值和最大值），以构建参数集 V。在采样预算 $N$（案例研究3-6 中为 $N = 32$）的约束下，从这些边界组合中随机抽取子集 $\mathrm{T}_m$，以覆盖用于投影构建的设计点。

接下来，针对所选样本点，算法对两个模型执行不同的 PPA 获取（$X_i$ 和 $Y_i$）。对于目标模型，调用仿真函数，以在目标环境中运行 NeuroSim 仿真。对于基础模型，我们假设设计空间的 PPA 已在目标模型优化之前预先完成仿真。相应的性能记录 $\mathrm{Perf}_b$ 直接从现有的基础数据集 $\mathrm{S}_b$中检索。

最后，鉴于硬件性能指标在不同技术间通常遵循非线性缩放定律，因此采用幂律模型，其公式如下：

$$Y = \alpha X^{\beta} \tag{4}$$

其中，$\alpha$和 $\beta$为空间特定的缩放因子。

$$\ln(Y^{m,c}) = a_1^{m,c} \ln(X^{m,c}) + a_0^{m,c} \tag{5}$$

$Y^{m,\,c}$，$X^{m,\,c}$分别表示目标模型和基础模型中的硬件约束 $c$和存储器类型 $m$对应的值。对变换后的数据应用最小二乘法进行线性回归。所得投影系数将在后续设计空间探索阶段用于快速投影基础空间中的约束，以指导目标空间中的搜索。

### A.2 算法：Top-K 设计空间剪枝

算法 2 利用算法1中得到的解析投影系数，对庞大的目标设计空间 $\mathrm{D}_t$进行剪枝。通过投影基础模型中各设计点的性能，可在有效搜索空间内识别基础设计空间中的高性能区域。这些经剪枝且搜索空间规模缩小的区域将被设为目标模型优化过程中的初始设计空间。

算法首先遍历交集设计空间 I 的笛卡尔积，以评估所有共享设计点。在算法1的假设下，每个点 $s$ 的基准性能 $P_{base}$ 从数据集 $\mathrm{S}_b$ 中检索得到。利用幂律拟合系数（$a_0, a_1$），算法通过以下公式计算目标空间中的投影性能$\hat{}P_{target}$：

$$\hat{P}_{target} = \exp(a_0^{m,c}) \cdot (P_{base})^{a_1^{m,c}} \tag{6}$$

任何违反投影约束（$\hat{}P_{target}$ > 阈值$_c$）的设计点都会被移除。因此，所得有效集合 $\Omega_{valid}$ 仅包含可能满足目标环境中硬件需求的设计点。

接下来，算法根据优化目标 $T_{opt}$，从有效集合 $\Omega_{valid}$中识别性能最优的 Top-K 设计。超参数 $\rho$（案例研究3-6中为$\rho = 0.2$）决定选择比例。随后，算法对这些精英样本执行统计频率分析（*CalculateFrequency*），以确定每个硬件参数 $p$的分布。该过程识别出具有高性能的“高概率”参数范围。

最后一步中，为弥合基准设计空间与目标设计空间之间参数绝对值的潜在差异，该算法采用基于分箱的映射策略。两个空间中每个参数的范围都被划分为离散的分箱（例如“小”“中”“大”）。基准空间中超过频率阈值 $\tau$（在案例研究3-6中为$\tau = 0.85$）的分箱会被标记为优选区域（$\mathrm{L}_{chosen}$）。随后对目标设计空间 $\mathrm{D}_t$ 进行剪枝，仅保留落在对应优选分箱内的值。最终输出 $\mathrm{D}_t^*$是一个经过收窄的设计空间，其重点是基于从基础模型分析得出的经验，探索“潜在”的高性能区域。

**算法 2：** Top-K 设计空间剪枝

**输入：** 优化目标 $T_{opt}$；共享交集设计空间 I；基础数据集 $S_b$；约束集 C；拟合系数 $\{(a^{m}_{\cdot 0}{}^{c}, a^{m}_{\cdot 1}{}^{c})\}$；超参数：Top-K比例 $\rho$，Top-bin边界 $\tau$。

**输出：** 剪枝后的目标设计空间 $D_t^*$。

// 步骤 1：对基础设计空间进行约束投影和有效性筛选

$\Omega_{valid} \leftarrow \emptyset$;

$S_{all} \leftarrow$笛卡尔积(I); // 枚举所有共享设计点

**对每个** $s \in \mathcal{S}_{all}$ **执行**

  $Perf_s \leftarrow$ 获取($S_b, s$);

  $meets_constraints \leftarrow$True;

  **对于每个**约束 $c \in C$ **执行**

    $m \leftarrow s[memCellType]$;

    $P_{base} \leftarrow$ 性能$_s[c]$ ; // 从基础数据集 S 查询性能$_b$

    $\hat{P}_{target} \leftarrow \exp(a^{m}_{\cdot 0}{}^{c}) \cdot (P_{base})^{a^{m,c}_{1}}$ ; // 应用幂律投影

    **if** $\hat{P}_{target}$ > 阈值 $_c$ **then**

      $meets_constraints \leftarrow$假;

      **跳出**;

    **end**

  **end**

  **如果** $meets_constraints$ **则**

    $\Omega_{valid} \leftarrow \Omega_{valid} \cup \{(s,$ 性能$_s[T_{opt}])\}$;

  **end**

**结束**

// 步骤 2：性能排序和频率聚合

$K \leftarrow \lceil \rho \cdot |\Omega_{valid}| \rceil$;

$\Omega_{topk} \leftarrow$ 按性能排序($\Omega_{valid}, T_{opt}$)[1 : $K$];

$Freq \leftarrow$ 计算频率($\Omega_{topk}$) ; // 每个参数的统计分布 $p$

// 步骤3：基于分箱的映射与剪枝

$\mathcal{D}_t^* \leftarrow \mathcal{D}_t$;

**遍历** 参数 $p \in \mathcal{I}$ **执行**

  $B_{base} \leftarrow$ DiscretizeBins(I[$p$]); // 分箱：{small, mid, large}

  $BinFreq \leftarrow$ 聚合($Freq[p]$,$B_{base}$); // 选择优选分箱

  $L_{chosen} \leftarrow$ TopBinSelection($BinFreq, \tau$) ;

  $B^{target} \leftarrow$ DiscretizeBins($D_t[p]$, 数量 = |分箱$_{base}$|);

  $D_t^*[p] \leftarrow \{v \mid v \in B_{target}[l], \forall l \in L_{chosen}\}$;

**结束**

**返回** $\mathcal{D}_t^*$;

### A.3 基于性能验证的随机设计空间反剪枝

算法3是对算法2所执行剪枝操作的纠正机制。前述剪枝虽然缩小了搜索空间规模，但由于幂律拟合或基于分箱的离散化存在固有近似，可能会舍弃高性能配置。该算法采用随机方法，根据性能验证结果，有选择地将有前景的参数值恢复到设计空间 $D_t^*$中。

算法首先通过计算完整目标设计空间 $D_t$ 与当前已剪枝空间 $D_t^*$ 之间的差集，确定缺失参数值集合 $V_{miss}$。为控制计算开销，分配给缺失值的验证预算总额为 $N_p$（$N_p = \lfloor N_{total}/|V_{miss}| \rfloor$，案例研究3-6中为 $N_{total} = 32$），以此确定用于评估恢复可能性的试验次数。

其次，对于参数 $p$ 的每个候选值 $v$，算法都会将其与当前性能最佳的基线样本 $S_{base}$进行性能比较。通过替换其中的

**算法3：**通过性能验证进行随机设计空间反剪枝

**输入：**当前被剪枝的设计空间 $\mathrm{D}_t^*$；完整的目标设计空间 $\mathrm{D}_t$；约束集 C；基线顶级性能样本 $\mathrm{S}_{base}$；验证预算 $N_{total}$；胜率缩放因子 $\gamma$。

**输出：**去剪枝后的设计空间 $\mathrm{D}^{dp}$。

// 步骤1：识别缺失的参数 (p) 值 (v)

$\mathrm{V}_{miss} \leftarrow \{(p,v) \mid v \in \mathrm{D}_t[p] \setminus \mathrm{D}_t^*[p], \forall p \in \mathrm{D}_t^*\}$;

$N_p \leftarrow \lfloor N_{total}/|\mathrm{V}_{miss}| \rfloor$; // 每个缺失值的验证样本

// 步骤2：蒙特卡罗验证与性能比较

$WinRate \leftarrow \emptyset, \mathcal{P}_{valid} \leftarrow \emptyset$;

**遍历** $(p,v) \in \mathcal{V}_{miss}$ **执行**

  $\mathrm{S}_{verify} \leftarrow \emptyset, \quad n_{win} \leftarrow 0$;

  $\Omega \leftarrow$ 选择样本$(\mathrm{S}_{base}, N_p)$;

  **对每个** $s_b \in \Omega$ **执行**

    $s_{new} \leftarrow$ 替换参数（$s_b, p, v$）；

    $P_{target} \leftarrow$仿真$(s_{new})$;

    **if** $P_{target}$ 满足所有约束$C$ **则**

      $\mathrm{P}_{valid} \leftarrow \mathrm{P}_{valid} \cup \{(s_{new}, P_{target})\}$;

      $P_{base} \leftarrow$ 性能为 $s_b$;

      **如果** 更优$(P_{target}[T_{opt}], P_{base}[T_{opt}])$ **则**

        $n_{win} \leftarrow n_{win} + 1$;

      **end**

    **end**

  **end**

  $r_{win} \leftarrow n_{win}/|\Omega|$;

  $WinRate[p][v] \leftarrow \min(r_{win} \cdot \gamma, 1.0)$; // 缩放胜率

**结束**

// 步骤3：概率设计空间恢复

$\mathcal{D}_{dp} \leftarrow \mathcal{D}_t^*$;

**对每个** $(p, v) \in \mathcal{V}_{miss}$ **执行**

  $\alpha \leftarrow$ 均匀随机$(0, 1)$;

  **如果** $\alpha < WinRate[p][v]$ **则**

    $\mathrm{D}_{dp}[p] \leftarrow \mathrm{D}_{dp}[p] \cup \{v\}$; // 恢复缺失值

  **end**

**结束**

**返回**$\mathcal{D}^{dp}$;

将基线样本 $s_b$ 中的参数 $p$ 替换为缺失值 $v$，即可生成新的设计配置 $s_{new}$。对 $s_{new}$进行 PPA仿真后，如果 $s_{new}$ 满足所有硬件约束，且性能优于基线 $s_b$的 $T_{opt}$，则将其记录为一次“胜出”。最终胜率 $r_{win}$ 表示缺失值有助于获得最优设计的经验概率。由于反剪枝过程会定期执行，因此会乘以因子 $\gamma$（$\gamma = 1.0$ 在首次函数调用时设定，后续调用中则通过常数乘数 3.0 更新），以控制恢复的力度。

最后，恢复过程由概率阈值决定。对于每个缺失值，算法生成一个均匀随机变量 $\alpha$。如果 $\alpha$小于计算出的胜率 $WinRate[p][v]$，则将该值恢复到去剪枝后的设计空间$\mathrm{D}^{dp}$。如图6的流程图所示，该过程可在特定迭代间隔（去剪枝间隔，在案例研究3-6中设为2）触发，或在达到某些恢复阈值（恢复迭代次数）时触发。该算法自适应地调节设计空间恢复与剪枝之间的平衡，并随着高性能区域的暴露动态调整搜索空间。

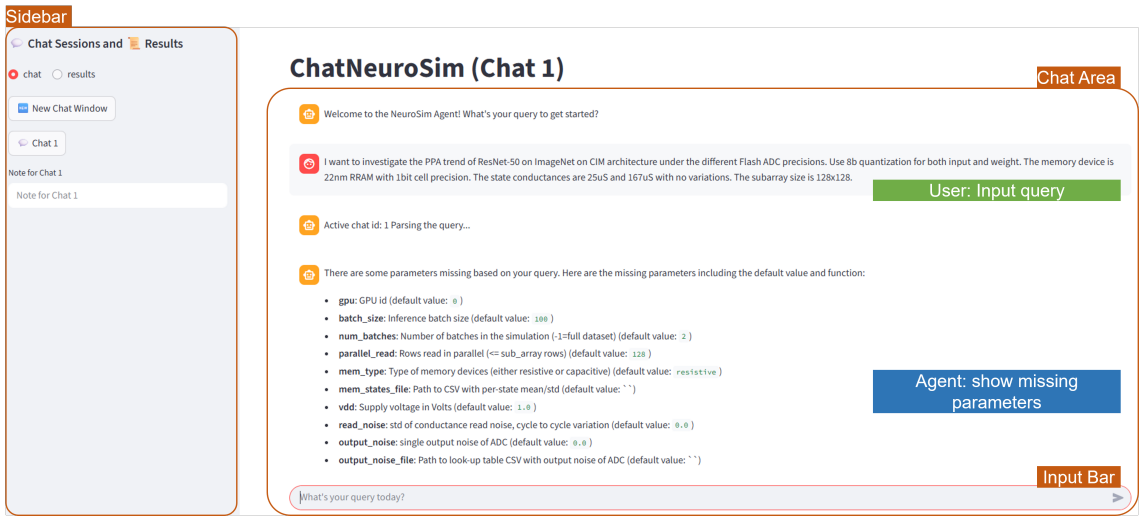



图13. 用户界面概览与对话示例。

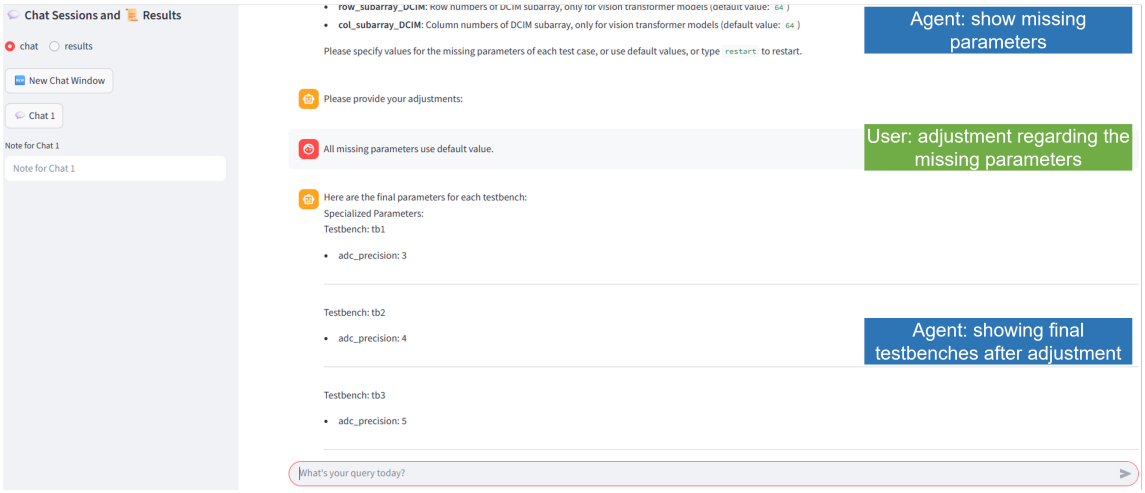



图14. 对话示例：用户针对缺失参数的调整。

## B ChatNeuroSim用户界面(UI)与对话示例

本附录部分展示了ChatNeuroSim用户界面以及一个对话示例。侧边栏、聊天区域和输入栏的用户界面概览如图13所示。在该示例中，用户输入测试平台自动设计请求，以检查不同模数转换器精度下的存内计算功耗、性能和面积。智能体检查缺失和无效参数，并将反馈发送给用户（图13和图14）。在用户提供调整后，智能体更改当前参数，并重新检查缺失和无效参数（图14）。在用户最终确认后，仿真成功执行（图15）。对于仿真结果，用户可切换到“结果”开关，查看所有配置及相应的功耗、性能和面积汇总（图16）。

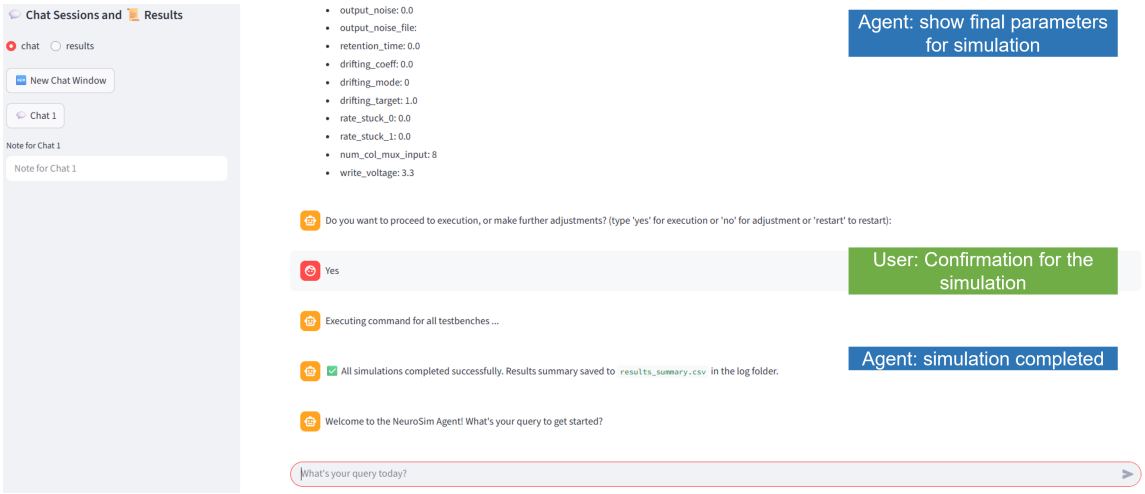



图15. 对话示例：仿真前的用户确认。

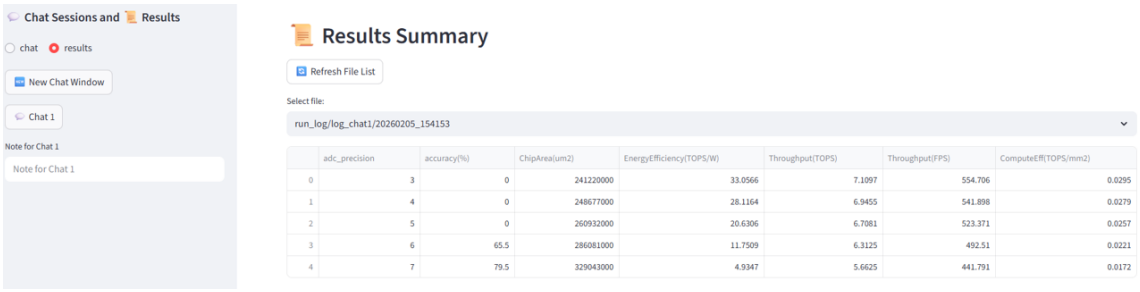



图16. 对话示例：仿真结果。