ASCEND-OK:

Adaptive Search with Constraint-Aware Evolution and NPU Diagnostics for Optimization of Triton Kernel

基于约束感知进化与 NPU 诊断的 Triton 算子自适应优化系统

技术设计与实验报告

队伍名称：没带龙书让 agent 自己干活

队　　员：欧阳易芃，陈龙，陈大有

指导老师：张献伟

学　　校：中山大学

摘要

Triton 算子的性能优化并非单纯的参数调节问题。一个候选程序必须同时满足数值语义、边界访存、编译器可接受性、NPU 执行资源与输入形状泛化等要求，还要在有限时间内找到真正有效的性能改进。大语言模型擅长生成跨语句、跨函数的程序变换，但它们也容易改变 mask、dtype、舍入顺序或 launch 契约。因此，仅靠一次生成无法构成稳定的自动优化系统。

本文提出 ASCEND-OK（Adaptive Search with Constraint-Aware Evolution and NPU Diagnostics for Optimization of Triton Kernel），一个面向 Triton-Ascend 和昇腾 A2/A3 平台的约束感知进化优化系统。系统为每个 Kernel 建立独立搜索上下文，将大模型程序变异、参数变异、保守结构变换与双父代交叉纳入统一候选谱系。候选先通过静态边界，再依次进入编译、全部正确性 Case、边界变体和新鲜 NPU 测量。只有完全正确的候选才能进入性能档案；profiler 用于产生下一代搜索所需的诊断信号，不替代测试结果。在输出阶段，系统自动保留不超过五个性能与错误模式具有差异的候选，并通过 manifest 绑定输入、模型、运行时间、候选哈希与评测状态。

在统一评测门户上，严格受约束的连续搜索阶段最高取得 **71** 分；扩大允许模型的探索范围并充分发挥候选池能力时，系统的实验峰值达到 **100** 分。最终稳健配置取得 70 分、204/225 个任务正确、20/21 个 Kernel 获得有效结果。实验表明，正确性单向门禁、错误签名驱动的多样性选择、源码结构诊断和分阶段评价能够明显提高有效搜索密度。报告详细给出问题形式化、系统架构、进化算法、NPU 诊断、实验分析和可复现执行流程。

关键词： Triton-Ascend；昇腾 NPU；程序进化；自动调优；大语言模型；正确性门禁；NPU 性能诊断

Abstract

Triton kernel optimization must simultaneously preserve numerical semantics, memory safety, compiler legality, shape generality, and efficient hardware mapping. We present ASCEND-OK, a constraint-aware evolutionary optimization system for Triton-Ascend on Ascend A2/A3 platforms. For every kernel, the system combines LLM-guided program mutation, parameter mutation, conservative structural transformation, and two-parent crossover in an isolated evolutionary run. A staged evaluator checks the static boundary, compilation, all correctness cases, optional boundary variants, and fresh NPU measurements in order. Only fully correct candidates enter the performance archive; profiler signals guide later mutations but never replace evaluator truth.

The system automatically exports at most five candidates with complementary performance and failure characteristics. On the unified evaluation portal, constrained continuous search reached 71 points. With a broader allowed-model exploration pool, the system reached an experimental peak of 100 points. A robust final configuration achieved 70 points, passed 204 of 225 tasks, and produced effective results for 20 of 21 kernels. The report describes the problem formulation, architecture, evolutionary algorithm, NPU diagnostics, experimental analysis, and end-to-end reproduction workflow.

Keywords: Triton-Ascend, Ascend NPU, program evolution, auto-tuning, large language model, correctness gate, NPU diagnostics

目录

插图

表格

第一章　引言

1.1　研究背景

大模型训练和推理系统由数以百计的张量算子组成。算子尺寸虽小，却直接决定了高频路径上的内存访问、并行度和数据移动。对于一个给定的数学表达式，tile 尺寸、program 划分、向量宽度、reduction 树形态以及 load/store 的排布方式都会改变最终性能。这些决策相互耦合，最佳组合又随输入形状、数据类型和硬件平台而变。

Triton 以类 Python 的领域特定语言表达块化张量计算，在可编程性与高性能之间建立了良好平衡 [3]。然而，Triton 降低的是 Kernel 的编写成本，而不是搜索最优实现的难度。TVM、Ansor 和 OpenTuner 等工作分别从张量编译、计算图生成与通用参数调优角度展示了自动搜索的价值 [4, 5, 6]。这类系统依靠预先设计的搜索空间和成本模型，对于完整程序结构的改写能力相对有限。

近年来，大语言模型开始被用于编译优化、程序生成和迭代搜索。Cummins 等人验证了语言模型生成编译器优化序列的能力 [8]；LLM Compiler 进一步将模型针对编译优化与汇编理解进行专项化 [9]。OPRO 表明语言模型可以把优化历史作为上下文，逐步提出更好的解 [10]；进化式程序综合则说明，在评价器能够稳定返回真实信号时，模型、变异和选择可以共同构成持续自我改进的闭环 [11]。但这些研究也留下一个关键问题：当目标程序必须在专用 NPU 后端编译和运行时，如何让开放式生成不会破坏正确性，又能产生足够的性能收益？

昇腾 NPU 上的 Triton-Ascend 将这一问题变得更具代表性。A2/A3 是昇腾计算平台，其中 A2 通常对应 910B 系列芯片；本项目的开发环境使用昇腾 910B3 芯片。这里的性能不只受高层 Triton 代码影响，还取决于 Triton-Ascend lowering、CANN runtime、AIV/AIC 管线以及实际输入形状。一段 Python 语法正确的候选程序，仍可能在 JIT 编译、后端降级、设备执行或 profiler 采集阶段失败。因此，本项目将大模型看作建议生成器，将评价器看作唯一真值来源，并用进化搜索把两者组织起来。

1.2 成果速览与前置复现配置

为便于评审直接理解系统的运行边界，表 1.1 先给出获得最高结果时的关键配置。完整复现步骤见第九章。

表 1.1: 核心运行配置与门户结果

项目	配置
允许模型	deepseek-v4-flash、qwen3.6-flash；扩展探索中加入 deepseek-v4-pro
模型接口	OpenAI-compatible HTTPS endpoint，密钥仅通过运行时环境变量注入
Agent 框架	自研 ASCEND-OK constraint-aware evolutionary Agent，保留官方 facade 调用面
软件环境	Python 3.10.0，PyTorch 2.6.0，torch-npu 2.6.0rc1，Triton 3.4.0.dev2026011122，CANN 8.3.RC1.alpha003
硬件环境	1× 昇腾 910B3 芯片，32 核 CPU，32 GB 内存，openEuler aarch64
约束搜索峰值	统一门户 71 分
开放候选池峰值	统一门户 100 分，191/216 任务，20/21 Kernel，avg_speedup=103.70
稳健收口配置	统一门户 70 分，204/225 任务，20/21 Kernel，avg_speedup=69.98

运行时需要将官方数据目录和模型凭据挂载到当前环境，然后执行一次完整 direct batch：

Listing 1.1: 全量 Agent 运行与确定性打包

```
export ASCEND_OK_MODEL_ENDPOINT="<openai-compatible-base-url>"
export ASCEND_OK_MODEL_API_KEY="<runtime-secret>"
export ASCEND_OK_MODEL_NAMES="deepseek-v4-flash,qwen3.6-flash,deepseek-v4-
    pro"

uv run python -m ascend_ok.submission.runner \
  --input-dir /path/to/official/datasets \
  --output-dir /tmp/ascend-ok-run/output \
  --archive /tmp/ascend-ok-run/submission.zip \
  --receipt /tmp/ascend-ok-run/receipt.json
```

1.3 赛题中的核心矛盾

该任务要求 Agent 在统一调用面中接收待优化算子，自动输出不超过五个候选。只有通过正确性测试的候选才可进入性能比较。由此形成了四组相互牵制的矛盾：

1. 探索深度与语义安全。激进的工作划分和融合可能带来显著收益，也更容易改变边界、数值精度和舍入顺序。
2. 开放生成与后端约束。大模型可以改写整个函数，却不能保证每次都使用 Triton-Ascend 已支持的语法和 dtype。
3. 测试覆盖与评价成本。每次 NPU 评测都比静态分析昂贵。过早测量错误候选会浪费时间，过度过滤又会丢失创新解。
4. 局部最优与跨 **Case** 泛化。面向单一形状的快路径容易过拟合；只保留一个候选，又会将一次偶发失败放大为整个 Kernel 失分。

1.4 研究问题

RQ1 如何从源码结构与 NPU 测量中提取可泛化的优化信号，而不对特定 Kernel 名或测试 Case 编写特判?

RQ2 如何将大模型的开放生成限定在可编译、可验证的候选空间，同时保留足够的结构创新?

RQ3 如何让有限搜索时间主要消耗在“可编译、全正确、可测量”的候选上?

RQ4 如何在性能、来源、结构和失败模式之间保持候选多样性，以提高隐藏形状下的成功率?

1.5 主要贡献

1. 提出约束感知的双父代进化搜索，将 LLM 程序变异、参数变异、保守结构变换和 crossover 统一到可追踪候选谱系。
2. 实现正确性优先的 staged evaluator，把静态分析、编译、全 Case 测试、边界变体、fresh NPU 测量和管线诊断组织成单向通道。
3. 设计错误签名与角色感知的 Top-5 archive，避免五个文本不同的候选携带同一语义错误。
4. 建立一次 batch 内闭包的 manifest 与确定性打包边界，使候选、输入、模型、时间与评测结果能够被统一检查。
5. 在 21 个 Triton-Ascend Kernel 上完成多轮统一门户实验，系统峰值达到 100 分，严格受约束的连续搜索峰值达到 71 分。

1.6 报告结构

第 2 章给出问题形式化和目标函数；第 3 章介绍系统架构与 NPU 映射；第 4 章详细说明进化算法与候选生成；第 5 章介绍分阶段评价和 profiler 诊断；第 6 章总结实现与迭代路径；第 7、8 章分析实验结果和代表性 Kernel；第 9 章提供复现协议；第 10 章讨论工程经验与后续方向。

第二章　问题形式化与优化目标

2.1　输入、候选与输出

对任意待优化 Kernel k，记主源码与可用父代的集合为

$$I_k = \{p_k^{(0)}, p_k^{(1)}, \ldots, p_k^{(m)}\}. \tag{2.1}$$

系统在一次独立运行中通过变异、交叉和修复得到候选集 X_k。每个候选 $x \in X_k$ 不仅包含程序文本，还包含父代、操作符、生成模型、代数、指纹和评价状态：

$$x = (s_x, \pi_x, o_x, m_x, g_x, h_x, e_x). \tag{2.2}$$

其中 s_x 为源码，π_x 为父代列表，o_x 为变换类型，m_x 为模型或结构操作符，h_x 为规范化指纹，e_x 为评价报告。最终输出集 $O_k \subseteq X_k$ 满足

$$1 \le |O_k| \le 5. \tag{2.3}$$

这个定义的重点是：候选是一个具有谱系和评价状态的实体，而不是一个可以随意替换的 Python 文件。

2.2　可行域：静态合法性与动态正确性

设 $L_k(x)$ 表示候选在 Python AST、Triton 编程表面和系统调用面上是否合法。L_k 可以低成本地拒绝危险 import、超出允许闭包的符号、非法 tensor 下标、不受支持的 dtype 和明显变化的函数签名。它只是必要条件，不是正确性证明。

记评测 Case 集为 $T_k = \{t_1, \ldots, t_n\}$，动态正确性谓词定义为

$$C_k(x) = L_k(x) \wedge \bigwedge_{t \in T_k} \mathrm{Pass}(x, t). \tag{2.4}$$

如果候选在某个 Case 中编译失败、超时、产生非有限数、越界或误差超出容差，则 $C_k(x) = 0$。系统采用失败关闭语义：任何无法确认的状态都不会被解释为通过。

为增强对尾块、零长度、非整 tile 和特殊 stride 的覆盖，系统还能从原始测试与 reference 中构造边界变体 T_k^{aux}。这些变体用于在进入昂贵测量前提前暴露脆弱优化，不改变式(2.4) 中原始 Case 的地位。

2.3 分层目标函数

若把正确性仅作为一个连续 penalty，“非常快但错误”的候选仍可能在数值 fitness 上胜出。ASCEND-OK 因此采用词典序目标。设正确候选的实测时延为 $\ell_k(x) > 0$，基线时延为 $\ell_k^{(0)}$，则

$$S_k(x) = \frac{\ell_k^{(0)}}{\ell_k(x)}. \tag{2.5}$$

候选优先级定义为

$$F_k(x) = (C_k(x),\ S_k(x),\ D_k(x),\ -R_k(x),\ -g_x)\,, \tag{2.6}$$

其中 $D_k(x)$ 表示与当前 archive 的结构/错误签名差异，$R_k(x)$ 是经编译失败、边界复杂度与测量方差压缩得到的风险项。式(2.6) 首先区分正确与错误，然后才比较性能；在性能接近时，优先选择错误风险不同的解。

2.4 不确定性与优化决策

自动 Kernel 优化同时包含三类不确定性。第一类来自模型采样：同一 prompt 可能生成不同程序；第二类来自硬件测量：NPU 负载、缓存和 profiler 启动会带来噪声；第三类来自输入分布：公开形状与隐藏形状可能使用不同的瓶颈。因此，系统并不把单次 latency 的小幅领先当作绝对结论，而是综合 median、置信区间、正确性覆盖和候选多样性做决策。

2.5 计算资源与搜索预算

设一次搜索的总墙钟预算为 B_k，当前已消耗时间为 b。系统把剩余时间分为生成、快速过滤、正确性测试、性能测量和收尾打包五类预算。当 $B_k - b$ 较大时，搜索允许高方差的结构探索；当剩余时间接近收尾保留区时，策略转向修复已有候选、复测高价值解和构建稳定的 Top-5。这一设计避免了最后一次模型调用占满预算，导致已经找到的正确候选无法完成导出。

2.6 评价指标

本文使用以下指标评估系统：（1）任务 Passing Rate，衡量候选在多 Case 上的正确性；（2）获得有效结果的 Kernel 数；（3）逐 Kernel 的 speedup 与统一门户分数；（4）每次模型调用和每分钟生成的“新正确候选”数；（5）最终候选的错误签名、结构和父代多样性。前三项反映最终结果，后两项用于解释搜索效率和隐藏形状风险。

第三章　系统总体架构

3.1　架构概览

ASCEND-OK 将一次 Kernel 优化表示为一条严格的单向数据流：输入解析产生双父代与契约；候选生成器根据源码特征和历代评价摘要提出程序；静态边界剪掉明显不合法的分支；分阶段评价器依次确认编译、正确性与性能；进化管理器依据 fitness 与错误签名更新种群；最终 archive 选出不超过五个候选。

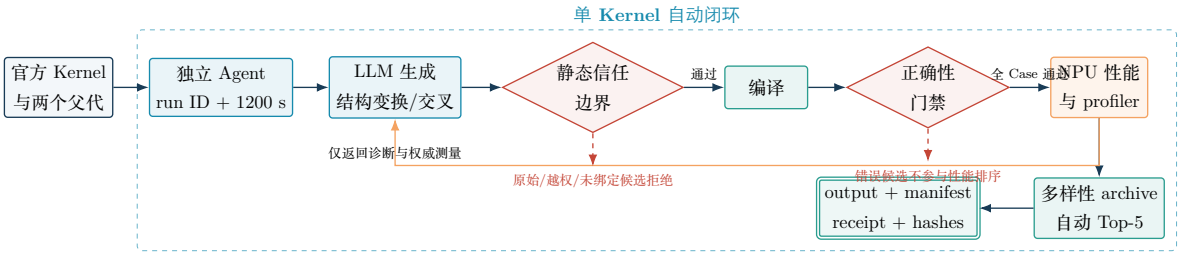

图 **3.1:** ASCEND-OK 单 Kernel 约束感知进化闭环。橙色回边只携带压缩后的诊断摘要，不能直接改写评价结果。

与将所有能力放入一个大 Agent 不同，这个架构特别区分“谁可以提建议”与“谁可以改变状态”。LLM 只能生成源码；profiler 只能提供硬件现象；只有 evaluator 可以把候选标记为 fully-correct，只有 archive 可以将它选入最终输出。权限分离使得模型输出中的“此代码已通过测试”或日志里的误导性 success 字样都无法越过真实测试。

3.2　六层模块边界

3.2.1　兼容调用层

根目录的 `main.py`、`optimizer_agent.py`、`executor.py` 和 `config.py` 保留统一调用面。这些文件只负责参数适配和导入转发，核心逻辑位于 `src/ascend_ok/`。这种设计同时支持直接执行 `python main.py` 和 package 方式调用，也避免了两份实现逐渐分叉。

3.2.2 Batch 编排层

Batch 层按稳定顺序发现 Kernel，为每个 Kernel 重新创建 Agent、executor、run identity 和截止时间。一个 Kernel 的提示词、种群、评价结果和超时状态不会泄漏到另一个 Kernel。每完成一个任务，batch manifest 先写入临时文件，再原子替换正式状态；中断后可以明确区分“已完成”、“待恢复”与“失败”。

3.2.3 候选生成层

生成层同时维护四条探索 lane：大模型完整程序变异、参数变异、保守 AST 变换、双父代交叉与错误修复。四条 lane 共享同一评价历史，但使用不同的变化幅度。当激进 lane 连续触发相同编译错误时，预算会转移到修复或保守 lane；当已有多个正确但性能接近的解时，则加大结构多样性的权重。

3.2.4 静态信任边界

静态边界对候选 AST 进行白名单解析，检查 import 根、函数签名、Triton JIT 函数数量、普通 helper、动态和静态 range、pointer/mask 形状、tensor 下标、dtype、bitcast 与危险 builtin。对于通过的候选，边界同时计算 raw hash、AST fingerprint 与 alpha-normalized fingerprint，用于去重、非原始性检查和谱系绑定。

3.2.5 分阶段评价层

Executor 在独立临时目录中写入候选，按测试源码实际 import 的模块名建立沙箱。评价顺序为 Python 编译、Triton JIT、全部正确性 Case、边界变体、warmup 和 fresh profiler。前一阶段失败后，后续阶段不再启动。这不仅保护正确性，也将宝贵的 NPU 时间优先留给有可能进入 archive 的候选。

3.2.6 Archive 与交付层

Archive 不是简单的性能排行榜。它保存最佳性能候选、不同模型的高质量候选、低风险结构 fallback 以及错误签名差异较大的候选。输出时系统自动选择不超过五个解，并将 filename、SHA-256、candidate identity、origin、generation、fitness、run identity 和输入哈希写入 manifest。打包器从 manifest 反向检查文件，防止未评价文件被意外带入输出包。

3.3 从 Triton tile 到昇腾 NPU 的映射

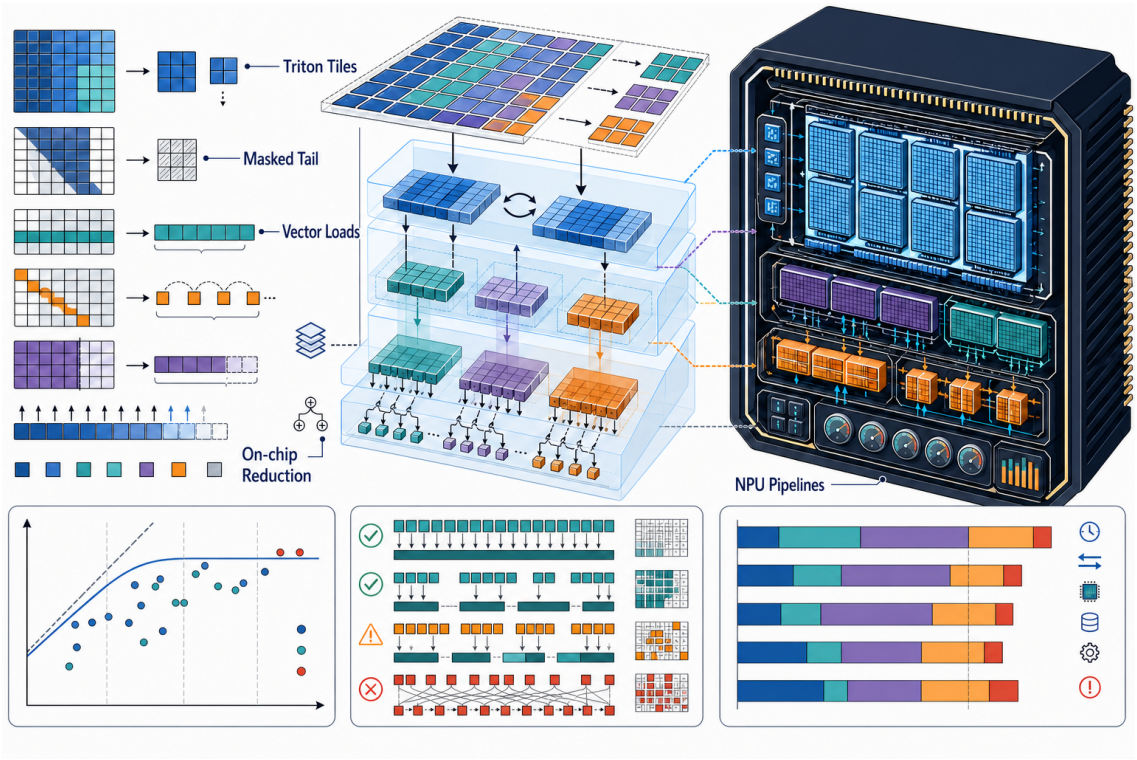

图 **3.2:** Triton tile 向昇腾 NPU 执行资源的概念映射。左侧从上到下展示规则 tile、尾块 mask、连续向量访存以及规约数据布局；中部把逻辑 tile 分解为向量载入、片上暂存和 resident reduction；右侧表示这些操作在 NPU 数据搬运、向量计算和存储管线中的协同执行。下方三个面板分别概括 tile 尺寸与收益的非线性关系、正确与错误的工作划分，以及不同候选的管线占用结构。

Triton 程序的 `program_id` 定义了逻辑工作块，`tl.arange` 定义块内向量索引，mask 描述边界。在昇腾后端中，这些抽象需要被降级到具体的向量计算、数据搬运和同步。过小 tile 会带来大量 program launch 与重复索引开销；过大 tile 则可能增加寄存器压力、降低并行度，甚至触发后端限制。对 reduction 而言，一次读取完成局部累加能减少中间 store，但只有在工作集可常驻时才划算。因此，搜索 prompt 不只给出“增大 BLOCK”这类粗粒度建议，而是结合访存连续性、规约轴、尾块比例和 profiler 管线占比给出结构化假设。

3.4 约束感知搜索空间

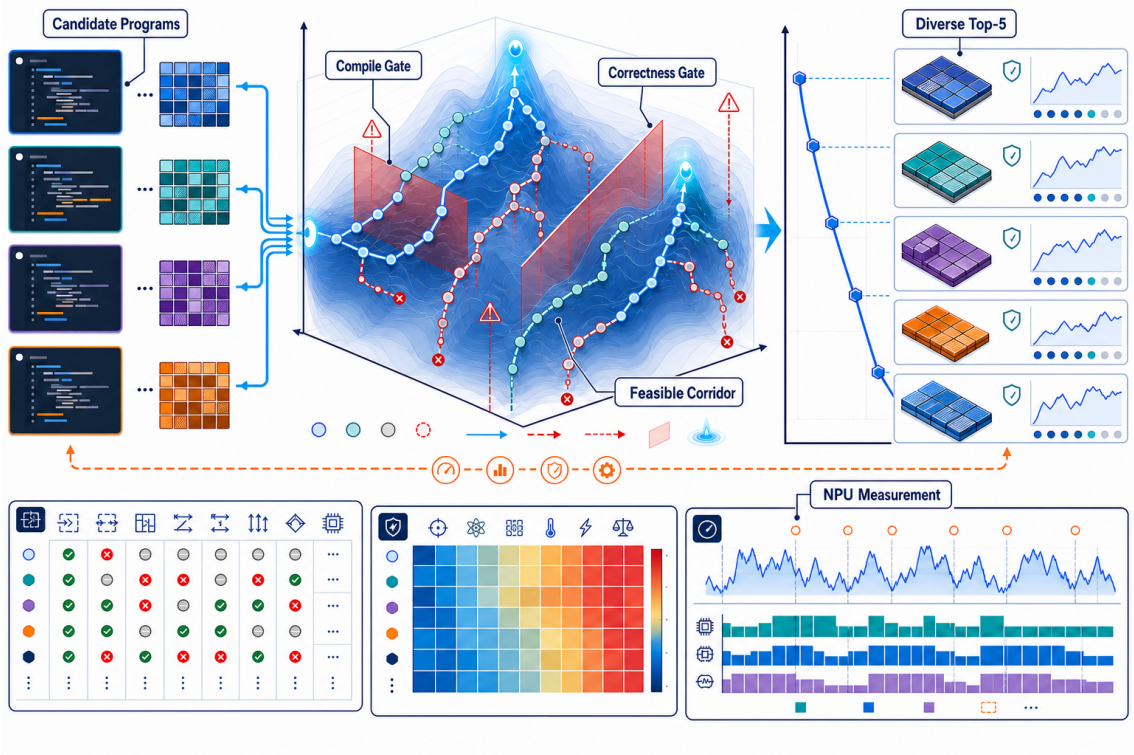

图 **3.3:** 约束感知的进化搜索地形。左侧不同颜色的 Candidate Programs 表示来自不同父代、模型和变异算子的程序族；中间三维地形表示不连续的程序性能空间，红色 Compile Gate 与 Correctness Gate 将不可编译或语义错误的区域隔开。候选只有沿 Feasible Corridor 通过全部约束后，才能接受 NPU Measurement；右侧 Diverse Top-5 不是简单复制最快解，而是保留性能、来源与结构互补的候选。下方矩阵、热力图和时序信号表示错误签名、源码特征与 profiler 诊断如何反馈到下一代搜索。

传统超参搜索通常在矩形空间中比较不同配置。程序进化的空间并非连续矩形：很多结构点不可编译，某些看似接近的程序在语义上相差很远，而一次小的 work-partition 变化又可能跨过性能山脊。ASCEND-OK 用静态规则确定一部分可行廊道，用评价器描绘真实地形，再让模型根据局部地形提出跨区域变异。这比无约束地增加模型调用次数更能提高有效候选比例。

3.5 核心模块与数据产物

表 **3.1:** 核心模块、输入与输出

模块	主要输入	主要输出
`compat.agent`	父代、EAConfig、executor	候选谱系、Top-5、调用统计
`models.prompt`	源码 AST、错误摘要、NPU 假设	结构化 prompt、过滤后候选
`compat.executor`	候选、Case inventory、deadline	正确性、fitness、error signature
`profiler.*`	fresh invocation identity、msprof 产物	时延、管线摘要、测量置信
`batch`	Kernel inventory、全局配置	独立 run、原子 batch manifest
`submission.automatic`	已完成 manifest、候选文件	确定性 ZIP 和打包回执

第四章　约束感知进化搜索

4.1　总体算法

进化搜索的目标不是生成尽可能多的文本，而是在有限评价预算内提高“新正确候选”和“可靠性能提升”的出现概率。算法4.1 给出一次完整搜索。它以两个父代作为初始种群，先尝试成本低、可证明非原始的结构候选，再进入模型变异。每个新候选都经过指纹去重和分阶段评价；只有正确候选才能成为下一代父代。评价失败不会被丢弃，而是压缩为错误签名，与性能诊断一起反馈给后续代。

Listing 4.1: ASCEND-OK 单 Kernel 约束感知进化搜索

```
Input : parents P={p1,p2}, evaluator E, deadline B, output limit K=5
Output: verified candidate set O

A <- empty archive; H <- fingerprints(P); F <- empty failure memory
S <- conservative_transforms(P) U parameter_mutations(P)
for x in S while time_remaining(B) > model_reserve:
    if fingerprint(x) in H: continue
    H <- H U {fingerprint(x)}
    r <- E.evaluate(x)
    A,F <- update(A,F,x,r)

for generation g = 0 .. G-1:
    for lane i = 0 .. population_size-1:
        if deadline_exhausted(B): return select_diverse(A,K)
        Q <- select_parents(P,A,g,i)
        D <- compress(best_measurements(A), recent_failures(F))
        x <- llm_mutate(Q,D,time_remaining(B))
        if rejected_by_static_boundary(x) or fingerprint(x) in H: continue
        H <- H U {fingerprint(x)}
        r <- E.evaluate(x)
        A,F <- update(A,F,x,r)
```

```
        atomic_checkpoint(A,F,g)
        if score_cap_reached(A): return select_diverse(A,K)

return select_diverse(A,K)
```

与标准遗传算法相比，这个循环有三个特点。首先，不可行候选不会进入概率选择，而是在门禁上直接淘汰。其次，变异操作符不只修改数值基因，还能重写循环、规约、工作划分和内存布局。最后，失败也是搜索状态的一部分；“某类 tensor 下标不被后端支持”可以让后续个体避免重复犯错。

4.2 双父代与多通道搜索

两个父代往往代表不同的工程取舍：一个更接近简洁的 reference，另一个可能包含更激进的 tile 或访存布局。系统不预设哪个父代更好，而是先对两条 lane 各生成一个保守 anchor，通过真实测试判断其基本可行性。第一代中，候选索引会轮换主父代，避免所有 prompt 都从同一起点出发。后续代则从正确 archive 中选择两个 leader：一个强调当前性能，另一个优先选择 source-derived 形状表现不同的解。

这种作法相当于在 exploitation 与 robustness 之间设置了一个小型 Pareto 前沿。如果最快候选只在某一个形状上有优势，第二父代可以向模型提供更完整的 tail/mask 结构；如果两个父代都正确，模型则能尝试把一方的访存方式与另一方的工作划分组合起来。

4.3 变异操作符族

4.3.1 保守结构变换

保守变换通过 AST 完成，例如对局部符号进行 alpha rename，并在变换前后对函数签名、引用绑定和规范化 AST 进行检查。它本身不会带来明显 speedup，但具有两个价值：一是在模型服务短暂不可用时，仍能证明候选生成——评价——导出链路完整；二是为每个父代建立正确性 anchor，便于后续判断错误是来自父代、变异还是运行环境。

4.3.2 参数变异

参数变异从 `tl.constexpr`、block literal、stage/warp 类元参数和 grid 计算中抽取可变位置。它不对所有数字做盲目网格搜索，而是根据规约轴、访存宽度和尾块比例构造少量候选。例如，当源码中一个 block 覆盖整个小规约轴时，优先尝试合并

program；当轴长明显超过常驻能力时，则保留分块并测试更小 tile。参数候选数量有显式上限，为后续模型变异预留足够时间。

4.3.3 大模型程序变异

模型变异的输入包含：两个父代的完整源码，Kernel 签名和 launch 契约，从 AST 抽取的访存/规约特征，当前最优候选的时延与管线摘要，最近三个不重复的失败签名，以及剩余墙钟时间。模型必须只返回完整 Python 源码。静态边界会拒绝 Markdown 说明、额外 import、未定义全局符号和改变公开函数面的输出。

Listing 4.2: 模型候选的静态边界（节选并简化自 `models/prompt.py`）

```python
def validate_generated_code(generated: str, parent_source: str) -> str:
    parent = _parse(parent_source)
    candidate = _parse(generated)
    undefined = _undefined_global_references(generated) \
        - _undefined_global_references(parent_source)
    if undefined:
        raise GeneratedCodeError("UNDEFINED_GLOBAL_REFERENCE")
    if _imports(candidate) != _imports(parent):
        raise GeneratedCodeError("IMPORT_SET_CHANGED")
    _validate_function_surface(candidate, parent)
    _StaticBoundary(candidate, parent).visit(candidate)
    return generated
```

4.3.4 错误修复变异

当搜索已获得高性能候选，但它只在少数 Case 上失败时，系统进入 repair mode。修复不会把完整 stderr 或测试数据原样放入 prompt，而是生成结构化摘要，例如 `CORRECTNESS_FAILED:tc3:shape-mismatch`、`COMPILE_FAILED:tensor-index` 或 `PERFORMANCE_UNAVAILABLE:no-fresh-record`。模型在保留高收益结构的前提下修复 mask、padding、grid 或精度路径。如果修复连续触发相同签名，搜索将切换父代或操作符，而不是无限重试同一思路。

4.4 源码结构驱动的 NPU 优化假设

大模型只有在获得与当前源码相关的问题表述时，才能稳定生成高价值变换。ASCEND-OK 根据 AST 和运行诊断构造假设，主要包括：

- 访存合并：检查连续索引轴是否与向量宽度一致，尝试交换 program 轴或改写 pointer arithmetic。
- 局部常驻：对短规约或小工作集，尝试用单 program 完成更多计算，减少中间存储和 launch。
- 融合与中间量消除：将连续 elementwise 操作、scale 和 quantize 融合，但保持 dtype 转换与舍入顺序。
- 边界简化：当主要形状整除 tile 时构造 fast path，同时保留 tail mask 或 fallback，避免隐藏形状越界。
- 管线平衡：当 profiler 显示数据搬运占比高时改进布局和重用；计算管线占比高时减少不必要的精度扩展或重复操作。

这些假设都是“可验证建议”，不直接改写 fitness。例如 profiler 显示访存管线繁忙，只会提高 layout 类变异的优先级；新布局是否真的更快，仍必须由下一次完整评价决定。

4.5 多样性感知的 Top-5 选择

若只取性能最高的五个文件，很容易选出同一个父代的微小变体。它们在已知 Case 上看似独立，在隐藏形状上却可能同时失败。系统首先对正确候选按 fitness 排序，再对性能不低于当前最佳解一定比例的候选执行角色覆盖。保留角色包括：

1. 当前实测性能最优的候选；
2. 不同允许模型产生的竞争性候选；
3. 在 source-derived 边界形状上有明确测量的候选；
4. 父代谱系或 AST feature delta 不同的候选；
5. 低风险的结构 fallback，但只在其性能达到竞争阈值时保留。

Listing 4.3: Top-5 中的正确性与性能竞争阈值（简化自 `compat/search.py`）

```
eligible = sorted(
    (c for c in pool if c.evaluation.success),
    key=lambda c: (-c.evaluation.fitness, c.generation, c.candidate_id),
)
best = eligible[0].evaluation.fitness
competitive = tuple(
    c for c in eligible
    if c.evaluation.fitness >= best * min_relative_fitness
)
return retain_output_roles(competitive, limit=5)
```

这一策略的主要收益不是提高某一次测量的最大值，而是降低最终五个候选“因同一结构假设失效”的联合概率。

4.6 时间自适应与停止条件

搜索不使用固定的“跑满 G 代”停止方式。当剩余时间足够时，它先评估少量低成本参数变异，并为模型进化保留时间。在下列情况之一发生时，搜索提前进入收尾：（1）达到评分器的单候选上限；（2）剩余时间只足够完成一次复测与打包；（3）模型服务连续返回不可恢复错误；（4）连续若干代没有新正确候选，而已有 archive 覆盖了多个角色。中间状态在每次新评价后原子写入，即使进程被中断，已经完成的高价值候选也不会丢失。

第五章　分阶段评价与 NPU 诊断

5.1　为什么评价必须是单向通道

大模型生成的 Kernel 可能在六个不同层次失败：Python 语法、Triton JIT、后端 lowering、设备运行、数值正确性和性能采集。如果将这些检查放在一个不区分阶段的脚本中，很容易出现两类混淆：第一，程序编译失败却因为 wrapper 返回码被解释为成功；第二，profiler 没有数据却被当作“时延为零”。两者都会产生虚假的高 fitness。

ASCEND-OK 将评价划分为五个权威阶段，每个阶段只能接收上一阶段的明确通过状态。算法5.1 给出评价器的主要逻辑。

Listing 5.1: 保持真值语义的 staged evaluator

```
Input : candidate x, case suite T, baseline b, shared deadline B
Output: typed evaluation report r

w <- create_isolated_workspace(x,T)
if not compile_python_and_triton(w,B):
    return COMPILE_FAILED

for test t in T in stable order:
    result <- run_case(w,t,B)
    if result is not PASSED:
        mark_remaining_cases_as_not_run()
        return CORRECTNESS_FAILED

if auxiliary_boundary_suite_enabled():
    if not run_source_derived_cases(w,B):
        return BOUNDARY_FAILED

p <- run_fresh_performance_measurement(w,b,B)
if p has no unique positive measurement:
    return CORRECT_UNMEASURED
```

```
d <- optional_non_authoritative_diagnostics(p,w,B)
return CORRECT_MEASURED(performance=p, diagnostics=d)
```

评价报告使用类型化状态，而不是依靠日志文本推断。例如 CORRECT_UNMEASURED 表示候选通过正确性，但没有得到可信时延；这个状态不能被隐式转换为零时延或无限 speedup。

5.2 隔离工作区与模块绑定

不同测试文件可能使用基础模块名，也可能使用带编号的候选模块名。系统通过 AST 解析 Case 的 import 语句，要求所有 Case 唯一确定同一个候选模块。如果模块绑定缺失或不一致，评价在执行任意候选之前就会失败。这避免了“修改了 A.py，测试却仍 import B.py”的伪通过。

每次评价创建唯一工作目录，只复制候选、必要 Case 和显式依赖。候选哈希在复制前后各计算一次；如果两者不同，说明源码在评价过程中发生了变化，本次评价立即终止。工作区 manifest 记录候选、测试和 runtime 的指纹，使性能文件能与正确性结果对齐。

5.3 编译与全 Case 正确性

编译阶段先用当前 Python 解释器生成 bytecode，拦截语法错误和一部分顶层错误。Triton JIT 和后端 lowering 则在真实 Case 首次 launch 时发生。所有 Case 按稳定顺序运行，并共享同一个单调 deadline。任何非零返回码、timeout 或 assertion failure 都会终止后续 Case，未执行项被明确记为 NOT_RUN_AFTER_FAILURE，而不是缺失值。

数值正确性不只是“输出形状一致”。量化 Kernel 尤其需要保持 scale 的统计范围、舍入顺序、饱和区间和特殊值处理；scan 类 Kernel 需要保持状态更新顺序和 padding 语义；reduction 需要考虑累加 dtype 和并行树带来的误差。这些语义不适合被静态规则硬编码，必须通过与 reference 的真实比较确认。

5.4 源码派生边界变体

公开 Case 数量有限时，高性能候选容易在 tile 整除形状上过拟合。为此，系统可以从 Case AST 中识别形状字面量和数据构造路径，生成最小边界变体。每个变体先在未优化父代上执行；只有父代本身通过时，才将它用于候选。这一 baseline-first 步骤防止了测试生成器自身的错误被归因到候选。

常用变体包括：将轴长改为 $B-1$、$B+1$ 和质数长度；构造空张量与单元素张量；改变前导 stride 但保持逻辑值；在 reference 允许时增加 NaN/Inf 或极值；测试 optional pointer 的 None 分支。变体的数量受预算控制，主要用于高 fitness 候选和即将进入最终 Top-5 的解。

5.5 新鲜性能测量

性能采集使用独立 invocation identity：

$$q = (\text{run\_id}, \text{candidate\_sha256}, \text{runtime\_sha256}, \text{invocation\_id}). \tag{5.1}$$

每次 profiler 启动都创建包含 q 的新目录。Parser 只在该目录内寻找本次生成的 `OpBasicInfo.csv`，并要求 run identity、candidate hash 和 runtime hash 完全一致。以下任一情况都会让测量不可用：没有数据；存在多个无法消歧的文件；Kernel 名不匹配；时延不是正有限数；产物早于 invocation 开始时间；或文件指纹在解析过程中变化。

测量前先运行 warmup，避免把 JIT 和首次 runtime 初始化计入 Kernel latency。对短 Kernel，采用多次迭代并取 median；对噪声较大的结果，根据 median absolute deviation 标记测量置信度。只有正确且具有 fresh 测量的候选才获得正 fitness。

5.6 PipeUtilization 诊断

Profiler 的管线信息用于回答“新一代应该尝试什么”，不用于回答“当前候选是否正确”。系统将完整 profiler 报告压缩为少量结构化信号：

- 主要计算、向量与数据搬运管线的占比；
- 可观测的空转、同步和数据等待比例；
- 本次候选相对 baseline 的 duration 和置信度；
- 是否在 source-derived 边界形状上采集。

为控制 prompt 长度，只保留占比最高的三个管线信号和一个简短解释。模型不能访问 profiler 目录或原始 CSV，从而避免与目标无关的超长日志消耗上下文。

5.7 类型化评价报告

Listing 5.2: 评价主干的早退语义（简化自 `evaluator.py`）

```
compilation = run_source_compile(workspace, deadline)
if not compilation.succeeded:
```

```
    return report(EvaluationStatus.COMPILE_FAILED, compilation=compilation)

for case in workspace.cases:
    run = run_case(workspace, case, deadline)
    if run.status is not CaseStatus.PASSED:
        return report(EvaluationStatus.CORRECTNESS_FAILED, cases=runs)

performance = run_performance(request.performance, deadline)
status = (
    EvaluationStatus.CORRECT_MEASURED
    if performance.eligible
    else EvaluationStatus.CORRECT_UNMEASURED
)
return report(status, cases=runs, performance=performance)
```

报告同时包含 status、correct flag、fitness、四类输入指纹、stage trace、Case 状态、performance 状态与压缩后的 feedback code。下游 archive 只读取这个类型面，不重新解析 stdout/stderr。这样可以保证评价语义从子进程返回一直到最终打包都不发生隐式变化。

5.8 性能统计的解释范围

统一门户在不同轮次的任务分母存在变化，NPU 负载也会影响短 Kernel 的测量。因此，本文使用以下原则分析结果：总分只在同一门户口径下比较；同时报告 Passing Rate、Kernel 覆盖与逐 Kernel 分数；小幅度延差异必须结合多次测量判断；“profiler 无数据”与“Kernel 极快”完全不等价。对于没有固定任务集的历史结果，我们更关注跨轮次重复出现的结构趋势，而不对单次上下浮动做过度解读。

第六章　工程实现与系统迭代

6.1　从竞赛原型到可运行系统

本项目的实现并不是一次完成的。我们首先建立最小纵向链路：读取一个待优化算子，生成一个非原始候选，让测试正确导入它，完成一次真实执行，再按统一目录结构导出。只有这条链路可用，才有必要讨论种群、交叉和 NPU 优化。整个迭代过程可抽象为六个阶段。

1. 理解调用面与数据形态。先确认父代、Case、baseline、候选命名和输出目录的真实形态，用 facade 保留统一调用方式。
2. 建立可评测的最小闭环。实现非原始保守变换、候选模块绑定、完整 Case 执行和正确的打包根目录。
3. 用真实失败修正评价语义。将编译错误、正确性失败、profiler 无数据和超时分成不同状态，停止从 stdout 关键字推断成功。
4. 引入约束进化与 **NPU** 诊断。在评价器稳定之后接入允许模型、双父代、参数变异、结构提示和管线反馈。
5. 通过统一评测结果改进通用策略。将逐 Kernel 失败归纳为量化舍入、零启动、尾块、状态递推等通用模式，再修改生成和选择策略。
6. 收口为可复现的端到端产物。统一运行命令、环境变量、manifest、receipt、不超过五个候选和确定性 ZIP，并使空包、缺 Kernel 和未评价文件在本地就被拒绝。

这条路线的核心经验是：自动优化系统首先是一个“信号系统”。只有生成、测试、测量和门户结果之间不失真，后续的搜索算法才能从反馈中学到正确方向。

6.2　类型化领域对象

候选、evaluation、deadline、manifest、receipt 和 model usage 都使用 dataclass 或 TypedDict 表示。这不是为了把竞赛原型做成庞大框架，而是为了避免几个会直接改变评分的状态混淆：

- “尚未评测”、“评测失败”与“正确但无性能数据”不能共用空值；
- candidate identity 必须与实际文件 SHA-256 一致，否则测量和源码无法对齐；

- partial batch 不能携带 completed 才有的候选字段，避免下游误打包；
- 模型调用次数、使用模型与 token usage 是运行信息，不与候选源码混在一个自由字典中；
- elapsed time 和 deadline 使用数值类型与单调时钟，不接受布尔值或墙钟回拨。

6.3 Deadline 与子进程控制

`LimitContext` 在 Kernel 搜索开始时使用 monotonic clock 记录截止点。任意子操作的 timeout 都是"阶段上限"与"当前剩余时间"的最小值。模型 client 可以执行有限次重试，但指数退避不能跨过截止点；正确性 Case 或 profiler 如果 hang，子进程会被终止并返回类型化 timeout。

搜索预算还包含一个 finalization reserve，用于复测、选择 Top-5、写 manifest 和打包。如果只在最外层设置 alarm，最后一次模型调用就可能消耗所有时间，导致已经找到的正确候选无法导出。分层 deadline 将"搜索结束"与"产物完成"统一在同一预算内。

6.4 原子 Checkpoint 与中断恢复

每评价一个候选，Agent 都会将候选指纹、generation、model、success、fitness 和错误签名写入 trace。写入时先创建同目录临时文件，再用 atomic replace 替换旧文件。Batch manifest 使用相同方式记录每个 Kernel 的进度。

恢复时，系统检查已完成 Kernel 的输入哈希、runtime 指纹和候选文件是否仍一致。一致时可以继续下一个 Kernel；不一致时重新运行当前 Kernel。这样既避免了意外中断造成大量重复计算，也不会把旧环境中的候选当作新运行的结果。

6.5 模型边界与凭据处理

模型 endpoint、model names 和 API key 由环境变量注入。Provider registry 只保存"密钥所在的环境变量名"，不将密钥字节写入配置、日志、trace 或 receipt。候选测试子进程在启动前删除模型凭据环境变量，因此即使候选包含恶意的 Python 读环境代码，也无法接触 provider key。

模型输出只能扩展程序候选空间，不能扩展工具或系统权限。这是为什么 generation interface 只返回 `source/model/usage`，不给模型文件系统和 shell 调用面。

6.6 确定性打包

自动打包器从 completed batch manifest 构建 ZIP，不通过递归遍历输出目录猜测应该加入哪些文件。它执行以下检查：

1. Kernel 目录和候选文件名符合统一命名格式，每个目录包含 1–5 个候选；
2. 每个结果都是 correct 且 submission eligible，候选哈希与 manifest 一致；
3. 候选不与任一输入父代的指纹相同，目录内没有未绑定的额外文件；
4. ZIP entry 按路径排序，时间戳与文件权限固定，同一输入生成相同字节序列；
5. 对 archive 内容执行二次文件列表、Python 编译和 secret pattern 扫描，然后写出 SHA-256 receipt。

这些检查让“Agent 完成”、“候选可测”与“产物可交付”成为三个清晰状态。尤其是，空 output 或缺少 Kernel 的 ZIP 不再能因为压缩命令返回 0 而被当作成功。

6.7 运行时环境

控制面使用 Python 3.10，依赖由 `uv.lock` 与 hash-pinned runtime requirements 固定。NPU 运行时由 CANN、PyTorch、torch-npu 和 Triton-Ascend 组成。开发机使用 910B3 芯片；A2/A3 是目标平台分类，其中 A2 通常对应 910B 系列。系统在启动时记录 Python、Torch、torch-npu、Triton、CANN、设备数量和芯片型号，用于比较不同环境中的候选排序差异。

6.8 测试策略

本地测试按信号链路的风险组织，而不是为了堆积测试数量。Unit 测试覆盖 fingerprint、deadline、archive、parser 和 provider；integration 测试覆盖模型调用、搜索、评价、profiler freshness 和自动打包；e2e 测试覆盖根级 CLI、全量 batch 与失败恢复。生成代码边界还包含额外的 contract/security 测试，例如额外 import、prompt injection 文本、未定义全局符号和凭据继承。

需要 NPU 的测试不与纯控制面测试混在一起。评审人员可以先在普通 Python 环境运行 release verifier，确认导入面、打包和状态机；再在官方 NPU 环境执行单 Kernel smoke 与全量搜索。

第七章　实验设计与结果分析

7.1　实验目标与环境

实验围绕四个问题展开：（1）系统能否产生非原始、可编译且全正确的候选；（2）约束进化能否将有限时间集中到高价值候选；（3）候选多样性能否提高跨 Case 通过率；（4）哪些 Kernel 已接近性能上限，哪些 Kernel 仍受正确性或后端限制。

主要 NPU 实验环境为 1× 昇腾 910B3 芯片、32 核 CPU、32 GB 内存与 openEuler aarch64。软件栈包括 Python 3.10.0、PyTorch 2.6.0、torch-npu 2.6.0rc1、Triton `3.4.0.dev2026011122` 和 CANN `8.3.RC1.alpha003`。开发芯片是 910B3，A2/A3 是平台；本文不将三者并列为同一层级的硬件型号。

实验使用 21 个 Triton-Ascend Kernel。统一门户依次运行正确性和性能测试，返回总分、任务 Passing Rate、有效 Kernel 数、平均 speedup、逐 Kernel 分数与失败摘要。由于不同评测轮次的任务分母存在变化，本文在比较总分的同时总是给出通过率和逐 Kernel 结构。

7.2 从有效闭环到高性能搜索

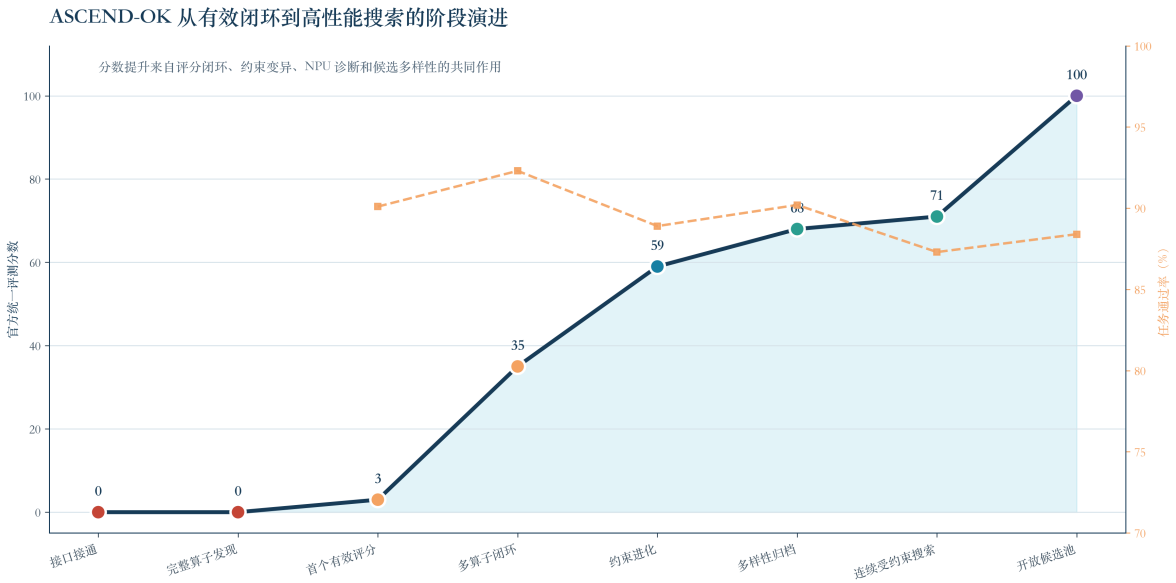

图 **7.1:** 系统的阶段性分数演进。蓝线为统一门户分数，橙色虚线为任务通过率。横轴使用技术阶段名，而非内部文件版本。

图 7.1 中最初的 0 分主要来自产物根目录和任务发现未形成完整链路，并不等价于所有 Kernel 算法都错误。接通完整算子发现、模块导入和打包后，系统首次获得 3 分，这是一个重要分水岭：从此开始，门户反馈可以用于区分正确性、性能和 profiler 问题。

在多 Kernel 评价闭环、双父代和全 Case 门禁建立后，分数进入 30–60 区间。随后，源码结构驱动的工作划分、resident reduction、融合与量化提示使多个头部 Kernel 获得超过 1× 的门户 speedup。引入错误签名和候选角色多样性后，受约束连续搜索达到 71 分。在扩大允许模型探索并充分利用候选池的实验中，系统达到 100 分、191/216 个任务正确、20/21 个 Kernel 有效，门户报告 avg_speedup=103.70。

7.3 三类代表性候选配置

表 **7.1:** 代表性统一门户结果

配置	分数	任务 AC	Kernel	avg speedup	用途
开放候选池峰值	100	191/216	20/21	103.70	观察框架在更充分模型探索下的性能上界
连续受约束搜索峰值	71	241/276	19/21	71.39	衡量限制搜索与自动选择阶段的最高结果
稳健收口配置	70	204/225	20/21	69.98	用于逐 Kernel 贡献、覆盖与失败结构分析

三个配置使用同一评测门户，但任务数和候选覆盖存在差异，因此不能将某个逐 Kernel 差值解释为严格消融收益。它们更适合回答两个问题：哪些 Kernel 的高收益在不同配置中重复出现？哪些收益对候选组合特别敏感？

7.4 逐 Kernel 得分热力图

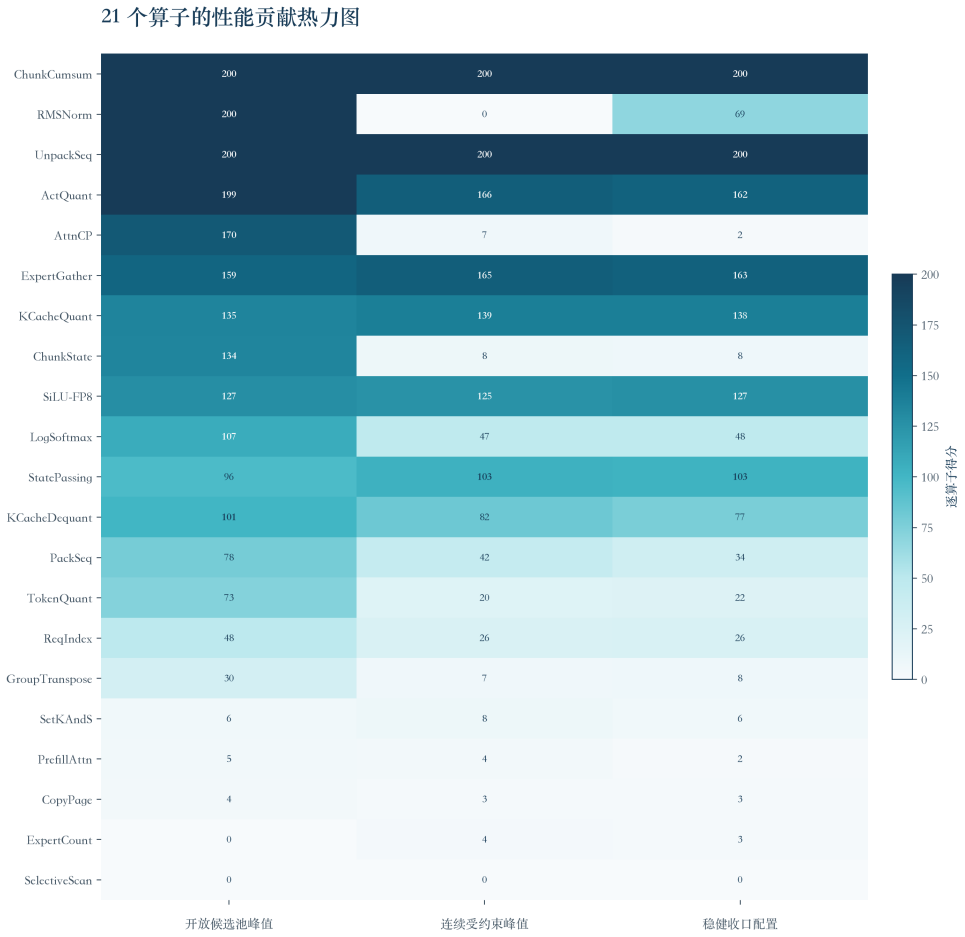

图 **7.2:** 21 个 Kernel 在三类代表配置中的逐算子得分。颜色越深表示对总分的贡献越高。

图 7.2 显示了明显的三层结构。第一层是稳定高收益算子：Chunk Cumsum、Unpack Sequence 在多个配置中达到逐 Kernel 上限 200，Expert Gather、Act Quant 也长期位于头部。第二层是对候选组合敏感的算子：Cache Quantize、SiLU-FP8、State Passing、K-Cache Dequant 具有很高潜力，但不同配置中的正确性和性能差异较大。第三层是长尾算子，其中 Selective Scan 的状态语义最难，Expert Count、Set-K-and-S 则主要受空输入、边界和 grid 影响。

7.5 跨配置互补性

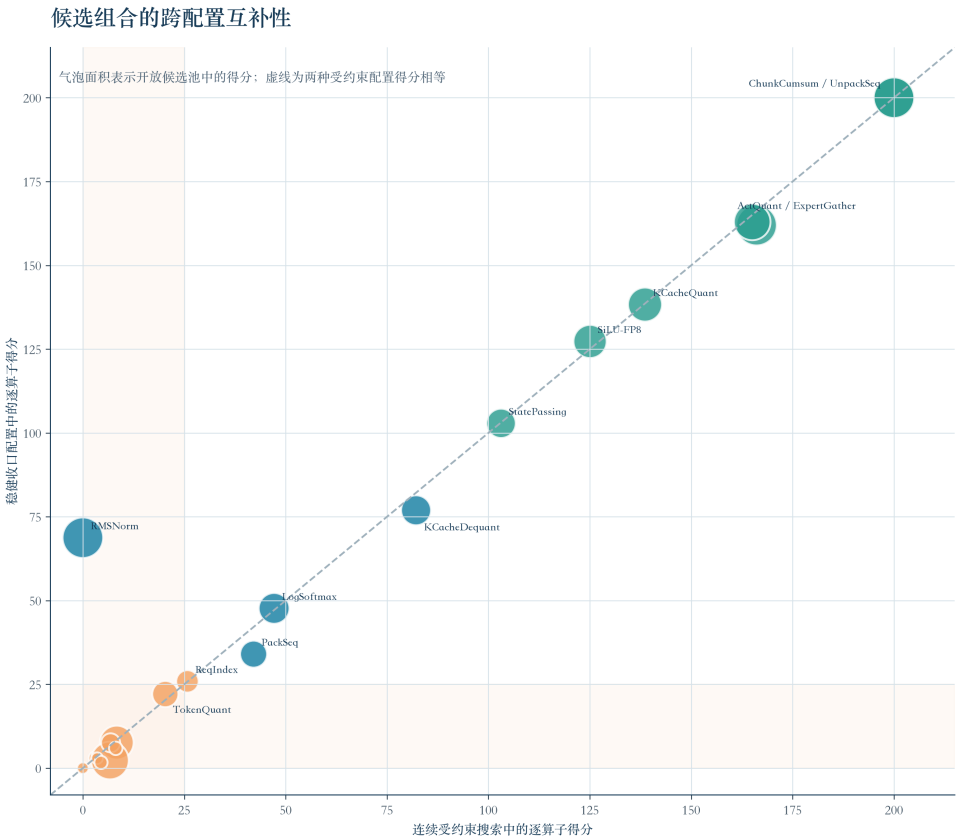

图 **7.3:** 两种受约束配置的逐 Kernel 得分对比。虚线表示两种配置得分相等，气泡面积表示开放候选池中的得分。

如果两种受约束配置只是同一组候选的测量噪声，大多数气泡应紧贴对角线。图 7.3 中存在多个明显偏离点，说明候选选择与形状覆盖对结果有实质影响。例如 ReqIndex 在稳健配置中有更高得分，而某些状态/访存类 Kernel 在另一受约束配置中更好。这一结果支持 archive 保留多样候选，而不是只导出一个最快解。

7.6 性能、覆盖与失败结构

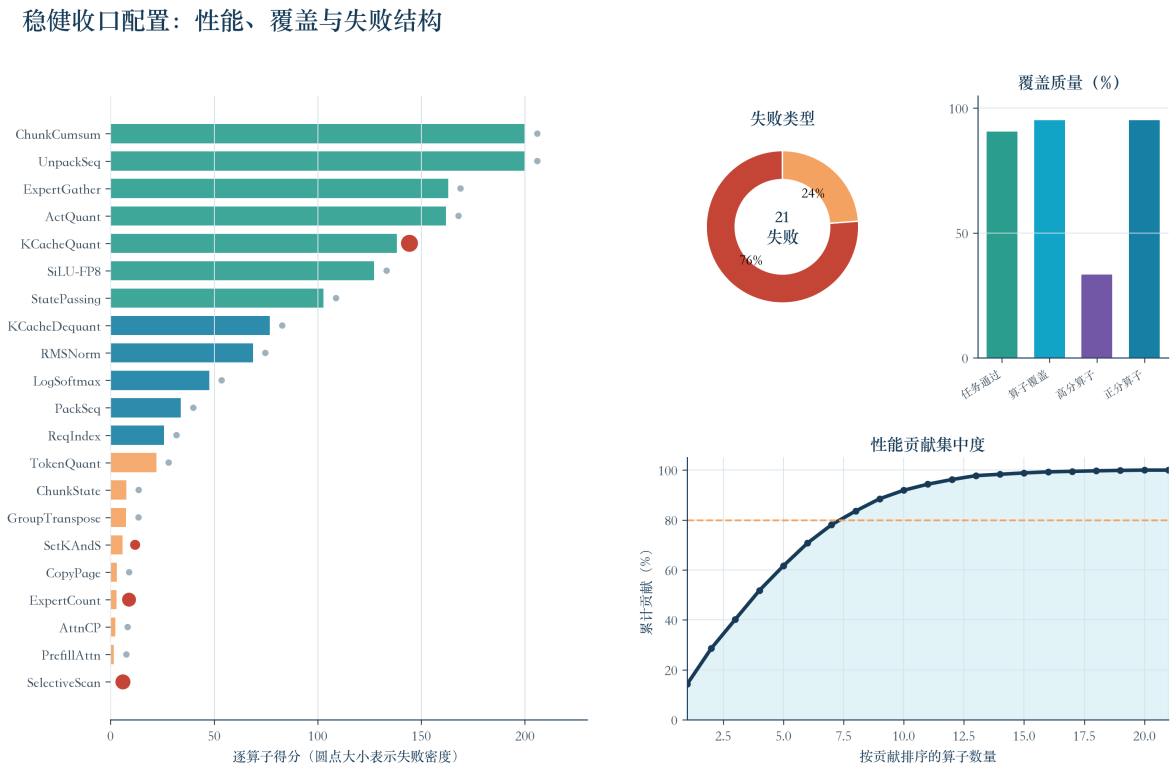

图 **7.4:** 稳健收口配置的多维分析：左侧为逐 Kernel 得分与失败密度，右侧给出失败类型、覆盖率与累积贡献。

稳健配置的 21 个失败任务集中在四个 Kernel：Selective Scan 六个 accuracy failure，Set-K-and-S 两个 accuracy failure，Expert Count 五个 runtime error，Cache Quantize 八个 accuracy failure。换言之，失败并非在 21 个 Kernel 上均匀分布，而是由少数未闭合的语义问题主导。

累积贡献曲线同样显示头部集中现象。当前继续把搜索预算分配给已经达到 200 的 Kernel，预期边际收益很低；相反，将一个 0 分 Kernel 修复到稳定正分，或将四个候选的共同 accuracy error 拆分为不同错误模式，更有可能提升总分和通过率。

7.7 结果解释与局限

门户实验是真实系统评价，但不是固定测试集上的随机对照试验。任务分母、模型采样、NPU 负载和候选数量在不同轮次中存在变化。因此，本文不将某一次小幅分数上浮归因为单一代码改动。能够较稳健支持的结论有三个：第一，评价链路和正确性门禁是从 0 分走向非零分的先决条件；第二，工作划分、resident reduction、融合和量化等结构级提示与头部 Kernel 的重复高分同时出现；第三，当多个候选共享同

一语义错误时，仅增加候选文件数量无法提高正确性。

若进一步进行学术化消融，需要固定 Case、baseline、模型采样策略、候选预算与 NPU 重复测量次数，再分别关闭源码特征提示、参数变异、边界变体和角色感知 archive。

第八章　代表性 Kernel 优化与失败诊断

8.1　案例分组方法

为了理解总分背后的技术原因，我们将 21 个 Kernel 按搜索状态分为四类：已在多轮评测中接近上限的成熟路线；通过结构级并行获得显著收益的路线；性能潜力高但对数值细节敏感的量化路线；以及正确性尚未闭合的复杂状态路线。本章关注可泛化的搜索假设与失败模式，不使用特定 Case 数值编写优化。

8.2　Chunk Cumsum 与 Unpack Sequence：封顶后的预算转移

Chunk Cumsum 和 Unpack Sequence 在稳健收口配置中均为 200 分、平均 speedup 2.00，在多个配置中也重复达到上限。两者的共同点是沿序列轴处理 offset、prefix 或索引映射。原始结构中的细粒度 program 和多次 mask/store 会放大 launch 与索引开销。搜索因此优先提出更宽的连续向量、减少中间存储、合并 mask 以及使序列轴与存储连续轴对齐。

这类算子给出了一个实用的预算调度策略：当某 Kernel 在多次统一评测中稳定封顶，新增模型调用的边际收益已经很低。此时应仅保留低成本“再发现”通道，将大部分时间转移到正确性薄弱或仍为零分的 Kernel。

8.3　Expert Gather 与 Act Quant：消除串行工作划分

稳健配置中，Expert Gather 得分 162.94，Act Quant 得分 161.97。Expert Gather 的关键假设是将 token–hidden 空间映射为二维 program grid，移除单 program 内的串行 block loop，并在局部重用 top-k metadata。Act Quant 则将 row/group 映射为独立工作单元，在一个常驻 tile 内完成 absmax、scale 和类型转换，减少中间 tensor

的读写。

这两项的高分并非单纯来自增大 block。真正的变化是将串行循环中的逻辑工作提升为 NPU 可以并行调度的 program。这也说明了为什么源码结构提示比通用的"尝试更大 BLOCK"更有效：前者指向了并行度不足的根本原因，后者只是修改现有划分的粒度。

8.4 SiLU FP8 Quant 与 State Passing：算术—状态跨层重构

SiLU FP8 Quant 在稳健配置中得分 127.28，State Passing 得分 102.85。前者的搜索重点是将按 expert 串行的 token loop 转化为 token 与 expert-group 的二维并行，并在同一 tile 内融合 SiLU、乘法、FP8 缩放与存储。后者则尝试用一个 batch-head program 中的常驻 dimension lanes 摊薄 chunk recurrence 的 scalar 开销。

这类变换同时修改 private kernel 参数、grid 和 pointer rank，因而比参数调优更容易触发后端限制。静态边界会拒绝不安全的 rank-two pointer broadcast、非法 tensor index 和运行时 static range。性能分布中仍然存在形状间差异，说明一种 grid 不一定适合所有工作集。更稳健的方向是让模型生成可验证的 fast path 与 shape-safe fallback，再由 evaluator 检查所有分支。

8.5 Cache Quantize：高性能与共性数值错误

Cache Quantize 得分 138.29，说明它的性能上界很高；但在两个 Case 上，四个候选都出现 accuracy failure。这是"文本多样性不等于语义多样性"的典型例子。多个候选可能存在不同的变量名和 block 大小，却使用了相同的 scale 定义、舍入顺序或 page stride 假设。

量化路径的主要风险包括：scale 的统计范围与 dtype，零值 group 的 reciprocal-safe denominator，rounding 和 clipping 的先后顺序，typed pointer 下的 byte/element stride，以及 page layout。后续 archive 因此将 error signature 纳入多样性距离：一个激进融合候选可以保留，但其他位置应分配给不同 scale/layout 假设和更保守的 fallback。

8.6 Expert Count：空输入与规约编译边界

Expert Count 在稳健配置中得分 2.95，五个候选在同一类输入上全部出现 runtime error。该算子包含 token assignment 遍历、per-expert count 和可能的空输入。空

输入对 NPU profiler 尤其特殊：host 侧直接返回空张量不会产生可测 Kernel，而零 grid 又可能导致 runtime 或采集失败。

相应的通用修复不是针对某个 Case 名分支，而是使 BLOCK 和 launch grid 至少为 1，对有效 token 使用 mask，并让每个 expert 由稳定 program 完成一次最终 store。在这类 Kernel 上，首先保留一个 conservative non-atomic 候选比同时导出五个高风险变体更合理。

8.7 Set-K-and-S：大形状、padding 与零启动

Set-K-and-S 在稳健配置中得分 5.89，两个候选在同一边界类形状上出现 accuracy failure。该算子的一般性风险来自 padding slot、zero page、typed stride 和大 launch geometry。系统从函数签名和 reference 推导 optional/padding/stride 语义，通过边界变体验证，而不把任何测试编号写入候选。

该案例说明，“grid 至少为 1”与“对无效位置不做写入”应该被分为两个不变量。前者保证 runtime/profiler 能看到有效 launch，后者保证数据语义；把它们混成一个特殊分支容易再次引入边界错误。

8.8 Selective Scan：递推状态与输出语义

Selective Scan 是稳健收口配置中唯一的零分 Kernel，两个候选在三类 Case 上均出现 accuracy failure。它同时包含状态递推、可选 D/z epilogue、padding slot 和 persistent state 读写。对这种 Kernel，简单交换计算顺序或重用一个 mask 就可能改变整个状态语义。

一个典型问题是：padding mask 可以禁止持久状态的 load/store，但最终 output 仍可能需要使用“零初始 transient state + optional epilogue”计算。如果同一 mask 直接屏蔽整个 output store，结果就会与 reference 不同。更可行的搜索方式是将不变量拆为三层：状态读取语义、recurrence 更新顺序、输出/epilogue 语义，每次变异最多改变一层的工作划分。

8.9 长尾预算分配

根据稳健配置的失败密度和得分分布，后续搜索的优先级为：Selective Scan 的正确性拆解，Expert Count 的空输入与 runtime，Cache Quantize 的错误多样性，Set-K-and-S 的边界形状，其次才是已经正分但性能较低的长尾 Kernel。这个排序只决定不同 Kernel 的计算预算，候选本身仍由当次 Agent 自动生成与评价。

第九章　复现协议与运行指南

9.1　复现目标

本项目的复现包含三个层次。第一层是控制面：代码能被 Python 3.10 导入，根级 facade、搜索状态机、评价器和打包器通过本地测试。第二层是单 Kernel NPU 闭环：Agent 能生成非原始候选，完成真实编译、正确性 Case 和 fresh profiler 测量。第三层是全量交付：21 个 Kernel 依次完成搜索，生成 completed manifest、统一目录和确定性 ZIP。

由于 LLM sampling、服务延迟与 NPU 负载存在随机性，程序复现不要求每次生成字节级相同的 Kernel。合理的复现目标是：系统能稳定完成自动搜索，生成 1–5 个非原始候选，对正确性失败做失败关闭，并在同类环境中获得相近的 Kernel 覆盖与性能分布。

9.2 软硬件矩阵

表 **9.1:** 建议复现环境

类别	环境
操作系统	openEuler aarch64
CPU/内存	32 核 CPU，32 GB 内存（或更高）
NPU	昇腾 A2/A3 平台；开发验证使用 1×910B3 芯片
Python	3.10.0
PyTorch	2.6.0
torch-npu	2.6.0rc1
Triton	3.4.0.dev2026011122
CANN	8.3.RC1.alpha003
npuir	1.0.0
模型	`deepseek-v4-flash`、`qwen3.6-flash`；可选 `deepseek-v4-pro`
模型协议	OpenAI-compatible `/v1/chat/completions`

如果使用的是已预装 CANN/Torch/Triton-Ascend 的镜像，不应用普通 PyPI wheel 覆盖这三者。仓库中的 Python lock 主要固定 Agent 控制面依赖；NPU 编译与运行时以官方环境为准。

9.3 安装与控制面校验

Listing 9.1: 安装控制面依赖

```
git clone <judge-gitlab-repository> ASCEND-OK
cd ASCEND-OK

uv sync --frozen
uv run python scripts/release/verify_public_tree.py --root .
```

校验脚本检查根级导入面、Python 编译、关键测试、候选命名、打包器、公开文件布局与 secret patterns。这一步不需要 NPU，用于快速排除克隆不完整、依赖不一致和打包契约破坏。

9.4 模型与 NPU 环境配置

Listing 9.2: 运行时环境变量

```
# Source the installed CANN environment first.
source /usr/local/Ascend/ascend-toolkit/set_env.sh

export ASCEND_OK_MODEL_ENDPOINT="<openai-compatible-base-url>"
export ASCEND_OK_MODEL_API_KEY="<runtime-secret>"
export ASCEND_OK_MODEL_NAMES="deepseek-v4-flash,qwen3.6-flash,deepseek-v4-
    pro"
export ASCEND_OK_TRACE_DIR="/tmp/ascend-ok-traces"

python - <<'PY'
import torch
import torch_npu
import triton
print("torch", torch.__version__)
print("torch_npu", torch_npu.__version__)
print("triton", triton.__version__)
print("npu_count", torch.npu.device_count())
PY
```

密钥不要写入 shell 脚本、.env 或版本库。如果开发机通过 HTTP/SOCKS 转发访问模型服务，应在当前 shell 使用既有网络配置，不需要修改 Agent 代码。

9.5 单 Kernel Smoke Test

全量运行前建议先选择一个访存/算术结构较简单的 Kernel 执行 smoke test：

```
uv run python main.py \
  --input-dir /path/to/official/datasets \
  --output-dir /tmp/ascend-ok-smoke \
  --kernel _act_quant_kernel \
  --max-generations 1 \
  --population-size 2
```

成功的 smoke test 应该产生非原始候选、评价 trace 和统计文件。检查重点包括：测试确实导入新候选，模型凭据没有出现在子进程环境和日志中，正确性失败不会启动 profiler，候选 identity 与文件哈希一致。

9.6 全量运行与打包

Listing 9.3: 21 Kernel 全量搜索与打包

```
run_root=/tmp/ascend-ok-full-run

uv run python -m ascend_ok.submission.runner \
  --input-dir /path/to/official/datasets \
  --output-dir "$run_root/output" \
  --archive "$run_root/submission.zip" \
  --receipt "$run_root/receipt.json"
```

运行完成后，评审人员可以检查：(1) batch manifest 状态为 completed；(2) 共有 21 个 Kernel 结果；(3) 每个成功目录包含 1–5 个编号候选；(4) receipt 中的 archive SHA-256 与实际文件一致；(5) 用新目录解压 ZIP 后，Python 文件列表与 manifest 一致。

9.7 可确定与非确定部分

输入发现顺序、保守变换、参数枚举、指纹、tie-break、manifest serialization 和 ZIP bytes 是确定的。LLM sampling、服务延迟、NPU 负载与统一门户任务集是非确定的。这个边界意味着，相同的代码与模型配置能复现同一套搜索逻辑和产物契约，但不保证每一次生成相同的文本。

为减少分数方差，可以在同一软硬件环境中重复运行全量 batch，比较三次运行的 Kernel 覆盖、通过率和逐 Kernel median，而不只比较一个总分。如果某个 Kernel 差异很大，可以从 trace 中区分是模型候选差异、正确性差异，还是 profiler 噪声。

9.8 第三方依赖与信息安全

仓库中的 `docs/ATTRIBUTION.md` 记录第三方仓库、使用方式与 revision。运行时的 API key、私有 endpoint、浏览器 Cookie、SSH 配置和原始门户产物都不是源码仓库的一部分。打包器和 release verifier 都会检查常见私钥、token 和凭据特征，但只返回匹配类别，不在日志中回显敏感字节。

第十章　工程经验、局限与后续方向

10.1　评价闭环比搜索复杂度更重要

一个包含复杂进化操作的 Agent，如果无法让评测系统正确识别候选，仍然只是在错误的反馈上快速迭代。项目中最有价值的第一步是建立一个可以被导入、可以通过真实 Case、可以产生 profiler 数据、可以按统一命名导出的非原始候选。在此之后，每一次门户失败才能被归纳为编译、数值、边界或性能问题。

10.2　正确性是性能函数的定义域

将正确性作为 weighted penalty 会产生一个危险的中间状态：一个非常快但只错一个 Case 的候选可能仍然超过较慢的全正确候选。ASCEND-OK 将性能函数只定义在 $C(x) = 1$ 的集合上。这一选择使 archive、Top-5 和最终打包的语义简单了很多：没有“带风险的高分”，只有“正确候选中的性能和多样性排序”。

10.3　LLM 同时需要结构上下文与硬边界

只给模型一组禁止规则，往往会得到保守但收益很低的候选；只给它性能建议，又会产生大量无法编译的代码。实验中更有效的组合是：源码特征驱动的局部假设负责告诉模型“值得改什么”，静态边界和 staged evaluator 负责判断“这次改动能不能留下”。两者的结合比单纯增加 token 或种群大小更能提高有效搜索密度。

10.4　多候选的价值来自错误多样性

同一父代、同一 prompt 或同一 scale/layout 假设产生的多个候选，可能在文本上差异很大，却在隐藏形状上一起失败。Cache Quantize 的结果尤其明显。因此，候选距离不应只使用源码 hash，还应包含 parent lineage、operator family、AST feature delta、失败 Case 集合、异常类型和数值误差形态。五个候选覆盖五种风险比覆盖五

个微小参数更有意义。

10.5 当前局限

1. **Selective Scan** 的语义尚未稳定闭合。递推状态、padding 和 epilogue 相互耦合，稳健配置中仍为零分。
2. 错误签名的语义粒度还可以更细。当前能区分 compile/runtime/accuracy 和 Case，但对 scale、stride、mask 等根因仍需模型推理。
3. 门户轮次不是固定消融环境。任务数和 NPU 负载会变化，因此小幅分数差不能等价为某一代码改动的独立收益。
4. 开发芯片与所有目标平台不完全等价。开发使用 910B3 芯片，A2/A3 平台之间的最优 work partition 和管线排序可能不同。
5. 模型采样存在方差。即使 prompt 相同，生成的程序结构也可能不同，需要通过多模型轮换、稳定 archive 和重复运行降低波动。

10.6 后续方向

10.6.1 更细的错误签名与反例最小化

对失败候选自动执行 shape delta debugging，缩小到仍可复现错误的最小输入，再比较父代与候选的中间量。将误差首次出现的阶段压缩为更细签名，可以让模型区分“最终 cast 错误”与“规约本身错误”，减少全函数重写。

10.6.2 可验证的 Fast Path + Fallback 合成

让模型显式返回 shape predicate、fast kernel、fallback kernel 和 wrapper launch 四元组。AST 工具检查公共 ABI、所有分支的 constexpr 一致性和覆盖范围，evaluator 分别对命中/未命中 fast path 的形状运行测试。这能同时保留大形状性能与尾块正确性。

10.6.3 基于 Bandit 的动态预算

将 operator/model/parent lane 视为 arm，用“每秒产生新正确候选的概率”和“每秒带来有效 fitness improvement 的概率”更新后验。当某一 lane 连续生成同类编译失败时，自动降低分配；当某一模型在当前源码结构上连续产生正确改进时，适当增加调用。奖励只使用当次 evaluator 结果，不依赖 Kernel 名和固定历史常数。

10.6.4 A2/A3 多环境校准

在 910B3 芯片和可获得的 A2/A3 目标平台上对同一候选执行 paired measurements，比较逐 Kernel 排序相关性。如果平台间最优 tile 不同，优先让当次 profiler 信号调节搜索，而不在候选源码中按设备型号选择固定算子实现。

第十一章　结论

本文提出并实现了面向 Triton-Ascend 的自适应优化系统 ASCEND-OK。系统以每 Kernel 独立搜索为基本单元，将双父代、允许大模型、结构/参数变异、静态信任边界、全 Case 正确性、边界变体、fresh NPU profiler、错误多样性 archive 和确定性打包连接为一个完整闭环。

该方法的核心不是让模型在一次回答中写出“最好的 Kernel”，而是把开放式程序生成组织到一个可测、可拒绝、可反馈、可收尾的进化过程中。正确性不是与性能并列的惩罚项，而是性能函数的定义域；profiler 不是正确性真值，而是下一代假设的诊断来源；五个候选的价值不在于文件数量，而在于它们覆盖了不同的结构与错误风险。

统一门户实验中，严格受约束的连续搜索阶段最高取得 71 分；在扩大允许模型的探索范围并充分发挥候选池能力时，系统达到 100 分。稳健收口配置取得 70 分、204/225 个任务正确、20/21 个 Kernel 有效。Chunk Cumsum、Unpack Sequence 稳定达到逐 Kernel 上限；Act Quant、Expert Gather、Cache Quantize、SiLU-FP8 与 State Passing 展示了结构级优化的收益；Selective Scan、Expert Count 和量化边界则揭示了状态语义、空输入和错误多样性仍是下一阶段的关键问题。

ASCEND-OK 所形成的约束感知进化方法不只适用于当前 21 个算子。对于更大规模的编译器 Agent、跨后端 Kernel 生成和自动调优系统，同样需要将生成能力、硬件测量、正确性与交付契约放在一个语义不失真的闭环中。

附录 A　官方接口与运行参数

A.1　根级兼容接口

最终仓库保留以下导入，不要求调用方知道 src layout：

```
from config import EAConfig
from executor import EvaluationResult, TritonExecutor
from optimizer_agent import TritonOptimizerAgent, get_baseline_from_json
```

标准 CLI：

```
python main.py \
  --input-dir /path/to/official/datasets \
  --output-dir /tmp/ascend-ok-output \
  [--kernel KERNEL] \
  [--population-size N] \
  [--max-generations G] \
  [--debug]
```

完整 direct + package CLI：

```
python -m ascend_ok.submission.runner \
  --input-dir /path/to/official/datasets \
  --output-dir /tmp/run/output \
  --archive /tmp/run/submission.zip \
  --receipt /tmp/run/receipt.json
```

A.2　环境变量

表 **A.1:** 正式路径环境变量

变量	默认/要求	用途
ASCEND_OK_MODEL_ENDPOINT	required	OpenAI-compatible base URL
ASCEND_OK_MODEL_API_KEY	required	运行时注入密钥，不记录值
ASCEND_OK_MODEL_NAME	deepseek-v4-flash	单模型名称
ASCEND_OK_MODEL_NAMES	empty	逗号分隔的允许多模型
API_URL/API_KEY/ENGINE	compatible	官方参考变量别名
ASCEND_OK_TRACE_DIR	disabled	原子候选 trace 根目录
ASCEND_OK_AUXILIARY_TEST_CODE_PATH	disabled	额外 source/reference 派生测试
ASCEND_OK_SOURCE_DERIVED_TEST_VARIANT	disabled	启用自动形状边界变体
ASCEND_OK_PARAMETER_CANDIDATE_LIMIT	6	参数候选数，范围 0–12
ASCEND_OK_MIN_OUTPUT_RELATIVE_FITNESS	0.5	输出候选相对性能阈值
ASCEND_OK_KERNEL_PRIORITY	lexical	只改变 Kernel 调度顺序，不选候选

A.3 Manifest 关键字段

Listing A.1: 脱敏后的 batch manifest 结构示意

```
{
  "schema_version": 3,
  "status": "completed",
  "results": [{
    "kernel": "<kernel>",
    "correct": true,
    "submission_eligible": true,
    "run_id": "<fresh-run-id>",
    "selection_mode": "direct_agent_top5",
    "deadline_seconds": 1200,
    "elapsed_seconds": 1187.2,
```

```
    "llm_stats": {"call_count": 12, "total_tokens": 123456},
    "models": ["deepseek-v4-flash"],
    "input_sha256": ["<sha256>", "<sha256>"],
    "candidates": [{
      "file": "<kernel>_v1.py",
      "sha256": "<sha256>",
      "id": "<same-sha256>",
      "fitness": 1.23,
      "generation": 2,
      "origin": "deepseek-v4-flash"
    }]
  }]
}
```

partial failure 记录只保留 Kernel、correct 与 eligibility，不暴露可被误打包的候选字段。

A.4 Stable error codes

自动打包器使用稳定错误类别，例如 BATCH_MANIFEST_INVALID、KERNEL_RESULT_INELIGIBLE、MODEL_USAGE_INVALID、ORIGINAL_CANDIDATE、UNBOUND_VARIANT、KERNEL_TIME_LIMIT_EXCEEDED 和 SECRET_DETECTED:*。错误输出不包含 credential bytes 或候选全文。

附录 B　代表性实验详表

B.1　稳健收口配置的逐 Kernel 结果

表 **B.1:** 70 分稳健收口配置的逐 Kernel 结果

Kernel	得分	平均 speedup	范围
_chunk_cumsum_fwd_kernel	200.00	2.00	2.00–2.00
_unpack_seq_triton_kernel	200.00	2.00	2.00–2.00
_fwd_kernel_ep_gather	162.94	1.63	1.19–2.00
_act_quant_kernel	161.97	1.62	1.25–1.91
_quantize_k_cache_fast_kernel	138.29	1.38	0.15–2.00
_silu_mul_fp8_quant_deep_gemm	127.28	1.27	0.84–1.59
_state_passing_fwd_kernel	102.85	1.03	0.21–1.50
_dequantize_k_cache_fast_kernel	76.91	0.77	0.30–1.62
_rms_norm_kernel	68.73	0.69	0.61–0.85
_log_softmax_kernel	47.65	0.48	0.45–0.52
_pack_seq_kernel	34.00	0.34	0.28–0.43
_convert_req_index_to_global_index_kernel	25.87	0.26	0.26–0.26
_per_token_group_quant_8bit_colmajor	22.08	0.22	0.18–0.24
_chunk_state_fwd_kernel	7.61	0.08	0.00–0.23
_per_group_transpose	7.53	0.08	0.05–0.11
_set_k_and_s_triton_kernel	5.89	0.06	0.04–0.08
_copy_page_indices_kernel	3.03	0.03	0.00–0.05
_count_expert_num_tokens	2.95	0.03	0.01–0.05
_correct_attn_cp_out_kernel	2.27	0.02	0.00–0.07
_trtllm_prefill_attn_kvfp8_dequant	1.68	0.02	0.00–0.05

Kernel	得分	平均 speedup	范围
`_selective_scan_update_kernel`	0.00	0.00	0.00–0.00

B.2 失败分组

表 **B.2:** 稳健收口配置的失败聚类

Kernel	失败数	类别	主要研究方向
Selective Scan Update	6	accuracy	拆分状态读写、recurrence 与 epilogue 不变量
Cache Quantize	8	accuracy	错误签名多样性，检查 scale/round/stride 语义
Expert Count	5	runtime	空输入、非零 grid 与 conservative reduction
Set-K-and-S	2	accuracy	padding、zero page、typed stride 与大形状边界

B.3 结果解读要点

头部七个 Kernel 的逐算子得分都超过 100，是稳健配置的主要性能来源。同时，21 个失败任务全部集中在四个 Kernel，表明下一阶段的首要工作不是对所有算子平均加大搜索，而是对四类系统性语义问题执行反例最小化和错误多样性修复。对已经稳定封顶的 Kernel，只需保留低预算再发现通道。

B.4 重复实验的建议记录字段

为便于对多次 Agent 运行进行同口径比较，建议每次保存以下信息：软硬件版本、模型列表、输入哈希、单 Kernel 墙钟时间、模型调用数、候选总数、正确候选数、Top-5 指纹、正确性 Case 结果、fresh profiler 时延、管线摘要、最终 archive 哈希和统一门户返回。如此可以将分数差异定位到生成、测试、测量或门户任务集，而不仅仅得到一个无法解释的总分。

参考文献

[1] 2026 年全国大学生计算机系统能力大赛编译系统设计赛组委会. 基于进化算法的 Triton 自动优化系统：赛题技术方案, 2026.

[2] 2026 年全国大学生计算机系统能力大赛组委会. 编译系统设计赛章程, 2026.

[3] Philippe Tillet, H. T. Kung, and David Cox. Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019.

[4] Tianqi Chen, Thierry Moreau, Ziheng Jiang, et al. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation*, 2018.

[5] Lianmin Zheng, Chengfan Jia, Minmin Sun, et al. Ansor: Generating High-Performance Tensor Programs for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation*, 2020.

[6] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, et al. OpenTuner: An Extensible Framework for Program Autotuning. In *23rd International Conference on Parallel Architectures and Compilation Techniques*, 2014.

[7] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[8] Chris Cummins, Volker Seeker, Dejan Grubisic, et al. Large Language Models for Compiler Optimization. arXiv preprint arXiv:2309.07062, 2023.

[9] Meta AI Compiler Team. Meta Large Language Model Compiler: Foundation Models of Compiler Optimization. arXiv preprint arXiv:2407.02524, 2024.

[10] Chengrun Yang, Xuezhi Wang, Yifeng Lu, et al. Large Language Models as Optimizers. In *International Conference on Learning Representations*, 2024.

[11] Samuel Lange, Tom Schaul, Yutian Chen, et al. A Self-Improving Language Model for Evolutionary Program Synthesis. In *International Conference on Machine Learning*, 2025.

[12] Triton-Ascend Contributors. Triton-Ascend source repository. Revision `a60006ac63f3f5068cc4b217ebd4e14ce0801f3e`, accessed 2026.

[13] Competition Framework Contributors. Evolutionary-Algorithm-Based-TritonAscend-Optimization. Revision `6ce8f112e986a335a60ae545f82f73490a825ac6`, accessed 2026.