国产算力平台大模型推理引擎综述

A Survey of Large-Model Inference Engines on Domestic Computing Platforms

张书豪 [1]，刘世峰 [1]，杜雨枫 [1]，马俊豪 [1]，邱瑞杰 [1]，廖小飞 [1]，金海 [1]

Shuhao Zhang[1], Shifeng Liu[1], Yufeng Du[1], Junhao Ma[1], Ruijie Qiu[1], Xiaofei Liao[1], Hai Jin[1]

[1] 华中科技大学计算机科学与技术学院，大数据技术与系统国家地方联合工程研究中心，服务计算技术与系统教育部重点实验室，集群与网格计算湖北省重点实验室，武汉 430074

[1]School of Computer Science and Technology, Huazhong University of Science and Technology, National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, Wuhan 430074, China

通信作者：廖小飞，E-mail：xfliao@hust.edu.cn

Corresponding author: Xiaofei Liao, E-mail: xfliao@hust.edu.cn

摘 要： 本文面向国产算力平台上的大模型推理系统，不把综述写成项目名录，而是围绕“生态格局与路线分化–分析骨架与核心架构–基于骨架的路线比较–由路线比较收束出的关键挑战–未来收敛方向”这一主线，讨论国产芯片、编译栈、运行时与部署环境如何共同重塑推理引擎的设计边界。本文以近年可交叉核验的公开证据为基础，综合系统论文与技术报告、官方文档与开源仓库、标准或白皮书，以及少量仅用于补充真实摩擦点的社区部署记录；对仅见于单次演示、新闻稿或未公开复现链路的材料，不直接据此给出性能与成熟度结论。基于这一证据边界，本文首先梳理当前公开生态为何分化为开源主干复用增强路线、平台一体化方案与国产原生/独立路线三类，继而以统一抽象、统一指标与可观测证据链为底座，以执行与通信、状态治理、压缩与成本协同三条主路径为分析骨架，比较不同路线究竟把复杂度主要压在 backend、platform runtime、service state machine 还是 toolchain/control plane。进一步地，本文将平台异构、复杂任务负载、压缩收益不稳定、上游快速演进与评测体系失真等问题收敛为若干结构性挑战，强调国产推理系统的真正竞争点不在局部算子是否足够快，而在平台能力能否沉淀为契约、运行时状态能否被统一治理、成本与交付能否进入同一闭环。最后，本文结合 vLLM-HUST 这类基于 vLLM 的本土增强实践以及 vLLM-Ascend、MindIE、LMDeploy、Chitu、xLLM 等对象，讨论平台抽象、状态中心化运行时、控制面治理与模型结构反向塑造系统边界的未来方向，以期为国产推理系统的比较分析与架构演进提供更可复核的讨论框架。

关键词： 国产算力平台；大模型推理；推理引擎；运行时；软硬协同

Abstract: This survey examines large-model inference systems on domestic computing platforms through a writing spine that moves from ecosystem divergence to analytical framing, route comparison, structural challenges, and future convergence. Rather than enumerating projects, it builds its discussion on cross-checkable public evidence from system

papers and technical reports, official documentation and open-source repositories, standards or white papers, and a limited set of community deployment records used only to expose recurring engineering frictions. Materials that appear only in one-off demos, news releases, or non-reproducible validation reports are not used as standalone evidence for performance or maturity claims. On top of this evidence boundary, the survey explains why the public ecosystem has split into three broad routes: upstream-core reuse and enhancement, vertically integrated platform stacks, and domestic-native or standalone systems. It then adopts an analytical scaffold built on a unified abstraction, metric, and observability base together with three main paths for execution and communication, state governance, and compression-cost co-optimization, and uses this scaffold to compare where different routes place their primary complexity: backend adaptation, platform runtime, service state machine, or toolchain and control plane. Based on this comparison, the survey further distills structural challenges involving hardware heterogeneity, complex workloads, unstable compression gains, rapid upstream evolution, and evidence gaps between benchmarks and production. The core argument is that the decisive issue is not isolated kernel speed, but whether platform capabilities can be turned into stable contracts, runtime states can be governed coherently, and cost, evaluation, and delivery can be brought into one engineering loop. Drawing on localized enhancement practices such as vLLM-HUST together with systems including vLLM-Ascend, MindIE, LMDeploy, Chitu, and xLLM, it finally outlines future directions around platform abstraction, state-centric runtimes, control-plane governance, and the continuing feedback from model evolution into system boundaries.
Keywords: domestic computing platforms; large-model inference; inference engine; runtime systems; hardware-software co-design

1 引言

大模型应用的快速扩张，正把推理系统从单机实验环境推向面向生产的高并发服务平台。随着国产 AI 加速芯片、服务器整机与集群软件生态逐步成熟，围绕国产算力构建高性能、可维护、可扩展的推理引擎，已成为学界和产业界共同关注的方向。但“国产算力推理引擎”并不是把既有 serving 框架迁移到另一类设备上的简单重复。相较通用硬件环境，国产平台往往同时叠加设备能力离散、编译链约束差异、运行时治理复杂和交付路径多样等因素，使许多原本可以局部吸收的问题迅速上升为系统级问题。平台边界、运行时状态与工程闭环一旦同时变化，推理系统的设计重心也会随之迁移。与训练系统相比，推理引擎更关心动态批处理、显存复用、流式时延、长上下文，以及工具调用、多模态和结构化输出下的稳定性；对国产平台而言，还要额外处理硬件后端差异、算子完备性、编译运行时兼容性，以及上游框架快速演进带来的维护压力。

近年来，以 vLLM 为代表的推理框架通过 PagedAttention、连续批处理与高效内存管理显著改写了大模型服务系统的设计范式，SGLang 等系统则进一步强调请求编排、结构化控制流和复杂工作负载下的执行效率 [1–3]。通用 serving 文献也表明，推理系统的研

究重心正在持续外移：Orca 与 Sarathi-Serve 说明，迭代级调度、连续批处理和 chunked prefill 已成为高吞吐服务的基础组织方式 [4, 5]；DistServe 与 Splitwise 表明，prefill 和 decode 的阶段解耦正在从局部优化上升为整体 goodput 设计问题 [6, 7]；XGrammar 与 ElasticMM 则进一步把演进推向结构化执行与多模态并行，说明推理引擎正在从 token 生成器演变为状态密集型运行时 [8, 9]。这意味着，当推理系统进入国产芯片、国产编译运行时和本土部署环境后，关键问题会迅速从单一硬件后端优化转向统一框架抽象与本土设备能力的持续对齐。此时，调度、缓存、多模态和结构化执行的可迁移性、可维护性与稳定性，往往比单点优化更重要。从产业侧看，国产算力推理引擎面向的也早已不只是学术基准测试，而是在线问答、代码助手、政企知识服务、具身智能中间层和 Agent 工作流等多种 AGI4S 场景；这些场景既要求较高的吞吐密度和资源利用率，也要求流式首 token 时延、上下文切换开销、错误恢复能力和服务可观测性达到生产标准。因此，国产推理引擎的竞争焦点正从“能否完成基础部署”转向“能否在真实业务下稳定、高效且可持续演进地运行”。

在这一背景下，开源主干复用增强路线、平台一体化方案和国产原生/独立路线虽然都在解决国产部署问题，但三者分化的根源并不只是优化手段不同，更在于对系统边界和演进节奏的假设不同：前者强调上游复用，并允许在共享主干外侧持续注入平台与场景增强，其中既包括直接围绕主线后端边界展开的适配，也包括以 vLLM-HUST 为代表的本土增强实践；中者强调平台统一性交付；后者则更侧重围绕自研状态机、缓存治理和部署闭环组织系统能力。相较训练侧更容易用 FLOPS、集群规模等指标描述阶段进展，推理侧的核心矛盾更多分布在请求形态、缓存复用、阶段干扰、故障恢复与升级节奏等难以被单一数字概括的维度上。尤其当模型族从纯文本扩展到多模态、语音和 Agent 工作流后，系统能否在较长时间尺度内维持接口兼容、性能稳定和运维可控，往往比一次局部 benchmark 的领先更能决定其工程价值。基于此，全文采用“一个底座、三条主路径”的分析骨架：底座是统一抽象、统一指标和可观测证据链，用来支撑多平台、多版本和多阶段优化条件下的可组合、可比较与可复现；三条主路径分别是执行与通信、状态治理、压缩与成本协同，对应执行骨架收敛、长上下文与高并发下的状态组织，以及单位 token 成本进入系统闭环。在此基础上，后文依次讨论生态格局与路线分化、分析骨架与核心架构、基于骨架的路线比较、由路线比较收束出的关键挑战，以及未来可能的收敛方向。全文重点不在于为某一路线背书，而在于识别哪些能力应沉淀为能力契约，哪些状态必须纳入运行时统一管理，哪些优化适合以插件或外围工具链独立演进，以及哪些 benchmark 足以支撑路线选择。

为使讨论建立在可交叉核验的材料之上，本文还对讨论范围和证据边界作如下限定。首先，本文的“国产算力推理引擎”既包括直接面向国产芯片的原生推理系统，也包括基于主流 serving 主干的国产后端适配、面向多类国产硬件的部署工具链，以及围绕服务状态机、缓存治理和交付闭环组织的独立运行时；但不把单纯的硬件整机、一体机交付页面或应用层封装直接等同于推理引擎本体。其次，本文优先使用四类证据：系统论文与技术报告用于界定机制与设计目标，官方文档与开源仓库用于确认能力边界和工程入口，标准或白皮书用于辅助描述生态收敛趋势，社区部署记录仅作为观察真实摩擦点的补充材料，而不单独支撑性能结论。再次，文中对路线比较主要围绕可演进性、状态治理能力、复杂负载承载能力和交付确定性展开，而不试图在缺少统一复现实验条件的前提下给出跨平台绝对性能排序。本文更关注复杂度被压在哪里、证据是否足以支撑相关判断，

以及系统是否具备可持续演进边界，而不是把不同公开口径简单折算为单一优劣结论。

1.1 研究对象与证据方法

本文的综述对象，限定为近年在公开论文、技术报告、官方文档或开源仓库中能够形成交叉验证的国产算力推理相关系统与工程实践。就时间范围而言，文章重点覆盖大模型 serving 主干与国产后端适配快速演进的近年公开材料，并在必要处回溯少量更早的基础工作，用于交代连续批处理、PagedAttention、阶段解耦和结构化执行等机制来源。就对象纳入而言，只有当某一对象至少能够在“机制说明”“工程入口”“公开实现”三者中满足其中两项时，本文才将其纳入路线比较或章节主线；若某材料仅以新闻稿、发布会口径、单次演示或未公开复现链路的技术验证出现，则只作为生态线索或趋势性观察引用，不直接据此给出成熟度与性能判断。

基于这一原则，本文将证据分为四层使用。第一层是系统论文与技术报告，用于界定执行、状态、压缩和评测等机制问题的研究脉络；第二层是官方文档、源码仓库与安装部署入口，用于确认对象的实际能力边界、组件关系与可进入性；第三层是标准、白皮书与生态报告，用于辅助刻画国产生态的分工变化与标准化趋势；第四层才是社区部署记录和兼容性经验，它们仅用于补充真实工程摩擦点，而不单独支撑横向性能结论。按这一分层，正文中的比较表与路线判断优先回答三类问题：一是复杂度主要被压在何处，二是哪些系统边界已经具有较稳定的公开证据，三是哪些判断仍应保留为条件性的工程观察。这样做的目的，不是把本文伪装为严格元分析，而是把综述中的描述性事实、比较性判断与趋势性推断尽量分层组织，使不同性质的材料不被直接并列。

2 面向国产算力的大模型推理引擎生态

面向国产算力的大模型推理引擎生态并不是单一软件系统，而是由硬件厂商基础软件、开源推理框架适配层、算子与编译栈、以及上层服务化平台共同构成。根据公开仓库与官方文档，可以将相关工作大致归纳为三类：第一类是硬件厂商自研推理引擎或原生后端，例如华为 Ascend MindIE、寒武纪 MagicMind、摩尔线程 MT-Transformer、天数智芯 IxRT/IGIE 等；第二类是围绕 vLLM、SGLang、LMDeploy、FastDeploy 等主流大模型推理框架的国产硬件适配；第三类是 DLinfer、Triton-Ascend、Xinference、GPUStack 等中间层或服务化平台，用于降低上层框架、算子库、图编译器与硬件运行时之间的耦合。

从公开生态看，vLLM 的硬件插件化正在成为国产算力适配的主要形态之一。vLLM-Ascend、vLLM-MLU、vLLM-MUSA、vLLM-MetaX、vLLM-Kunlun、vLLM-GCU 等项目均采用或参考 vLLM 的硬件插件机制，使厂商能够在尽量不侵入 vLLM 主干代码的情况下接入各自的 NPU、MLU、GPU、GCU 或 XPU 后端。这一路径的优势在于能够复用 vLLM 已有的连续批处理、KV Cache 管理、OpenAI 兼容服务接口、量化与投机解码等推理框架能力；挑战则在于底层算子、通信、图捕获、显存/片上内存管理、版本依赖和模型覆盖度仍需要各厂商持续适配。

华为 Ascend 生态是目前公开资料较为完整的代表。一方面，MindIE 作为面向 Ascend 的 AI 推理加速套件，覆盖大模型推理、服务化和 PyTorch 生态适配等能力；另一方面，vLLM-Ascend、SGLang-Kernel-NPU 和 Triton-Ascend 分别从推理框架插件、SGLang 算子库和 Triton 编译后端三个层面补充了开源生态。这说明国产算力推理引擎的竞争点已

经从单一模型执行扩展到框架调度、KV Cache、MoE 通信、算子融合、编译优化和服务化接口等完整技术栈。

除单一芯片生态外，跨硬件适配层也在快速发展。上海人工智能实验室 DeepLink 生态中的 DLinfer 试图在上层 LLM 推理框架与下层厂商融合算子或图引擎之间定义统一接口，LMDeploy 则通过 PyTorch Engine 和 DLinfer 等路径支持多类国产硬件。百度飞桨 FastDeploy 以 PaddlePaddle 为基础，面向文心等大模型提供推理部署能力，并在官方文档中列出对昆仑芯、Ascend、海光、天数智芯、沐曦、燧原等硬件的适配说明。这类项目的共同目标是减少每个推理框架分别对接每个硬件后端的重复工程成本。

表 1: 面向国产算力的大模型推理引擎与适配生态

组织/单位	面向算力	代表项目/引擎	公开形态与技术关注点
华为 Ascend 生态	Ascend NPU，Atlas 推理/训练服务器	MindIE，vLLM-Ascend，SGLang-Kernel-NPU，Triton-Ascend	覆盖厂商推理套件、vLLM 硬件插件、SGLang NPU kernel 与 Triton 后端。技术关注点包括 CANN/torch-npu 适配、服务化推理、MoE kernel、注意力算子、编译后端和硬件插件化接口 [10–13]。
SGLang 社区与 Ascend 适配	Ascend NPU	SGLang-Kernel-NPU，DeepEP-Ascend	面向 SGLang 的 Ascend NPU kernel 库，包含专家并行通信、attention、normalization、activation、LoRA 等推理关键算子，适合在综述中归入高性能 serving 框架的国产硬件 kernel 适配方向 [12]。
上海人工智能实验室 DeepLink / OpenMMLab / InternLM 生态	多类国产加速器，包括 Ascend、Cambricon、MetaX 等	DLinfer，LMDeploy	DLinfer 面向 LLM 推理框架和厂商算子/图引擎之间的解耦，LMDeploy 则提供 TurboMind 与 PyTorch Engine 两类推理引擎，并通过构建目标或适配层支持多种国产硬件 [14–16]。
百度飞桨 / 昆仑芯生态	Kunlun XPU，同时覆盖多种国产硬件	FastDeploy，vLLM-Kunlun	FastDeploy 面向大模型和多模态模型的生产级部署，支持 PD 分离、统一 KV Cache 传输、OpenAI/vLLM 兼容接口、量化和投机解码等能力；vLLM-Kunlun 则以 vLLM 硬件插件形式支持昆仑 XPU[17, 18]。
寒武纪	Cambricon MLU	vLLM-MLU，MagicMind，CNServing	vLLM-MLU 基于 vLLM 插件系统在 MLU 硬件上提供大模型推理服务能力；MagicMind 是寒武纪面向 MLU 的推理加速引擎，负责模型转换、图优化、代码生成和部署 [19, 20]。

组织/单位	面向算力	代表项目/引擎	公开形态与技术关注点
摩尔线程	MUSA GPU	vLLM-MUSA，vLLM-MTT / MT-Transformer	vLLM-MUSA 强调与 vLLM 插件体系和 MUSA 软件栈集成；vLLM-MTT 则基于摩尔线程自研 MT-Transformer 后端，强调面向 Transformer 推理的底层算子融合和定制优化。二者体现了“通用兼容插件”和“高性能原生后端”两条路线 [21, 22]。
沐曦集成电路	MetaX GPU，MACA 软件栈	vLLM-MetaX，LMDeploy/DLinfer 适配	vLLM-MetaX 通过 MACA 类 CUDA 后端接入 vLLM，目标是在沐曦 GPU 上获得接近 CUDA 生态的开发体验；DLinfer 也将 MetaX 作为其支持的硬件后端之一 [14, 23]。
燧原科技	Enflame GCU，S60/L600 等	vLLM-GCU，FastDeploy-Enflame	vLLM-GCU 面向燧原 GCU 适配 vLLM，支持量化、Fused MoE、Multi-LoRA、自动 Prefix Cache、Chunked Prefill、Speculative Decoding 等推理特性；FastDeploy 也提供燧原硬件部署说明 [17, 24, 25]。
天数智芯 / DeepSpark 生态	Iluvatar CoreX、智铠/天垓系列	IxRT，IGIE，DeepSparkInference，FastDeploy-Iluvatar	IxRT 和 IGIE 分别面向高性能推理运行时和基于 TVM 的通用推理引擎，DeepSparkInference 提供包括 ChatGLM、Baichuan、Qwen 等模型在内的推理示例；FastDeploy 也提供 Iluvatar CoreX 安装与部署路径 [26, 27]。
海光信息	Hygon DCU，K100-AI 等	FastDeploy-Hygon DCU	官方 FastDeploy 文档给出了在海光 DCU 上部署 ERNIE 系列模型的示例，说明该方向已有面向大模型推理的公开部署路径。但公开资料更多体现为部署适配和示例，成熟度仍需通过复现实验进一步判断 [17, 28]。
壁仞科技	Biren SUPA / BR 系列加速器	BIRENSUPA，EngineX-Biren vLLM 适配	壁仞公开开发者生态展示了 BIRENSUPA 软件平台；同时公开模型社区中存在基于 vLLM OOT 插件的壁仞适配项目。由于公开资料相对有限，综述中宜将其列为待持续跟踪的适配生态，而非直接给出性能结论 [29, 30]。

组织/单位	面向算力	代表项目/引擎	公开形态与技术关注点
瀚博半导体 / Vastai 生态	Vastai GPU，VACC/VUCA 软件栈	Xinference VACC，Vastai 开发者平台	瀚博侧公开资料更多体现为开发者平台和模型生态；Xinference 等服务平台已出现对 Vastai GPU/VACC 的适配支持。该方向可归入国产硬件服务化部署平台生态，而不是单独的核心推理引擎 [31, 32]。
Xinference / GPUStack 等平台	多类异构 GPU/NPU	Xinference，GPUStack	这类项目位于模型服务和集群管理层，通常不直接替代 vLLM、SGLang、LMDeploy 等推理引擎，而是对其进行编排、部署和多硬件管理。它们适合放在“服务化与运维平台”小节中讨论 [32, 33]。

上述生态显示出几个趋势。首先，国产算力适配正在从修改上游推理框架源码转向硬件插件化和后端解耦。这种方式能够降低维护成本，也便于跟随 vLLM、SGLang 等主流推理框架的快速演进。其次，厂商原生推理引擎仍然重要，尤其是在算子融合、图优化、MoE 通信、低精度量化和特定硬件内存层次优化方面，原生后端通常更容易发挥硬件特性。再次，DLinfer、Triton-Ascend、FastDeploy 等中间层说明，国产算力生态正在从单点适配走向“统一接口 + 厂商实现”的模式。

3 分析骨架与核心架构

从系统视角看，大模型推理引擎不是自下而上的模块堆叠，而是一套同时处理执行流、状态流与控制流的运行时。模型执行与算子层决定抽象模型如何映射到具体设备能力，缓存与调度层决定请求到达过程中资源如何被持续重写，而接口与运维层则决定这些内部状态能否以可升级、可观测、可诊断的方式进入真实生产。对国产平台而言，很多看似属于“局部性能”的问题，首先暴露在模块边界处：图模式回退是否需要被调度层感知，多模态前处理应停留在接口层还是进入状态系统，平台插件究竟只负责设备发现还是还要承担环境修复与后端选择。若这些边界定义不清，模型接入、性能优化与工程交付就会反复冲击共享主链。

因此，本文将国产推理引擎的核心架构压缩为“一个底座、三条主路径”。统一抽象、统一指标与可观测底座提供共同语义与证据链；执行与通信、状态治理、压缩与成本协同三条主路径分别回答系统如何运行、如何持有和迁移状态，以及如何在质量约束下控制成本；服务接口与交付闭环则检验这些能力能否稳定进入生产环境。表2 汇总了这一分析骨架下最核心的技术问题；若缺少其中任意一类，分析就容易停留在系统名录而不是系统机制上。

图1 以“服务入口–运行时核心–平台边界”三层压缩全文分析框架，强调国产推理引擎的性能、稳定性与交付能力来自层间联动，而不是某个热点内核的孤立优化。后文据此从最小推理闭环、统一分析底座、三条主路径和服务层收束几个层次展开，讨论国产平台上的复杂度如何被压缩到可治理边界内。

表 2: 国产推理引擎关键技术点总览

技术主路径	关键技术点	典型评价维度
执行与通信	统一执行 IR、动态图与图模式切换、Prefill/Decode 阶段解耦、推测解码协同、拓扑感知通信运行时、计算–通信重叠	MFU、吞吐、TTFT、TPOT、跨节点扩展效率、回退稳定性
状态治理	Prefix/共享状态索引、PagedAttention 与块化缓存、多级存储驻留、状态生命周期管理、跨节点状态迁移、命中感知调度	KV 命中率、迁移流量、恢复时延、显存/主存/NVMe 占用、长上下文稳定性
压缩与成本协同	混合精度量化、KV 动态量化、稀疏–量化协同、低比特算子路径、质量约束下的单位 token 成本优化	精度保持、吞吐/时延变化、状态容量变化、带宽占用、单位 token 成本
统一底座与交付	接口协议、指标口径、trace/log/metrics、回归验证、第三方测试证据链、环境治理与部署控制面	可复现性、可比较性、升级回归成本、交付成熟度

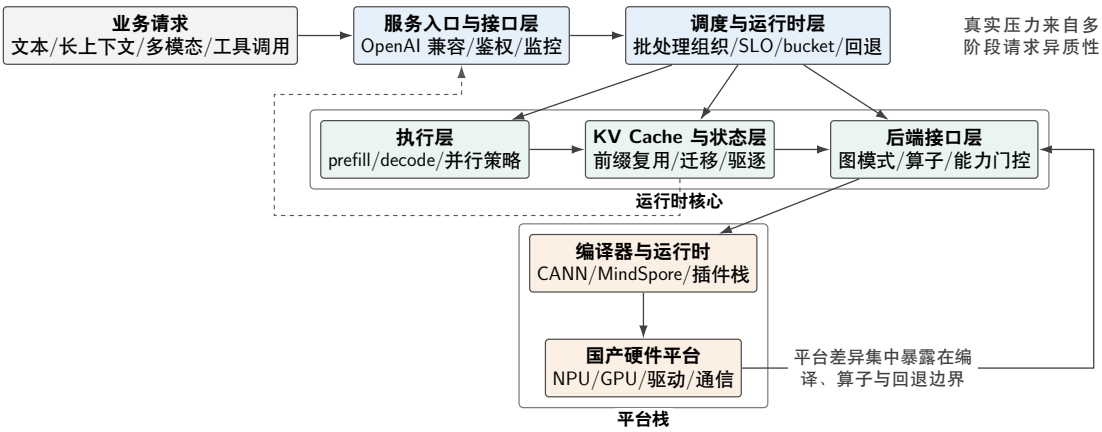


图 1: 国产推理引擎系统设计示意图

3.1 从最小推理闭环到统一分析底座

为便于后文讨论执行、调度和缓存问题，先用当前最常见的 decoder-only 自回归模型概括一次最小推理闭环。输入文本首先经过 tokenizer 切分并映射为 token id，再由 embedding 层转换为连续表示，并与位置信息一起形成 Transformer block 的输入。对本文而言，关键并不在于重复展开 Transformer 的教科书式结构，而在于说明这些计算单元如何被组织成可服务的运行时：多头注意力负责跨 token 建模依赖关系，前馈网络、归一化与残差连接共同完成逐 token 的表示更新。

对推理服务而言，prefill 与 decode 直接决定系统行为。Prefill 对整段提示词并行建模并建立 KV Cache；最后一个位置的 hidden state 经 LM head 和采样策略后产生第一个输出 token；随后 decode 每步只处理新 token 的增量计算和缓存更新，直到满足停止条件。于是，首 token 时延主要受前处理、排队和 prefill 主路径影响，稳定生成速度则更多受 decode 主路径、带宽与缓存组织影响。连续批处理、chunked prefill 和阶段解耦，本质上都在利用前者更接近 compute-bound、后者更接近 memory-bound 这一差异 [4–7]。

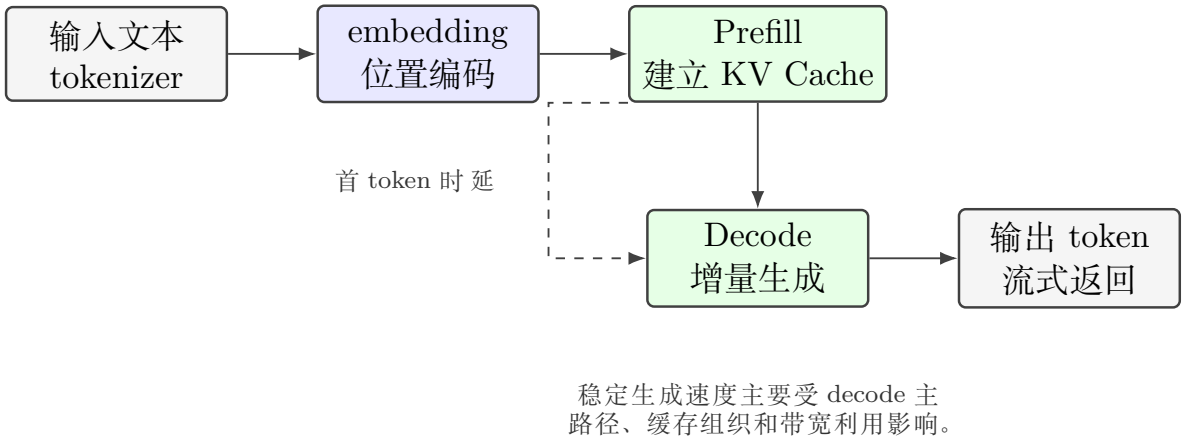


图 2: 从输入到输出的最小推理闭环

推理优化也会很快从“减少重复计算”转向“组织运行时状态”：KV Cache 通过保留历史 Key/Value 避免每一步重算整段上下文，PagedAttention、Prefix Cache、MQA/GQA、KV 压缩与跨实例迁移则进一步决定缓存如何被切块、复用、迁移和回收 [1, 34, 35]；与之并行，FlashAttention 侧重减少注意力计算中的高带宽内存访问，量化则通过降低权重或 KV 精度换取更低的显存与带宽压力 [36–39]。

近年来推理系统研究中的代表性工作呈现出相对一致的演进方向：早期系统更强调单一瓶颈的突破，例如连续批处理、PagedAttention 或注意力 kernel 优化；而随着 Agent、多模态、长上下文与结构化生成逐步进入主流负载，系统设计开始转向跨模块协同，把执行层、缓存层、调度层和约束解码视为一个统一运行时问题来处理 [1, 4, 8, 9, 36]。对于国产算力推理引擎而言，这一变化尤其关键，因为平台差异往往首先放大模块边界上的协同成本，而不是某个单点算子的理论性能损失。但仅有最小推理闭环仍不足以支撑路线比较。国产平台推理系统通常同时面对多类芯片、多套编译链、多种部署模式以及不断变化的模型版本；如果执行事件、状态对象、压缩配置和服务指标没有统一语义，那么即便局部优化已经实现，也很难在跨平台、跨阶段和跨任务条件下形成可比较的证据链。很多团队看似在做性能优化，实际却把大量时间消耗在“指标无法对齐、trace 无法归因、回归结果无法复现”的工程摩擦上。

因此，统一底座的意义在于为后续主路径优化提供一致的能力口径：执行层统一描述 prefill/decode 阶段、图模式切换和通信事件；状态层统一描述前缀命中、驻留层级、迁移开销和生命周期；压缩层统一描述量化配置、精度约束、状态容量变化和单位 token 成本；服务层则把 TTFT、TPOT、吞吐、尾时延和异常回退记录到相同口径中。在这一前提下，不同平台、不同版本和不同优化策略的收益与代价才可能被稳定比较，第三方测试、阶段回归和工程交付也更容易形成可复用证据链。进一步地，高频指标还需要被压缩为可以复用的统一口径：哪些指标用于描述服务体验，哪些指标用于描述状态组织质量，哪些指标用于描述后端执行效率，哪些指标只能在特定负载边界内解释。相关口径被稳定定义后，不同平台、不同执行模式和不同压缩策略之间的比较才不易出现语义漂移。

3.2 主路径一：执行编排、算子与后端能力边界

模型执行层负责组织 Prefill 与 Decode 两类核心路径，协调张量并行、流水并行、专家并行等分布式策略，并在运行时管理设备上下文、流与事件。对于推理系统而言，这一层不

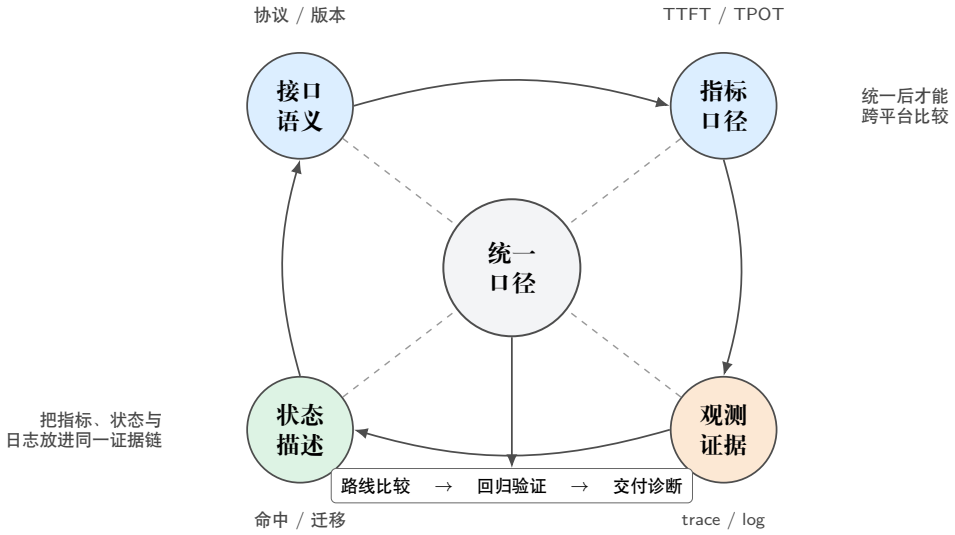


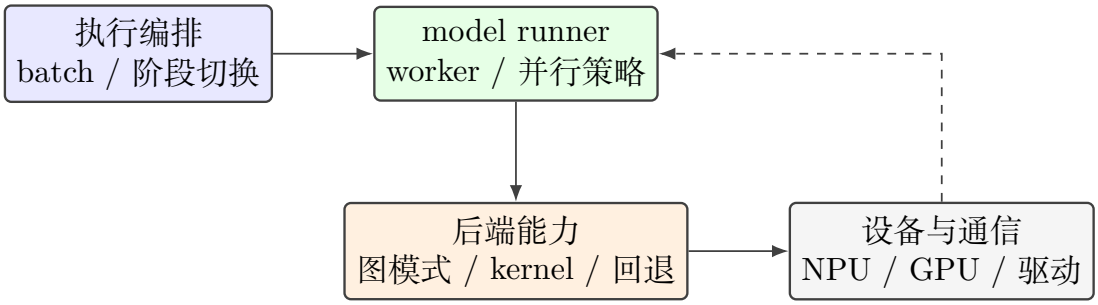
图 3: 统一抽象、指标与观测证据链示意图

仅决定算子调用顺序，还决定请求合并粒度、静态图与动态图切换时机，以及多租户场景中的资源隔离方式。国产算力场景下，该层通常还承担设备能力探测、后端选择与降级回退职责。与成熟 CUDA 生态相比，国产硬件在图编译约束、动态 shape 支持、通信原语完备性以及自定义算子接入方式上差异较大，因此模型执行层往往需要显式管理更多平台相关信息，例如是否启用图模式、哪些路径必须回退到 eager 执行、哪些模型结构需要特殊 patch 才能稳定运行等。执行层因而更容易成为国产适配中产生侵入式分叉的区域。更可持续的组织方式，是将设备能力判断、图模式封装和模型特化逻辑尽量收敛到平台适配层或插件层，而不要直接散落在通用调度逻辑中。这样做虽然会增加抽象设计成本，却能显著降低后续跟踪上游版本时的合并压力。执行层既不能退化为纯粹的设备调用外壳，否则无法吸收模型结构变化；也不能无限吞并平台特例，否则又会成为分叉最严重的区域。让执行层显式依赖平台能力描述、缓存策略接口和调度回调，而不是直接携带某个平台或某类模型的特判实现，新增复杂度就更容易在后续扩展 MoE、长上下文、多模态和语音链路时被映射为局部能力增量，而不是整条执行主循环的持续膨胀。

从更广义的推理系统研究脉络看，执行层与运行时设计一直围绕吞吐、时延与资源受限条件之间的平衡展开。Pope 等对 Transformer 推理可扩展性给出了系统分析，FlexGen 展示了受限显存条件下的卸载与调度策略，StreamingLLM 与 LongServe 则进一步讨论了流式状态维持和长序列服务组织问题 [40–43]。这些研究虽然主要面向通用硬件环境，但其关于运行时状态组织、带宽瓶颈和阶段切分的结论，对国产算力推理引擎同样具有直接参考价值。

执行层今天已经不再只是“执行模型前向计算”的封装层，而更像是协调多类状态生命周期的控制层。一方面，它要感知模型结构本身的变化，例如 MoE 路由、MLA 注意力、视觉编码分支或语音前端引入的新阶段；另一方面，它又要把这些变化映射为平台可接受的执行图、批处理形态和并行策略 [44–48]。如果国产平台上的运行时抽象仍停留在“设备适配 + 算子替换”层面，就很难支撑模型结构快速演化背景下的长期维护。执行层一旦真正落到国产平台，最先被收紧的往往不是调度策略，而是算子、图编译与后端接口这些能力边界。主路径一的难点不仅在于如何组织模型执行，还在于如何把平台

特化加速收敛为可维护的后端契约。



平台差异更适合被封装在后端能力边界内，而不是持续回流到共享调度主链。

图 4: 执行编排与后端能力边界示意图

算子与图编译层是国产推理引擎差异化最明显的部分。大模型推理涉及矩阵乘、归一化、注意力、采样、缓存搬运和多模态编码等多类操作，其中既有适合由厂商编译器或融合引擎统一优化的规律性计算，也有强依赖运行时状态的控制型逻辑。对于国产平台，系统设计者通常要在三种路径之间权衡：直接复用通用框架算子、通过图编译获得整图优化，或为热点路径实现定制算子。

这一层更值得观察的，并不是某个 kernel 在单一芯片上的峰值数字，而是系统是否能够把平台特化加速收敛为稳定的后端能力契约，并像图4 所示那样，把这种差异约束在共享执行主链之外。

这一层的关键不只是单点性能，还包括接口稳定性与维护成本。如果一个后端为了获得局部加速效果而在共享路径中嵌入大量平台条件分支，那么一旦上游框架修改模型执行接口、缓存布局或采样流程，本地后端就会同步承受较高维护负担。因此，面向国产算力的设计应优先使用平台注册表、设备抽象、能力门控和后端插件接口，把“平台差异”封装成可替换模块，而不是扩散成全局条件判断。近期围绕 NVIDIA Hopper 的工作把这种硬件耦合展示得尤为充分：FlashAttention-3 直接利用 Tensor Core 异步执行、TMA 与 FP8 支持提升 H100 上的注意力利用率，而 FlexAttention 则试图在保留 fused kernel 性能的同时恢复 attention 变体的可编程性 [38, 49]。这类工作说明，高性能推理后端的难点往往不在“有没有 kernel”，而在“如何把具体硬件代际能力组织成可持续复用的编译与运行时接口”。

因此，算子和图编译层更需要沉淀的并不是某几个孤立最优 kernel，而是一套更稳定的后端能力契约：哪些算子变体可以被图模式安全覆盖，哪些路径必须保留 eager 回退，哪些低比特或结构化输出逻辑需要在运行时显式感知，哪些优化能够以插件形式随平台演进而单独升级。对国产平台而言，这种契约化表达尤为重要，因为很多问题并不是“后端不可用”，而是“某种能力只在某些约束下可用”。若系统无法表达这种带条件的能力边界，调度器和执行层就只能以越来越多的探测和分支维持稳定，最终把后端问题重新扩散回共享主干。

3.3 主路径二：状态治理、KV Cache 与长上下文组织

状态治理的核心载体是 KV Cache 及其相关元数据系统，它直接影响并发能力、会话恢复成本与长上下文性能。PagedAttention 一类设计通过块化管理显存，降低了连续大块分配的压力，也为前缀复用、分层缓存与按需换入换出提供了实现基础 [1]。在在线服务中，KV Cache 已不再只是“中间结果缓存”，而是决定系统并发上限、会话恢复成本和长上下文单位 token 成本的关键资源。对于国产硬件，状态系统还面临两个额外约束：其一，不同后端对内存分配粒度、数据搬移代价与 host-device 同步开销的敏感度不同，导致在 CUDA 上有效的缓存布局未必可以直接迁移；其二，当系统需要支持 Prefix Cache、Chunked Prefill、分层缓存乃至跨设备迁移时，缓存元数据管理复杂度会显著提升。此时，如果缓存系统与调度策略、图编译策略耦合过深，就容易在长上下文或高并发下出现资源碎片化、性能抖动和回收不及时等问题。相关研究与工程实践已经表明，长上下文场景下的缓存与调度协同是推理效率的决定因素之一。无论是 Orca 对连续批处理和迭代级调度的强调，还是 Sarathi-Serve 对 Chunked Prefill 的系统化组织，都说明内存与调度不能被割裂看待 [4, 5]。

进一步看，围绕注意力高效实现与缓存压缩已经形成一条清晰的技术主线。FlashAttention、FlashAttention-2 与 FlashAttention-3 从算子层逐步降低注意力计算的内存访问代价，并展示了 kernel 设计如何随着 GPU 代际演进而持续贴近具体硬件特性 [36–38]；MInference、DuoAttention、vAttention 与 LServe 则分别从动态稀疏注意力、检索头/流式头分离、虚拟内存式管理以及统一稀疏注意力执行角度改写了长上下文推理的系统组织方式 [50–53]；H2O、KIVI、SnapKV、PyramidInfer 与 CacheGen 等工作则分别从重点击中、低比特量化、语义相关压缩、层次化压缩和缓存流式传输角度缓解了 KV Cache 的显存与带宽压力 [34, 35, 54–56]。这些工作共同说明，国产推理系统若要支持长上下文和高并发，必须同时关注注意力算子、缓存布局与压缩策略的协同设计。

因此，国产推理引擎在 KV Cache 设计上更适合采用“统一抽象 + 平台特化实现”的方式：统一暴露块管理、引用计数、前缀复用和驱逐策略接口，而将具体的数据排布、搬运实现和异步执行细节留给后端适配层处理。从系统演进角度看，KV Cache 还越来越像独立基础设施而非执行层附属模块。随着长上下文、多轮会话、多 Agent 工作流和多模态前缀共存，缓存不再只是服务某一次 decode，而是在更长时间尺度上决定哪些状态可以复用、哪些请求值得迁移、哪些阶段应被解耦执行。也就是说，缓存系统未来不仅要回答“数据放在哪”，还要回答“状态如何被计价、回收、复用和转移”。若缓存层长期停留在局部内存优化的定位上，后续很多关于长上下文和复杂负载的优化都会因状态治理能力不足而难以稳定落地。

3.4 主路径三：压缩、成本与服务质量协同

在国产平台语境下，压缩加速不应再被看作部署后期附加的一组“省显存技巧”，而应被视为与执行路径和状态治理并列的核心主路径。一方面，混合精度量化、KV 动态量化和稀疏–量化协同确实能够直接降低显存占用、带宽压力和单位 token 成本；另一方面，它们的实际收益又高度依赖后端是否存在稳定低比特路径、状态层是否允许容量变化、调度层是否能吸收由量化带来的 batch 形态变化。也就是说，压缩配置的效果从来不是单独由模型精度决定，而是由“压缩参数、执行效率、状态占用、服务质量”四者共同决定。

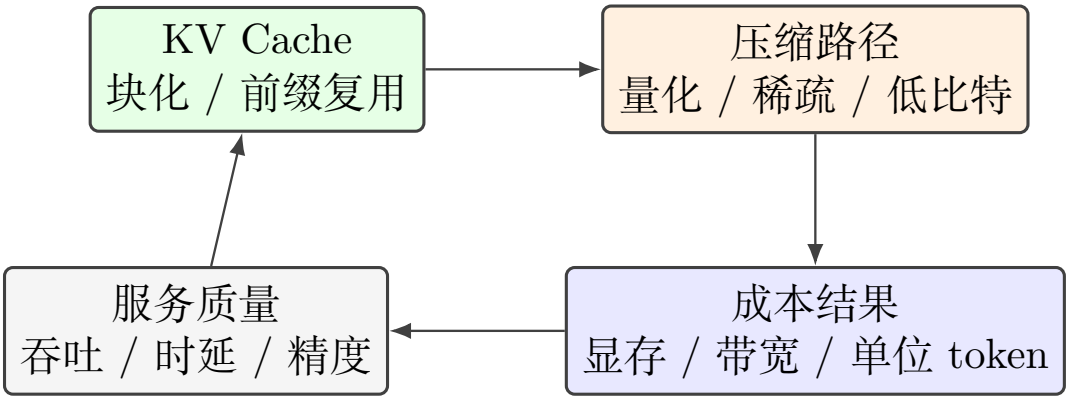


图 5: KV Cache、压缩与成本反馈闭环示意图

如图5 所示，压缩路径的工程价值需要回到 KV 组织、服务质量和单位 token 成本的联动关系中才能被正确评价；脱离这一反馈闭环讨论量化或稀疏化，往往只能得到局部最优。

相关系统已经说明，压缩能力只有进入运行时闭环后才具有工程价值。QServe 通过把权重量化、KV 压缩与 serving 系统协同组织，展示了低比特部署为何必须在 kernel、调度和缓存之间联合设计 [39]；KIVI、SnapKV 等工作则进一步表明，长上下文场景下的 KV 压缩只有和缓存布局、热度判断及恢复路径结合起来，才可能在不显著破坏服务质量的前提下释放收益 [34, 35]。因此，国产推理引擎若要把“低成本”转化为系统能力，就必须让压缩配置、运行时观测与成本核算使用统一接口，而不是把量化与稀疏化停留在模型离线导出环节。在这一前提下，单位 token 成本才会从宣传指标变成可以稳定优化、回归和交付的工程指标。比较国产推理引擎的压缩路线时，也更应关注系统是否建立了可回归的成本与质量反馈机制，而不只是统计系统支持了多少种量化格式或低比特变体。

3.5 服务层收束：调度、接口与交付闭环

服务层需要在吞吐与时延之间平衡，典型机制包括连续批处理、请求优先级控制、SLA 感知调度、流式返回，以及结构化输出或工具调用时的状态管理。对于国产硬件，还需要考虑编译图缓存、静态形状约束与后端初始化成本。也就是说，服务调度不仅要优化“请求何时执行”，还要优化“请求如何映射到平台最擅长的执行形态”。在简单文本生成场景中，调度目标通常是通过更高的 batch 命中率来提升吞吐；而在复杂 Agent 场景中，系统还必须处理函数调用、结构化约束解码、多轮上下文恢复和外部工具返回后的再进入执行。这使得调度器需要持有更丰富的请求状态，同时与采样器、缓存管理器和运行时形成更紧密的协同。XGrammar 说明结构化生成已经不再只是前端语法约束，而是需要与推理引擎共同设计、尽可能把额外开销压缩到接近零的系统问题 [8]；AI Metropolis 则进一步表明，当负载扩展到多 Agent 仿真时，依赖感知的乱序执行会成为提高并行度和硬件利用率的关键机制 [57]。对国产平台而言，如果底层后端对 shape 波动较敏感，那么调度器还需要通过 bucket 化、请求整形和预热机制减少编译抖动。

这一方向上的代表性路线还包括围绕高性能 serving 栈形成的部署体系。以 TensorRT-LLM、DeepSpeed-FastGen、FlashInfer 与 QServe 为代表的项目或系统，分别从图执行、运行时优化、高效算子以及低比特量化协同设计角度推动了推理系统的工程演进 [39, 58–60]。其中不少优化明显带有硬件特征，例如面向 NVIDIA 的 TensorRT-LLM

插件、Hopper 上的 FP8/异步 attention 路线，以及围绕 CUDA core 吞吐瓶颈展开的低比特 serving 设计。进一步看，TensorRT-LLM 已把多模态支持做成相对完整的运行时能力集合，包括独立的 multimodal runtime mode、模型支持矩阵、面向多 GPU 的 “LLM tensor parallel + 视觉编码器复制” 组织方式，以及在 chunked context 下的 embedding table offloading 等机制，这说明多模态部署已不再只是模型转换脚本问题，而是进入了缓存、并行与内存管理共同参与的运行时层 [61]。与此同时，OmniServe 开始尝试把 QServe 的低比特量化路径与 LServe 的长上下文统一稀疏注意力纳入同一套 GPU serving 框架，说明量化、高吞吐与长上下文优化正从孤立技巧走向组合式运行时设计 [62]；而在 AMD 生态中，围绕 MI300X 的工作则更多强调 prefix caching、chunked prefill、speculative decoding 以及 GPU partitioning 下的多实例部署权衡 [63–65]。这些工作虽然不直接针对国产算力，但为本土系统理解热点路径优化、调度协同、量化部署和工程化取舍提供了有价值的对照。

在调度策略方面，研究社区还提出了多种更激进的阶段解耦与服务组织方式。例如 DistServe 和 Splitwise 将 Prefill 与 Decode 的资源需求进一步拆解，以提升系统整体 goodput 或降低阶段间相互干扰 [6, 7]；Llumnix 通过跨实例动态迁移请求及其内存状态缓解异构请求导致的负载失衡，SOLA 则进一步把调度目标显式对齐到 TTFT 和 TPOT 等服务级目标，代表了从 “吞吐优先” 向 “状态感知和 SLO 感知” 调度的演进方向 [66, 67]。这类思路对国产平台尤其重要，因为图编译成本、shape 敏感性和设备初始化路径往往会放大不同阶段之间的资源竞争。因此，国产平台上的调度问题不能只被看作 batch 排队问题，而应被理解为一种 “执行形态控制” 问题。调度器需要决定的不只是谁先执行、谁后执行，还包括请求应被压缩进哪类 bucket、是否值得为某类形状维持图缓存、何时因工具调用或多模态阶段切换而暂时退出主循环、何时为了尾时延而主动放弃局部吞吐。若没有这种更强的执行形态意识，服务层即便短期内能提高平均吞吐，也很容易在真实业务负载下出现图缓存命中不稳、阶段争用严重和尾时延放大的问题。

状态治理继续向生产环境推进后，最终会落到交付闭环。推理引擎的对外层不仅包括 OpenAI 兼容接口、批量离线接口与嵌入式部署接口，也包括监控、日志、链路追踪和故障恢复等运维能力。这一层的重要性在国产推理系统中经常被低估，但在政企和行业部署中，系统往往首先因兼容性、可观测性或升级复杂度而被淘汰，而不是因为理论峰值性能略低。对外接口层的成熟还体现在生态兼容能力上。无论是以 Transformers 为核心的模型与 tokenizer 接口、以 Ray Serve 和 Triton Inference Server 为代表的服务编排与部署接口，还是以 FastChat 为代表的对话服务中间层，都在事实上塑造了现代推理引擎需要兼容的工程边界 [68–71]。因此，一个成熟的国产推理引擎架构应当把性能优化与可运维性一起视作一等能力。前者决定系统上限，后者决定系统能否稳定进入真实生产环境。

综合来看，国产推理引擎的架构难点并不分散在若干孤立模块里，而主要集中在模块交界处：执行层与图编译层决定可运行边界，缓存与调度层决定长上下文和并发表现，对外接口与运维层决定系统能否形成生产闭环。后文在讨论典型技术路线时，重点关注的也正是不同系统如何在这些边界上做取舍，而不是简单罗列其 “支持了哪些功能”。因此，讨论国产推理引擎架构时，更值得比较的不是模块名单本身，而是这些模块之间是否形成了稳定的责任分层。一个系统即便短期内通过大量特判也能跑通模型，如果执行、缓存、调度与接口之间没有清晰边界，后续一旦遇到上游模型改动、平台版本切换或业

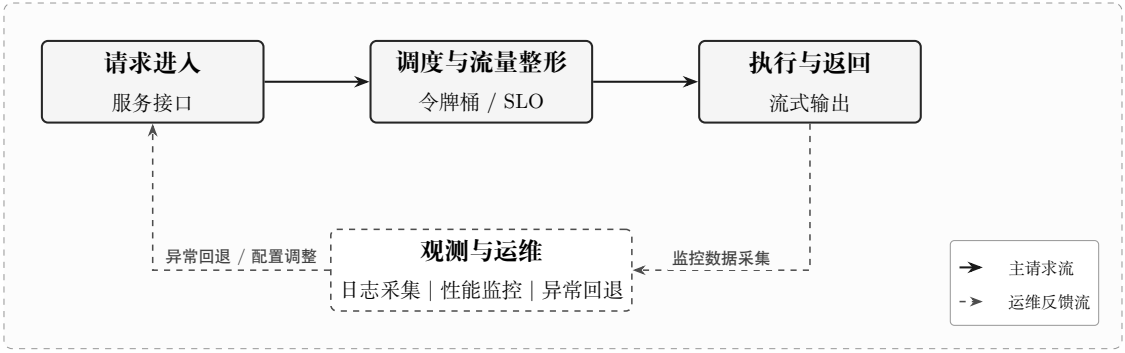


图 6: 调度、服务与运维闭环示意图

务负载变化，维护成本就会迅速放大。相反，那些能够把平台能力、运行时组织和运维闭环收敛进统一分层的系统，即使局部优化未必总是最激进，也往往更有机会沉淀为可持续演进的工程底座。核心架构的优劣，最终不取决于某一层是否拥有最复杂的实现，而取决于系统能否把高变动能力隔离在可治理的边界内。对国产推理引擎而言，平台抽象、运行时状态管理和运维闭环不是三个可分离的附加维度，而是同一套架构是否成熟的不同侧面。不同技术路线的比较最终也会落到同一问题：何种架构组织方式更能在国产平台上同时维持性能、稳定性与可演进性。

4 典型技术路线与复杂度分配

如果沿用前文“一个底座、三条主路径”的分析框架来观察本土路线分化，可以更清楚地看出不同方案真正分开的并不只是产品形态，而是它们把复杂度压在何处：有的路线优先掌握执行与通信主路径，以最快速度把国产后端接入主干框架；有的路线优先把状态治理、部署流程和平台软件栈整体收拢，以换取长期交付确定性；还有的路线试图在统一接口不破坏的前提下，同时经营执行优化、状态治理与压缩成本协同三条主路径。因此，比较不同国产推理引擎时，真正需要问的问题不只是“兼容谁”或“性能上限如何”，而是其是否具备稳定的统一抽象和观测底座，以及它把执行、状态和成本三个关键矛盾分别如何组织进系统中。

路线比较不能只停留在抽象层。真正决定系统成熟度的，是这些路线是否显式处理了统一执行 IR、阶段解耦与通信运行时，是否提供了前缀索引、多级驻留和状态迁移能力，是否把混合精度、KV 压缩和单位 token 成本带入同一优化闭环。只有把这些技术点阐明，才能保证 survey 不会漏掉真正决定国产推理系统上限的技术问题。

围绕复杂度主要落点，现有路线可归纳为三类：先是把复杂度压在执行主链与后端边界上的开源主干适配，再是把复杂度压在平台交付链和模板治理上的一体化方案，最后是把复杂度压在模块边界经营、部署入口和本土能力沉淀上的混合式与本土路线。这一划分使“路线差异”能够直接对应系统复杂度的主要承载位置。

4.1 路线一：把复杂度压在执行主链的开源后端适配

以 vLLM 与 SGLang 为代表的开源推理框架，正在通过平台接口、后端插件、模型执行器扩展和定制算子等方式适配国产硬件 [2, 3, 72, 73]。该路线的优势，在于可以持续复用上游社区在调度、缓存、结构化输出与多模态支持方面的能力演进，并借助社区已有的

模型生态快速覆盖主流开源模型。它的核心思想不是重写一个本土专用引擎，而是在保留上游主干能力的同时，为国产后端补齐执行和运维能力。

典型实现方式包括：在平台层增加设备选择与能力注册；在 worker 或 model runner 层注入后端特化逻辑；在注意力、归一化、采样等热点路径增加定制算子；以及在调度与缓存层引入面向本土平台的约束门控。其好处是迭代快、模型覆盖广、与上游社区距离较近；不足则在于，若抽象边界设计不当，后端 patch 会随着上游快速演进不断累积技术债。在国内部署实践中，这一路线的真实工作量通常还不止于“补齐 kernel 或后端接口”，还包括模型接入、量化转换、OpenAI 兼容协议、监控埋点和部署脚本等外围工程。后端适配是否成功，也往往取决于模型支持矩阵更新速度、异常回退路径是否可观测，以及版本升级时能否继续跟上上游抽象变化。LMDeploy 更强调部署工具链与模型接入效率，vLLM-Ascend 更能体现对上游执行边界的依赖，而 MindIE 一类方案则把这些问题更多吸收到平台交付链路内部 [73–75]。因此，本土团队在选择“开源主干后端适配”时，实际评估的往往不是单一性能收益，而是模型跟进速度、改动侵入度和后续运维成本之间的综合平衡。

若进一步看国内 fork 的工程组织方式，vLLM-HUST 更适合被视作介于“开源主干后端适配”和“混合式架构”之间、但整体更偏混合式的一类实践：它在上层继续保持 vLLM 兼容的服务接口、多模态、Prefix Cache 与 Multi-LoRA 等能力边界，在下层则通过平台插件、Ascend 运行时预检和外部运行时治理工具，把国产平台接入与环境治理从核心执行链中拆开 [76]。这意味着它并不是单纯在主干里补一个后端，而是在尽量复用上游 serving 主体的前提下，把平台特化与启动护栏做成相对独立、可演进的工程层。这类实践的重要性在于，它揭示了国产项目在“复用上游”之外的另一层真实工作：不是简单选择跟或不跟上游，而是重新设计哪些能力必须留在主干、哪些能力应沉淀为插件、哪些治理逻辑需要放在运行时外围工具链中。如果说 vLLM-Ascend 更像把国产平台能力直接压入主干兼容边界，那么 vLLM-HUST 则更强调在主干周围补齐平台护栏、能力注册和环境治理。这种差异说明，开源主干后端适配并不是单一形态，它内部也在分化为“主链后端实现优先”和“主链复用 + 外围治理增强”两种组织方向。

更具体地看，vLLM-Ascend 与 vLLM-MLU 应被视为同一技术族：二者都尽量继承 vLLM 的 OpenAI 兼容接口、调度语义和缓存抽象，再分别把 Ascend NPU 与寒武纪 MLU 的执行、依赖和分布式适配压入 plugin backend 边界 [73, 77]。SGLang Ascend 则不是另一个独立国产引擎，而是 SGLang 主线仓库中的 Ascend 平台后端；其意义在于 RadixAttention、PD 分离、structured outputs、chunked prefill 等主流 serving 机制正在被映射到国产 NPU 后端，而不是在本土分支里孤立重做 [3, 78]。这一路线的证据边界也应写清：插件式后端的成熟度受 CANN、torch_npu、厂商 SDK、容器镜像和上游版本节奏共同约束；SGLang Ascend 中 NPUGraph、HiCache/MoonCake、量化和投机解码等更高阶能力仍有不少属于 roadmap 或持续工程化阶段，不能把“主线纳入”直接等同于“所有高级特性已经稳定落地”[79]。

如果进一步参考这类项目的源码架构分析，可以把“主链复用 + 外围治理增强”的混合式路线抽象为更一般的组件分层：入口与启动护栏负责 CLI/OpenAI 服务和运行时预检，配置装配层负责把参数收敛为统一配置对象，引擎与 EngineCore 负责请求编排和调度通信，模型执行层负责 registry、worker、runner 与热点算子，平台与插件层负责能力探测和后端激活，多模态/Reasoning/Tool 层负责复杂能力的横切扩展，编译、KV

Cache、分布式与 tracing 则构成性能基础设施 [76]。这一结构表明，混合式架构的工程含义不在于笼统地同时追求兼容与优化，而在于通过分层把不同类型的国产化改动压回不同边界，使执行热路径优化、平台接入、环境治理和复杂能力扩展不必全部挤在同一条共享链路上。

从国产算力优化视角看，这种组件分层还有一个额外价值，即它能帮助识别“哪些组件最值得优先做本土化优化”。相比只给出系统级性能结论，表4 更细地展示了混合式开源引擎各组件与国产算力潜在优化点之间的对应关系。这个视角对综述有意义，因为它把“混合式架构”的讨论从路线口号推进到了可操作的工程层：哪些部分更适合做 graph/compile 优化，哪些部分更适合做插件化平台适配，哪些部分更适合承载多模态与工具调用状态机，哪些部分则应该承担运行时护栏与环境治理。vLLM-HUST 在这里更适合被理解为这种组织方式的一个实例，而不是本节唯一需要关注的对象。

这一类生态已不再只有少数头部系统，更值得关注的是开源 serving 生态正在沿几条相对清晰的机制路径分化。LightLLM、Text Generation Inference 与 MLC-LLM 更强调面向在线服务的高吞吐批处理、跨设备部署和统一服务接口，代表的是“优先优化服务主路径”的路线；Ollama、llama.cpp、ExLlamaV2 与 mistral.rs 更突出轻量化部署、本地运行和低比特执行，代表的是“优先保障受限资源下部署稳定性”的路线；Aphrodite、KTransformers 与 Lorax 则更关注运行时扩展、多租户适配和可定制 serving，代表的是“把推理引擎构造成可工程化组合平台”的路线 [80–89]。这种分化对国产算力尤其重要，因为它提示我们：所谓“兼容开源生态”，并不是简单接入某个框架，而是要明确自己究竟优先继承哪一类能力主线，是吞吐优先、资源受限部署优先，还是多租户和平台化能力优先。

进一步看，开源主干框架在不同硬件生态上的演进方式也并不相同。NVIDIA 侧往往更快沉淀出围绕 Hopper、TensorRT-LLM、FlashAttention-3、FlexAttention 和 FlashInfer 的 kernel、编译优化和多模态运行时组合，其特点是把硬件代际能力迅速转化为 fused kernel 与 runtime 特性；AMD 侧当前更常见的则是围绕 MI300X、ROCm、vLLM 以及 SPX/DPX 分区模式组织多实例 serving、prefix caching 和前缀/解码阶段优化，其重点更偏向在现有框架边界内重组部署形态和资源切分 [38, 49, 58, 60, 63–65]。它们虽然未必都直接面向国产算力，但共同构成了当前推理引擎生态的参照系，使国产后端适配必须同时回答“如何兼容主流接口”和“如何把本土平台特征转化为具体机制收益”这两个问题，而不只是把系统名字并列在一起。

4.2 路线二：把复杂度压在平台交付链的一体化方案

产业部署常采用“推理引擎 + 编译器 + 运行时 + 集群管理”一体化方案，以获得更强的可控性与稳定性。这类方案通常与特定芯片、编译链和部署平台深度绑定，能够对图编译、算子融合、通信调度和资源管理做更强约束，因此在某些固定模型与固定业务场景下往往具备更好的性能上限。但这种路线同样存在明显代价。首先，一体化系统往往需要为不同模型族、不同生成模式和不同上下文长度维护多组优化配置，其工程复杂度并不低。其次，这类平台若要兼容社区新模型或新服务特性，如工具调用、结构化输出和多 LoRA，往往需要较长的产品交付周期。因此，一体化方案更适合追求交付稳定性与平台统一性的组织，而开源后端适配路线更适合希望快速跟进模型生态和上游能力迭代的团队。

在国产算力语境下，这一路线通常与本地基础软件栈深度耦合。MindSpore、CANN 与 MindIE 共同构成了较典型的国产软硬一体化推理路径，DeepSeek-V2 与 DeepSeek-V3 技术报告披露的 MLA、MoE 和大规模服务经验也表明，后端通信、缓存组织、注意力内核与平台运行时的一体化设计往往直接决定了系统可扩展上限 [44, 45, 75, 90, 91]。与此相呼应，FlashMLA 展示了模型架构和底层注意力实现一体优化的工程方向，Mooncake 及其面向 KVCache 的存储解耦实践，则进一步说明产业界正在把缓存系统本身视作独立基础设施进行建设 [92–94]。从横向比较看，无论是 NVIDIA 上围绕 Hopper 代际能力展开的 attention/kernel 深度优化，还是 AMD 围绕 MI300X 分区与 ROCm/vLLM 的部署工程实践，都说明一体化方案的性能上限往往来自对“芯片特征 + 运行时 + 服务策略”的联动打磨，而不是单点替换某个 kernel[38, 63–65]。

这解释了为什么一体化方案在产业部署中往往具有更高的组织门槛。它要求团队不仅理解模型结构和推理算法，还要同时掌握编译器约束、运行时资源管理、设备通信语义以及部署平台的交付流程。对外表现上，这类系统常常提供的是一个“稳定可交付的平台能力包”，而非一个单独的 serving 框架；对内则意味着模型上线、版本回滚、性能调优与故障定位都被纳入同一套平台流程中。因此，一体化路线的优势通常体现在平台统一性、固定场景确定性和长期运维可控性上，而不是对所有新模型和新特性的最快响应。从产业决策角度看，一体化方案的吸引力也不只在于跑分更高，而在于能把很多本应由业务团队承担的系统复杂度前移到平台方内部。这类方案提供的，不仅是推理性能，还包括版本矩阵收敛、交付责任集中和问题定位路径清晰等“组织确定性”。这也是一体化方案在政企和行业部署中经常具有较强吸引力的原因：在这些场景里，系统是否能被稳定运维、统一回归和持续交付，往往比单轮模型跟进速度更关键。相应地，它的代价也不只体现在生态兼容慢，更体现在系统创新往往需要通过平台产品节奏释放，外部团队很难像使用开源主干那样快速做局部替换和组合式实验。

4.3 路线三：把复杂度压在边界经营的混合式与本土路线

与上述平台一体化方案相邻但不完全相同的，是 Chitu 与 xLLM 代表的国产原生/独立引擎路线。Chitu 的公开仓库表明，其重点不是给 vLLM 增加一个后端，而是围绕独立推理引擎组织多元国产算力适配、阶段解耦、低精度路径与集群部署能力 [95]。xLLM 则更突出 service-engine decoupled 架构：服务层承担在线/离线统一调度、动态 PD、Encode/Prefill/Decode 分离（EPD disaggregation）、多级 KV Cache 与容错治理，引擎层承担多流并行、图优化、显存管理、投机解码和 MoE 负载均衡；其公开硬件矩阵覆盖 NPU、MLU、ILU 与 MUSA 等多类国产或国产生态加速器，但模型覆盖和各硬件 feature 完整度并不均衡 [96–98]。这一路线对综述的价值在于说明，国产推理系统并非只能沿“主流框架加后端”演进，也可以从服务状态机、缓存治理、版本矩阵和部署闭环出发，自上而下设计一套独立系统边界。

国产算力场景下，推理引擎的差异不再仅体现在单点算子性能，还体现在是否支持长上下文、Prefix Cache、Chunked Prefill、多 LoRA、多模态编码链路、结构化输出和工具调用等高层能力，以及这些能力能否在实际服务中稳定运行。特别是在真实服务中，单项能力“可用”与该能力“可规模化、可观测、可维护”之间存在显著差异。综合来看，本土生态中的技术分化主要体现在四个维度：其一，系统是优先追求通用框架兼容还是优先追求特定平台上的极致性能；其二，系统能否以较低代价跟进上游模型和推理

特性的演进；其三，系统是否具备面向生产环境的灰度升级、故障回退与指标观测能力；其四，系统是否能够支持多样化工作负载，而不仅是固定 benchmark。这四个维度之所以重要，是因为它们对应了国产推理引擎最核心的四类成本：生态迁移成本、平台适配成本、交付运维成本和复杂负载成本。很多路线在单一维度上都能取得较好结果，但真正困难的是在多个维度上同时不过度失衡。例如，极端强调生态兼容的方案往往需要在平台深度优化上做妥协，极端强调单平台性能的方案又可能显著抬高后续模型迁移成本；而那些能力边界持续扩张的系统，若不能把这些支持组织成稳定抽象，最终仍会把成本转移为升级困难和维护负担。因此，国产推理引擎的比较不能只看功能清单，而要看一条路线究竟把成本压在了哪里、又把复杂度转嫁给了谁。

根据当前已经能够坐实的公开主体和产品形态，本土路线更适合按“复杂度主要落点”而不是按项目名划分：插件式后端适配把复杂度压在 backend 与上游接口边界，平台一体化把复杂度收进 SDK/runtime/compiler 与交付模板，国产独立引擎把复杂度前移到 service state machine、KV Cache 与部署闭环，工具链型部署把复杂度组织到模型接入、服务接口和硬件安装矩阵，跨架构编译/运行时探索则试图把重复适配成本压进 compiler/runtime abstraction[73, 75, 77, 78, 95, 98–101]。这种分层能够避免把“框架本体”“交付平台”“部署工具链”和“编译运行时抽象”混成同一层对象，从而误判各路线真正承担的复杂度类型。

还需要再向前分清一层：公开搜索里经常会同时出现工作站、一体机、云平台、训推平台和应用交付页面，但这些对象大多是在封装、承载或预集成上述引擎路线，而不是与推理引擎本体处于同一比较层级。部署载体回答的是“能力通过什么形态被交付出去”，推理引擎回答的则是“请求、状态、缓存、调度和算子如何被组织起来”。如果把两者直接并列，结论就会从路线比较滑向产品罗列，既不利于解释系统复杂度真正落在哪一层，也容易把工程入口便利性误写成引擎内核先进性。

表5 的意义不在于给出一个“单位名录”，而在于把国产算力推理引擎的差异还原为复杂度分配问题。若复杂度主要落在 backend，路线优势通常是更快复用上游生态；若复杂度主要落在 runtime 或平台交付链路，优势是固定平台上的确定性更强；若复杂度主要落在 service state machine 和 KV Cache 治理，则更容易支撑动态 PD/EPD、长上下文和多硬件资源池；若复杂度被压入 toolchain 或 compiler/runtime abstraction，则更有机会缓解多芯片重复适配与部署入口碎片化。由此可见，面向国产算力的推理引擎竞争并不是单一项目之间的横向竞赛，而是“谁掌握抽象边界、谁承担适配成本、谁负责交付闭环”的系统分工问题。

公开社区讨论虽然不能替代系统论文、官方文档和源码分析，但对理解本土路线为何分化仍有补充价值，因为它更直接暴露了复杂度首先溢出的边界。综合公开工程入口与部署要求可以看到，插件式后端适配的摩擦更容易集中在上游主干、插件分支、CANN/torch_npu、厂商 SDK 与容器镜像的协同；平台一体化路线的摩擦更容易集中在服务模板、监控、配置基线与回滚路径；独立引擎和工具链路线则更容易暴露服务状态机、KV Cache、模型支持矩阵和硬件安装入口之间的一致性问题 [97, 99]。这些材料的综述价值不是给出新的项目清单，而是说明国产平台的一线痛点往往是“先把系统稳定组织起来”，之后才谈局部极致优化。

工具链型路线进一步说明，国产算力适配不只发生在单一 runtime 的后端层。FastDeploy 更接近飞桨生态中的多硬件推理部署工具链：它把 OpenAI/vLLM 风格服务接

口、PD 分离、KV Cache 传输、量化、投机解码和 Chunked Prefill 组织在同一工程体系中，并面向昆仑芯、海光、Ascend、天数、燧原、沐曦等平台提供公开适配入口 [99, 103]。LMDeploy 则需要明确区分 TurboMind 与 PyTorchEngine：前者主要服务 CUDA 高性能路径，国产平台扩展主要通过 PyTorchEngine/Other Platforms 文档进入 Ascend、Cambricon 和 MACA 等后端，并且模型、设备与 eager/graph/量化模式的支持粒度并不对称 [74, 100, 102]。因此，二者都不宜被简单写成“又一个推理引擎”，而更适合作为“部署工具链 + 服务接口 + 硬件适配层”的工程体系样本。

最后还应保留一个次级但有趋势意义的生态位置：LightLLM 更适合作为国内团队在高吞吐 serving、TokenAttention、Efficient Router、调度与内核生态上的重要补充，而不是“成熟国产多硬件主线引擎”；其公开文档中最明确的国产硬件线索主要是 MUSA 路径，默认主线仍明显偏 CUDA/Triton[80, 104, 105]。与此不同，中国电信 Triton 统一跨架构推理框架并不处于 vLLM、LMDeploy 或 Chitu 同一层，而更像跨架构编译/运行时抽象：官方材料强调自研 Triton 跨架构编译器、统一算子库、vLLM-Triton 透明嵌入插件和图算融合编译器，用于缓解多国产芯片重复适配问题；但由于代码、许可证、版本矩阵和复现脚本尚未公开，相关迁移时间与性能损耗数字只能作为官方技术验证口径谨慎引用 [101]。

近年的公开部署记录与官方入口反复暴露三件事：其一，vLLM-Ascend 的工程可行性高度依赖“上游主干版本 + 插件分支 + CANN/torch_npu”严格对齐 [73]；其二，MindIE 更接近把镜像、服务化组件、监控和配置模板一起交付的平台化方案 [75]；其三，PD 分离、多机 RoCE/HCCL 连通性、环境预检和日志/配置模板频繁成为首轮排障对象。公开材料反复暴露的，不是某条路线的单点性能差异，而是部署不确定性由谁承担、以什么方式被吸收。

将这些社区信号收敛为选型判断，比直接比较“谁更快”更有实用性。若团队已有 vLLM/SGLang 服务栈，插件式后端适配的关键是能否承受版本协同与环境治理成本；若组织更重视统一交付，平台一体化路线的关键是模板、监控和回滚路径是否可控；若目标是围绕国产多硬件资源池形成自研能力，独立引擎更需要证明服务状态机、缓存治理和模型矩阵能够持续演进；若目标是快速工程落地，工具链型路线则应重点核验不同硬件上的模型、量化、graph/eager 与服务接口支持粒度 [73, 75, 78, 95, 96, 99, 102]。因此，路线选择本质上不是“项目偏好”，而是决定复杂度首先由 backend、runtime、service state machine 还是 toolchain 承担。

表6 给出不同路线更适用的前提、工程摩擦与系统收益；第 5 节的表10 则进一步对应这些路线首先会在哪些系统问题上付出代价。两表合看，国产推理引擎路线选择本质上仍是复杂度分配问题：有的路线把复杂度压在版本矩阵和平台适配上，有的路线压在平台交付与模板治理上，有的路线压在复杂控制流和运行时状态空间上，还有的路线压在模块边界与长期 merge-safe 演进上。

围绕多租户与定制化模型服务的能力也在持续拉开系统差距。Punica 与 S-LoRA 表明，当 LoRA 适配器数量迅速增长时，系统需要在缓存复用、批内共享和运行时隔离之间重新平衡 [106, 107]。与此同时，Guidance、Outlines 与 XGrammar 等工具说明，结构化输出与受约束生成正在把“推理引擎”从纯粹的 token 生成器转变为可编排的执行系统；而 XGrammar 的论文实现则进一步表明，这种能力已经需要以语法执行引擎、持久化栈和 GPU 重叠执行等系统机制来支撑 [8, 108–110]。

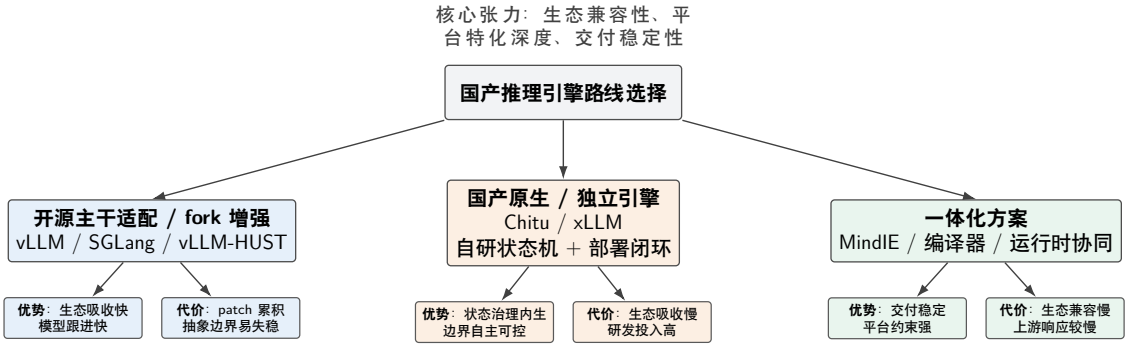


图 7: 国产推理引擎技术路线分化示意图

调度能力同样正在成为新的分化点，但这里更重要的也不是系统名本身，而是它们分别把哪一类调度变量显式化了。Llumnix 把跨实例动态迁移和内存状态搬运带入运行时，使“负载均衡”不再只是请求排队问题；SOLA 把 TTFT、TPOT 等服务级目标直接写进调度目标函数，使“调度最优”从吞吐优先转向 SLO 优先；AI Metropolis 则进一步把多 Agent 依赖关系显式化，通过乱序执行削减伪串行等待 [57, 66, 67]。这一组工作共同说明，现代推理系统的竞争焦点已经从“是否支持连续批处理”进一步演进到“是否能够在复杂状态空间里稳定组织执行”，而这正是国产推理引擎在复杂业务负载下最不能回避的技术点。

图7 将主要路线放到同一张图里比较，其重点不在功能罗列，而在它们对“生态兼容性、平台特化深度、交付稳定性”三者的不同取舍 [76]。从课题化建设视角看，这种分化本质上也是对执行与通信、状态治理和压缩协同三条主路径控制权的不同分配，因此更能解释为什么一些系统在 benchmark 上接近，却在长期维护和场景扩展上差异巨大。

从现实可行性看，混合式路线更符合当前阶段，因为它试图同时保留上游生态复用与本土关键路径特化。但它成立的前提，是上层生态节奏与底层平台节奏之间存在清晰边界；否则系统很容易演化为既继承上游复杂度、又承受本地特化负担的双重分叉。

需要指出的是，这些路线判断并非静态结论，而是会随着上游版本迭代、国产后端接入深度和具体部署目标持续变化。综述层面更值得关注的是一种判断框架：如果目标是快速吸收社区能力，应优先选择抽象边界清晰、插件接口完善的路线；如果目标是固定平台上的确定性交付，则更应重视图编译、运行时和运维体系的一体化程度。基于这一判断，再看后面的案例表就更有意义，因为它不再重复“哪类路线更强”，而是说明不同代表对象把工程投入压在何处。

多模态推理正在把引擎边界从文本 decode 推向异构阶段编排。就国内团队的工程案例看，Qwen2-VL 与 InternVL2 代表了多图与文档理解链路进入在线服务后的典型压力点：系统不仅要处理动态分辨率切图和图像 token 打包，还要面对视觉前缀复用不足、OCR 密集区域导致的 prefill 膨胀，以及图文混排输入带来的 shape 波动 [46, 47, 112, 113]。CogVLM2 则把视频理解显式纳入主流模型服务，使推理链路必须同时考虑帧采样、跨帧压缩、视觉 encoder 批处理以及 decode 侧连续 batching 之间的资源再分配 [114, 115]。在语音方向，Qwen2-Audio、GLM-4-Voice 与 MiniCPM-o 进一步把音频前端、speech tokenizer、流式 chunk 累积以及语音/文本交错生成纳入同一运行时，使首 token 时延、实时率和打断恢复直接成为推理引擎的系统约束 [48, 116, 117]。

如果从运行时组织角度进一步拆解，这些压力并非简单叠加在文本 serving 主路径

之上，而是在三个层次上重写引擎设计。第一，多图、文档与 OCR 负载首先重写的是 prefill 主路径：视觉 token 数量膨胀、切图策略离散化以及图文混排导致的长度抖动，会直接改变 bucket 划分、图模式命中率与前缀缓存复用策略。第二，视频负载重写的是阶段关系本身：帧采样、视觉 encoder 和语言 decode 不再是线性串接关系，而是必须围绕显存峰值、带宽占用和首 token 时延做跨阶段再平衡。第三，语音交互重写的是会话状态机：streaming chunk、实时打断、语音文本交错生成以及会话恢复，使系统不再只管理 KV Cache，还要同时管理音频前端状态、增量转写结果和流式输出节奏。多模态路径真正扩张的，不是“模型输入类型”，而是引擎内部必须长期维护的状态对象种类。

这也是为什么多模态引擎往往不能通过“在文本框架前面补一层预处理”来完成工程闭环。对于多图文档理解，系统需要把图像切块、版面解析和视觉前缀编码结果纳入可缓存对象，否则同一文档的重复查询难以获得稳定复用收益；对于视频理解，系统需要显式决定哪些阶段可以批内共享、哪些中间结果应跨帧复用、哪些请求必须在 encode 与 decode 之间做资源隔离；对于语音对话，则要把实时率、打断恢复和声文混合生成直接写进调度与回退逻辑。正因为这些约束同时涉及前处理、缓存、调度、图执行和服务协议，多模态推理才会成为检验一条引擎路线是否成熟的分水岭：如果系统只能做到单模型单链路跑通，而不能把这些异构阶段收敛为统一状态机，那么模型支持矩阵越扩张，运行时复杂度就越难被控制。

ElasticMM、EPDServe 与 Eevee 分别从弹性并行、阶段解耦和模块复用三条路径回答这些问题：前者统一管理多模态请求与前缀缓存，后两者分别通过 encode/prefill/decode 拆分和 module multiplexing 缓解阶段异质性与模块争用 [9, 118, 119]。它们共同说明，多模态系统的难点已不是“支持某个模型”，而是能否在请求异构、阶段异质和模块干扰并存时保持资源利用率与首 token 时延稳定。对国产推理引擎而言，这要求把前处理、跨模态缓存、阶段解耦和异构编码链路一起纳入运行时设计。

这些工作对国产路线的参考价值，不在于提供了可以直接照搬的实现，而在于它们把多模态运行时中最容易被忽略的几个控制变量显式化了。ElasticMM 强调多模态请求不应只在入口处分类，而应在运行时持续维护视觉前缀、缓存命中和并行度之间的关系；EPDServe 说明 encode、prefill 与 decode 一旦继续共用同一套排队和资源分配策略，视频和文档类请求就会系统性挤压文本 decode 的服务质量；Eevee 则进一步表明，模块复用并不是单纯节省显存，而是在异构阶段交替出现时减少重复加载与执行抖动的一种运行时组织方法 [9, 118, 119]。对国产推理引擎而言，这意味着多模态工程的关键不只是补齐若干模型适配器，而是把视觉 encoder、语音前端、结构化输出、tool/reasoning parser 与平台后端能力一起压进统一的生命周期管理中。只有当这些对象都能够被调度器、缓存管理器和异常回退链共同感知，本土系统才可能在多模态场景下同时维持吞吐、时延和可维护性。

以 vLLM-HUST 这类本土 fork 为例，其工程重点已经不再只是补一层国产平台接入，而是把多模态注册中心、reasoning parser、tool parser、OpenAI 兼容服务与平台插件链一起组织成横切能力层，使模型接入、工具调用和复杂输出协议能够在同一 serving 主体内持续演进 [76]。这类做法说明，国产推理引擎的“可持续性”越来越依赖能力边界是否模块化，而不只是某一条 kernel 路径是否被局部打磨到极致。

从本土生态的工程走向看，一个越来越清晰的趋势是：模型能力边界正在反向塑造推理引擎边界。无论是 LMDeploy 围绕部署工具链与模型支持矩阵的持续扩展，还是

MindIE 一类一体化系统对平台统一性交付的强调，抑或 vLLM-Ascend 这类后端适配项目对上游抽象边界的依赖，其共同约束都不再只是 “文本生成是否足够快”，而是多模态、结构化输出、长上下文和复杂工作负载能否被纳入同一套可维护的运行时框架 [73–75]。

当前国产推理引擎的比较维度，正在从 “谁支持更多模型” 转向 “谁能以更低工程代价吸收模型变化”。当上游模型不断引入 MoE、长上下文、多模态 encoder 和语音交互等新结构时，真正稀缺的能力不再是一次性的端到端适配，而是能否把这些变化收敛成可复用的调度、缓存、算子与接口演进机制。

据此，可以形成一组更具操作性的判别准则：若团队首要目标是快速跟进模型与服务特性，应优先选择上游距离近、插件接口清晰且共享路径修改受控的方案；若目标是面向固定业务长期稳定交付，则应更重视一体化平台对编译、运行时和回归流程的整体收敛能力；若团队既要维持开放生态接口，又要针对国产平台逐步形成本地优化和治理能力，那么混合式架构往往更值得投入，但前提仍是持续经营模块边界，而不是依赖经验性补丁维持兼容。

5 关键挑战：主路径如何相互放大

综合前文，国产推理引擎面临的难点并不只是单点性能不足，而是多个系统目标同时受限：既要持续吸收上游能力演进，又要适应本土硬件和基础软件栈的边界，还要在真实业务负载下维持稳定交付。这些挑战之所以顽固，不是因为每一项都新，而是因为它们会相互放大。平台差异会抬高抽象成本，抽象不稳又会削弱复杂任务支持；复杂任务支持一旦采用侵入式补丁实现，又会加速本地分叉累积；而如果评测仍主要停留在理想化 benchmark，团队就很难及时识别这些结构性问题。表9 将这种放大关系先压缩成一张总览表。

因此，本节不能只给出一张风险清单，而必须顺着前文骨架去看这些挑战如何沿主路径扩散：统一执行抽象、通信运行时和阶段解耦机制一旦缺位，系统通常会先在吞吐与时延主路径上失稳；前缀索引、多级驻留和状态迁移能力一旦不足，问题通常会先在长上下文和高并发条件下显现；量化、KV 压缩和成本核算一旦没有进入同一闭环，压缩收益就往往只存在于离线实验而无法沉淀为真实生产能力。下面按主路径和底座边界展开。

将公开社区讨论纳入观察可以看到，一线团队高频暴露的问题与本文挑战分解高度同构。公开部署记录与工程经验总结反复聚焦于版本矩阵对齐、PD 分离与双机组网稳定性、MindIE 服务化配置可控性，以及 LMDeploy 与 vLLM 在国产 GPU/信创环境中的适配取舍。此类讨论虽不等价于可复现实验，却足以说明真实摩擦往往先出现在环境治理、服务模板和平台工具链，而不是先出现在单点跑分上。

新一轮整理到的公开部署材料进一步说明，真实选型往往围绕组织能力边界而不是理论峰值展开。以 Ascend 相关部署记录为例，镜像权限、宿主机设备映射、环境脚本、张量并行参数与 pytorch_npu 检查常被写成一整套前置步骤，这说明在信创或一体机场景里，“先稳定启动起来” 本身就是核心工程门槛。公开讨论反复暴露的摩擦，最终都能回收到几类更稳定的系统挑战上：版本矩阵与插件对齐问题，本质上在放大平台抽象和共享路径分叉风险；PD 分离、双机组网和连通性问题，本质上在放大调度、状态迁移与网络控制面的耦合；模板与配置敏感性，则说明一体化路线的真正门槛并不只在执行性能，而在服务层参数、监控与回退路径是否足够可控。

如果把前文的路线比较与本节挑战分解合在一起看，还可以得到一个更直接的判断：不同路线并不是面对同一组问题再做不同优化，而是会把“版本矩阵、网络控制面、缓存/调度、运维回退”这些挑战优先暴露在不同边界上。路线选择本身，也就在决定团队首先要把系统能力压在哪一层。表10 将这一点收成更直接的工程判断。

与第 3 节表6 对照时，关键结论不是“哪条路线天然更优”，而是“不同路线把复杂度分配到不同系统边界”：vLLM-Ascend 更早暴露版本矩阵与分布式连通性，MindIE 更早暴露模板治理与平台回滚，LMDeploy 更早暴露跨平台基线与量化回退，SGLang 更早暴露复杂控制流与执行状态组织，混合式路线更早暴露模块边界经营与长期演进秩序。第 4 节真正要说明的，也正是这种“挑战先在哪一层爆出来”的差别。

5.1 执行与通信主路径：硬件异构与后端碎片化

不同国产加速平台在编程接口、图模式约束、算子能力与通信栈成熟度上差异明显，导致共享推理路径中容易出现大量平台特化分支。若缺乏清晰的后端边界，系统会迅速失去可维护性。表面上看，这只是“多写几段条件分支”的问题，实际上它会逐步侵蚀调度、缓存、采样乃至接口层的独立演进能力。更棘手的是，硬件异构通常并不只表现为性能差异，还表现为正确性与稳定性边界不同。例如，某些后端对动态 shape 更敏感，某些后端对特定融合算子支持不完备，某些平台则在通信或异步执行语义上与主流实现存在差异。一旦这些差异没有被收敛到统一的后端抽象中，系统开发者就会被迫在共享路径上叠加越来越多的特判逻辑，使代码难以测试、难以合并，也难以为新模型复用。难点不在“是否存在统一抽象”，而在抽象该把什么纳入、把什么留在边界外。若抽象过薄，平台差异会反复溢出到 worker、调度器、采样器和 API 层；若抽象过厚，又容易把短期平台特例固化成长期接口负担。较健康的工程策略，通常不是试图用单一层吞掉所有差异，而是把平台能力差异、启动期环境治理和运行期异常护栏分层处理：平台层负责设备能力与后端选择，运行时层负责降级、回退和状态一致性，环境治理层负责依赖、驱动和工具链的前置修复。这种分层更重要的价值，在于能把不同类型的问题压回各自的责任边界，使系统在面对新硬件或新模型时不至于整条执行链一起失稳。

这也是为什么许多通用部署项目即便能够在共享生态中快速演进，一旦迁移到新的硬件后端，仍然需要经历较长的抽象重构周期。Text Generation Inference、MLC-LLM、llama.cpp、ExLlamaV2 与 Triton Inference Server 等系统对运行时、设备接口和算子组织做了不同取舍 [70, 81, 82, 84, 85]；而 vAttention、QServe 与 FlashMLA 这类较新的工作又进一步表明，一旦目标转向长上下文、高吞吐或低比特量化，系统边界和底层内核实现会比传统框架式适配更快成为瓶颈 [39, 52, 92]。因此，国产后端若要在这些生态上实现高质量适配，往往首先要面对的不是单点算子问题，而是系统边界和接口契约问题。

更具体地说，国产算力适配的工程挑战首先表现为版本矩阵而非单一后端接口问题：CANN、torch_npu、厂商 SDK、Ray、容器镜像、分布式通信库与上游框架版本之间往往存在强耦合，vLLM-Ascend、vLLM-MLU 与 SGLang Ascend 的公开材料都体现出这种依赖 [73, 77, 78]。其次，不同硬件上的 feature parity 很难默认成立，graph mode、attention backend、量化、PD 分离、KV Cache 传输和 speculative decoding 等能力即使在同一工具链中也可能存在支持粒度差异，FastDeploy 与 LMDeploy 的多硬件文档都需要按设备和执行模式分别核验 [99, 102]。再次，公开性能数字不能直接横向比较，因为模型版本、输入/输出长度、并发度、精度、batch 组织、硬件代际和 runtime 版本的任一变化都可

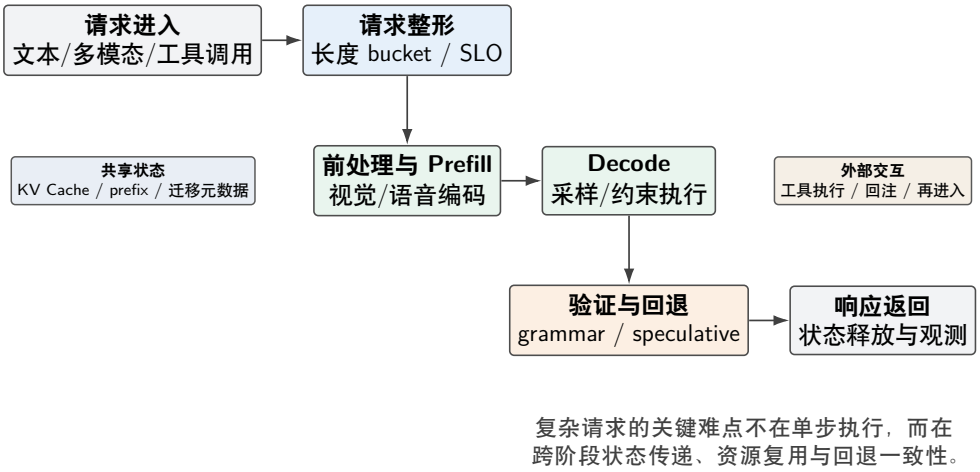


图 8: 复杂请求生命周期示意图

能改变结论；因此 roadmap、官方宣传、技术验证和开源可复现能力应分层表述，尤其不能把 SGLang Ascend 的 roadmap 能力或中国电信 Triton 的技术验证口径写成已经完全成熟、可复现的开源引擎能力 [79, 101]。

5.2 状态治理主路径：复杂任务场景的能力补齐

当前真实负载已从单轮文本生成扩展到长上下文、多模态理解、结构化输出、推理链、工具调用和多 Agent 协同 [8, 9, 57]。相应地，推理引擎的优化目标也不再是单一路径 decode，而是跨阶段状态管理：结构化输出要求把约束执行压进采样环路，工具调用要求支持生成、外部执行与回注之间的往返切换，多模态请求要求消化更长的前处理链和更剧烈的 shape 波动，多 Agent 负载则要求识别真实依赖并消除伪串行。对国产平台，难点不在接口名义上“可用”，而在这些能力会同时冲击调度、缓存和图模式。以国内多模态链路为例，多图与文档请求受视觉 token 膨胀和分辨率离散化约束，视频请求会放大 encode 与 decode 的资源竞争，语音对话则引入 streaming chunk、语音 token 和实时打断恢复；若运行时仍按文本模型的单阶段假设组织执行，就很容易出现局部功能可用而端到端时延和稳定性失控的情况 [46–48, 115–117]。更关键的是，复杂任务场景直接改写了“状态”的性质。文本 serving 时代，系统重点管理的是请求队列、KV Cache 和 batch 形状；而进入多模态、工具调用和 reasoning 场景后，运行时还必须额外管理视觉前缀、语法状态、工具回注上下文、外部执行等待、流式中断与恢复等状态对象。它们既不是纯粹的模型内部变量，也不是可以完全交给服务层处理的外围数据，而是会直接影响调度顺序、缓存复用和回退路径选择的运行时实体。复杂任务支持的核心，不只是完成能力接入，而是要让这些新增状态在 prefill、decode、验证、外部交互与回收之间有明确且低成本的迁移规则。

图8 按时间顺序压缩了复杂请求进入运行时后的生命周期，其重点不是再列一遍功能阶段，而是说明国内常见的多模态、结构化输出和工具调用请求，会在前处理、约束校验、外部回注和状态回收之间反复切换。因此，国产推理引擎的难点不只是“多支持一个能力”，而是如何在多个阶段之间维持稳定的状态传递和资源复用。

为了更直观地说明这种“能力补齐”为什么会迅速演变成系统问题，下面这段概念性伪代码把国产推理引擎在复杂请求下必须反复处理的几个关键决策显式化：请求要先

算法 1 复杂请求下的国产推理引擎运行时闭环（概念性伪代码）

Input: request stream R, platform capability set P
initialize scheduler state, cache state, and backend registry
for all incoming request $r \in R$ **do**
    classify r as text, long-context, multimodal, or tool-calling workload
    normalize prompt shape, length bucket, and service-level objective
    if r contains visual or audio prefix **then**
        run preprocessor and stage encoder outputs into shared runtime state
    end if
    if structured output or tool invocation is required **then**
        attach grammar state or external-call state to decoding context
    end if
    choose execution mode from eager, graph, or fallback path according to P
    update prefix cache, KV blocks, and migration metadata before execution
    while r is not finished **do**
        schedule prefill, decode, or verification step under current batch plan
        monitor TTFT/TPOT pressure, shape drift, and backend exceptions
        if constraint violation or backend instability is detected **then**
            trigger rollback, degrade execution mode, or rebalance request placement
        end if
    end while
    emit response, release cache blocks, and record observability signals
end for

被分类和整形，再决定是走 prefill、decode 还是多模态前处理分支；随后运行时还要同步维护图模式、缓存状态、结构化约束以及异常回退路径。也正因如此，复杂任务支持本质上是一个贯穿执行、调度与缓存的闭环，而不是附着在文本生成主路径外侧的附加功能。

推测解码、约束解码和复杂控制流生成进一步抬高了对执行一致性和调度灵活性的要求。SpecInfer 已经说明，新的生成加速机制会引入新的验证与回退路径，如果后端抽象和服务编排没有预留空间，局部优化反而会放大系统复杂度 [120]。Leviathan、Medusa、EAGLE 与 Lookahead Decoding 分别代表了辅助模型、多头预测、特征不确定性建模与并行前瞻等路线 [121–124]。这些方法的共同前提是运行时必须支持更细粒度的候选组织、验证与回退。硬件侧经验也说明收益高度条件化：在 MI300X 上，speculative decoding 主要受益于小 batch、带宽受限的 decode，随着 batch 增大收益会明显收缩；进入多模态 serving 后，text-only drafter 与 multimodal drafter 还要在额外开销和 acceptance length 之间重新权衡 [125, 126]。因此，推测解码不是可独立开启的局部优化，而是与 batch 组织、图执行模式、视觉 token 压缩和验证链路深度耦合的系统机制。这类现象对国产平台尤其关键，因为本地硬件往往在图模式切换、动态 shape 支持和异步执行语义上更敏感。一个在主流 GPU 平台上“只是多一次验证”的机制，迁移到新的后端后，可能同时意味着图缓存失效、batch 形状离散化加剧和异常回退链变长。于是，复杂任务支持是否可靠，最终评估的就不再只是某种 decoding 技术本身，而是系统能否在能力增强后仍维持可预测的状态空间规模。真正成熟的国产推理引擎，应当把这些新能力视作运行时状态机的自然扩展，而不是临时外挂在主循环外侧的特性开关。

5.3 压缩与成本主路径：压缩加速与单位 token 成本协同

对国产推理引擎而言，“低成本”并不是部署阶段最后再去追问的经济性指标，而是与执行效率和状态治理同级的系统目标。问题在于，压缩策略的收益往往不是线性兑现的：同一组量化参数在不同后端、不同上下文长度和不同负载强度下，可能分别表现为吞吐提升、尾时延恶化、状态迁移变多或精度明显波动。于是，很多团队虽然在模型层面已经得到不错的压缩结果，但一进入真实 serving 场景，就会发现低比特路径与通信粒度、图模式命中、KV 驻留层次和调度桶化同时发生耦合，使单位 token 成本难以稳定下降。

压缩问题在国产场景下更容易从“模型压缩”演变为“系统协同”问题。QServe 说明，只有把量化、kernel 组织和 serving runtime 一起设计，低比特收益才可能在真实服务路径中落地 [39]；KIVI、SnapKV 等工作则进一步表明，长上下文场景下的状态压缩如果不和缓存元数据、恢复路径与服务质量门限一起考虑，就会把容量节省重新转化为恢复时延或命中率损失 [34, 35]。对国产平台而言，这种耦合还会被放大，因为许多后端对低比特算子、动态 shape 和异步搬运的支持边界并不一致。所以，压缩加速真正困难的地方并不是有没有某种量化算法，而是系统能否把“配置参数–执行效率–状态占用–服务质量–单位 token 成本”组织成同一套可观测、可回归、可决策的闭环。如果不能做到这一点，那么压缩策略很容易停留在离线实验中的阶段性结论，难以成为国产推理引擎的稳定生产能力。

5.4 统一底座与演进秩序：上游演进与本地分叉维护

高性能推理框架仍在快速演化。国产后端如果通过侵入式修改共享路径获得短期收益，后续将面临高昂的合并成本。因此，如何通过插件、注册表、平台抽象和配置门控实现“可合并的本地优化”，是工程实践中的核心议题。这一问题的本质，是短期交付与长期演进之间的张力。许多本地优化在初期看似合理，例如直接修改共享 model runner、复制上游模块后做平台特化、或在公共调度路径中快速加入平台分支。这些做法能够尽快打通关键模型，但随着上游版本迭代，它们会迅速演变成难以维护的分叉。对综述对象中的国产推理项目而言，真正具有持续价值的能力，不仅是“适配成功”，而是“适配方式本身是否可持续”。从工程组织角度说，本地分叉之所以危险，还因为它会逐步改变团队的优化激励。若某个项目长期依赖复制上游模块并在本地修补，团队最容易积累的是“快速打补丁”的经验，而不是“设计稳定扩展面”的能力；随着时间推移，任何新模型、新特性和新后端都会优先以继续分叉的方式落地，最终导致系统在名义上仍与上游兼容，实则已经丧失合并能力。相反，若平台特化更多收敛在插件、注册表、解析器、模型映射和受控配置门面中，团队就更有机会把一次性的适配劳动沉淀为可复用的工程机制。以 vLLM-HUST 这类做法为例，其价值不只在于支持某个具体国产平台，更在于尝试把平台插件链、运行时护栏、多模态注册中心、reasoning parser 和 tool parser 这些高变动能力压在可独立演进的外围接口上 [76]。因此，讨论“上游演进与本地分叉维护”时，不能只把它理解为版本合并负担。更本质的问题是：一个国产推理引擎是否在逐步形成自己的模块化演进秩序。若平台适配、模型接入、协议兼容和复杂输出解析都能被压回清晰的扩展边界，后续新增能力就更可能表现为局部增量；反之，如果这些变化都直接冲击共享执行链，那么系统每次升级都会重新暴露整体脆弱性。这也是为什么 merge-safe 并不是“对上游友好”的附属要求，而是国产推理引擎能否持续演进的核心条件。

5.5 统一底座与证据链：评测体系与真实业务鸿沟

当前不少推理系统的优化仍主要围绕静态 benchmark 展开，例如固定 batch、固定上下文长度或单一模型族上的吞吐对比。但国产推理引擎面向的真实业务往往包含请求长度分布变化大、多租户并发、冷热模型混部、工具调用链条长和 SLA 约束严格等特征。若评测体系只覆盖理想化场景，就很难准确反映系统在生产环境中的优势与短板。因此，未来评测体系需要从“单点性能测试”扩展为“端到端工作负载评测”，将首 token 时延、稳定吞吐、缓存命中率、异常恢复时间、升级可用性和资源成本等指标纳入统一分析框架。这也是国产推理引擎从工程实现走向体系成熟的重要标志。在这一点上，OpenCompass 与 FlagPerf 分别代表了模型能力评测和系统性能评测两个侧面的生态努力 [127, 128]。前者帮助界定模型在多任务、多语言与复杂能力维度上的效果边界，后者则更强调硬件与系统栈在统一基准上的性能呈现。对于国产推理引擎而言，真正有价值的评测应当把这两类视角连接起来，即既看模型效果，也看支撑这些效果所付出的系统成本。

评测体系的短板并不只是“样本不够真实”，而在于很多关键系统代价缺乏统一表达。例如，长上下文服务是否真正受益于缓存复用，要看命中率、驱逐策略和跨阶段干扰；复杂调度是否有效，要看 SLO 达成率、迁移开销和队列整形效果；多模态 serving 是否可落地，则要看 encode/prefill/decode 三类阶段的资源占用是否被同时纳入统计。Llumnix、SOLA 与 ElasticMM 等工作之所以对综述有价值，正在于它们把动态迁移、SLO 感知与多模态阶段并行这些通常被 benchmark 忽略的系统变量显式化了 [9, 66, 67]。如果国产推理引擎缺乏这类近真实负载的度量框架，就很容易在评测环节高估局部优化、低估长期运维成本。

更进一步，评测体系本身也应当区分不同层次的问题。底层算子和内核优化适合用 microbenchmark 验证，其目的在于确认单点实现是否达到预期；运行时调度、缓存和回退策略需要用受控 serving workload 验证，其重点是吞吐、TTFT、TPOT 与尾时延；而平台演进能力、稳定交付能力和运维代价，则更适合通过持续集成基线、版本升级回归、真实流量回放和故障演练来评估。若这些层次被混在一起讨论，团队就很容易用“单点更快”替代“系统更稳”，用“离线吞吐更高”替代“生产交付更可持续”。国产推理引擎若要真正进入成熟阶段，评测闭环本身也必须从展示型 benchmark 转向决策型 benchmark，即能够直接回答哪一类路线在何种业务约束下更值得继续投入。

6 未来研究展望：由关键挑战走向长期收敛

第 5 节已将路线比较进一步收束为若干系统挑战，本节则把视角再上移一层，讨论未来几年更值得持续投入的研究方向。与其把“展望”理解为再列举若干可能出现的新功能，不如把它压缩为三条更稳定的主线：面向异构平台的能力契约与共享抽象、面向真实交付的控制面与证据链一体化治理，以及面向复杂请求图的状态中心化运行时。图9 所概括的，正是这三条主线在国产推理生态中的长期收敛关系。

未来演进大致受三组相互牵动的收敛压力支配：一是平台差异能否被沉淀为更稳定的能力边界与共享抽象，二是控制面、评测与交付能否从辅助工具上升为独立治理层，三是模型结构和任务形态变化会如何继续反向塑造系统边界。这些变化不是彼此独立的新功能，而是前文“一个底座、三条主路径”在未来几年内更可能沉淀出的高层研究议题。

这些研究方向并不漂浮在工程现实之外。它们之所以值得在第六章单独提出，恰恰

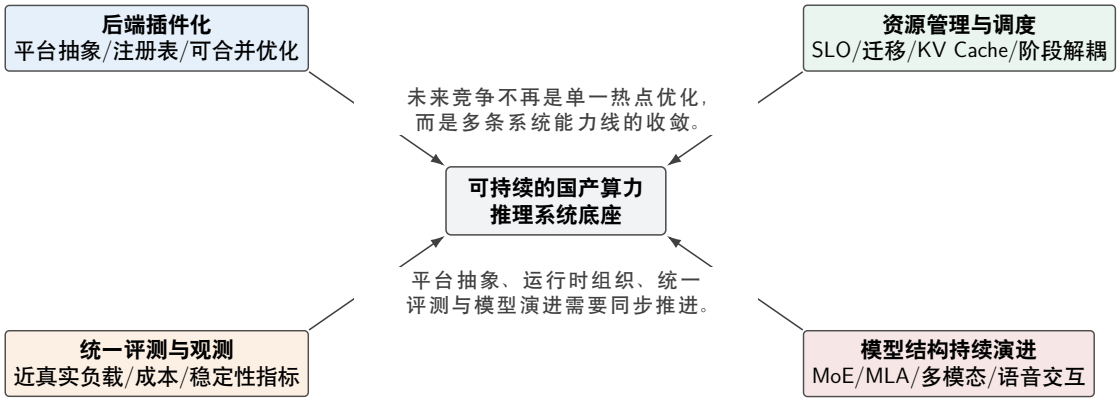


图 9: 国产推理引擎未来演进方向关系图

是因为相关压力已经开始倒逼不同主体重写分工：芯片与基础软件提供方需要把平台差异沉淀为稳定能力契约，开源推理框架维护者需要把复杂能力压回可扩展边界，平台交付方需要把控制面、版本矩阵和回归流程产品化，而研究与工程团队则更需要把工作负载、评测协议与系统证据链对齐。表11 给出的是这三条研究主线落到生态协同时所对应的主体视角路线图。

6.1 面向异构平台的能力契约与共享抽象

第一条研究主线关心的是，平台差异能否从“项目内部经验”上升为“生态可以共享的能力契约”。现有工程摩擦与公开研究共同提示，未来决定差距的未必只是某个热点 kernel 再快几个百分点，而更可能是 graph/eager 约束、通信语义、低比特路径和异常回退条件能否被稳定描述、持续验证并跨项目复用。若这一层抽象迟迟不能成形，国产适配就会长期停留在项目级 patch 与版本试错的循环中。

沿着这条主线继续推进，第 5 节讨论过的“能力契约”“merge-safe 扩展面”和“持续回归对象”就不再只是局部工程经验，而会转化为一组更明确的研究问题：后端能力应如何表达，才能既覆盖平台差异又不把短期特例固化为长期接口；回退语义应如何定义，才能保证图模式切换、缓存状态和分布式执行前后一致；共享抽象又应如何与平台特化协同演进，才能避免主链不断被项目级例外侵蚀。Llumnix、SOLA、LServe、AI Metropolis 与 ElasticMM 等工作之所以重要，也不仅因为它们提供了新的调度或状态机制，更因为它们显示出运行时的组织方式正在脱离“单一模型 serving 外壳”，转向更像资源与状态中枢的形态 [9, 53, 57, 66, 67]。

因此，这条研究主线最终要解决的并不是“谁来多写一些适配代码”，而是如何把“什么由平台保证、什么由引擎吸收、什么由控制面治理”经营成可预测边界。只有当平台差异能够被收敛为共享能力矩阵，而不是反复改写系统主链，新模型、新后端和新任务形态的进入才会从高风险迁移，变成较稳定的生态扩展。

6.2 控制面、评测与交付的一体化治理

第二条研究主线是，控制面能否从运维脚本和部署经验的集合，演化为连接评测、成本与交付的一体化治理层。成本、压缩和评测一旦进入同一闭环，版本矩阵、依赖树、部署模板、灰度发布、故障注入、回归脚本与证据链采集就不再只是帮助引擎跑起来的外围工具，而会越来越接近决定系统是否可交付、可升级、可回滚的核心层。就现有材料看，

这些能力一旦较早产品化，局部优化就更容易转化为组织能力。

这条主线之所以构成研究展望，而不只是工程 checklist，在于它要求把原本分散的问题重新抽象成统一对象。压缩收益要想稳定兑现，需要控制面维护策略基线与回退规则；评测要想真正支持决策，需要控制面维持近真实工作负载、版本口径和证据格式；平台适配要想减少人工排障，也需要控制面收纳依赖树、兼容矩阵和环境诊断。作为更偏主链复用与外围治理拆分的开源主干复用增强实践，vLLM-HUST 把平台插件、运行时护栏和环境治理拆开组织的做法，已经显露出“引擎主体 + 控制面工具链”的收敛方向 [76]；而官方适配仓库和平台文档长期强调的版本对齐、PD 分离、CANN/torch_npu 兼容和分布式连通性前置要求，也都在推动控制面从隐性经验变成显性层次 [73, 75]。

这一方向最终会收束为几个更明确的问题：近真实工作负载应如何标准化表达，才能让评测结果进入架构决策；版本基线、回退语义与故障注入应如何统一，才能让升级与灰度成本成为可治理对象；模型效果、系统成本与交付代价又应如何进入同一证据链，才能避免性能结论与生产结论长期脱节。执行面负责 token 生成，控制面负责让这些 token 处于可比较、可复现、可灰度和可回滚的条件之下；只有当控制面被视为与执行面并列的一等对象，前文讨论的成本、评测与交付问题才更可能从若干局部经验，转化为稳定的生态能力。

6.3 面向多阶段请求图的运行时重构

第三条研究主线指向运行时本身的重构。随着文本 decode、视觉 encode、语音前端、结构化约束验证、工具调用回注和多 Agent 协同被纳入同一服务链路，推理引擎越来越像是在管理一张多阶段请求图，而不再只是维护单一路径的 token 生成循环。对国产推理生态而言，这意味着后端接入、运行时组织、状态治理与交付工具链的分界会继续移动，原本按“模型家族”划定的系统边界也会越来越难维持。

模型侧的变化正在持续改写这张请求图的形状。以 DeepSeek 系列为代表的路线正在把问题从“模型结构更复杂”推进到“模型、推理内核、KV 基础设施与服务运行时是否一体收敛”：DeepSeek-V2/V3 披露的 MLA、MoE 与大规模服务经验，连同 FlashMLA、Mooncake 等配套工作，已经把通信组织、缓存布局、注意力实现和服务基础设施重新推回运行时中心位置 [44, 45, 92–94]。与此同时，Qwen2-VL、InternVL2、CogVLM2、GLM-4-Voice 与 MiniCPM-o 等模型也说明，视觉、视频和语音链路已经使文本时代的单一路径假设难以维持 [46–48, 115, 117]。这些变化放到第六章中，重点不再是逐项讨论运行时该如何补齐哪种状态治理能力，而是强调未来系统边界会被模型公司、应用形态和硬件路线共同牵引，任何一方的变化都会迫使另外两方重写分工。

从研究问题的角度看，这条主线最终落在三个层面：请求图中的状态转移应如何显式建模，才能避免多模态、工具调用和多 Agent 控制流把复杂度重新散落到外围特性开关中；共享缓存、编译产物与异常回退应如何围绕阶段关系而不是模型名单组织，才能让新任务形态进入时不必重写主链；而运行时又应如何同时容纳模型 co-design、状态治理和交付控制面，才能在复杂负载下维持可预测的系统边界。这个转变未必立刻带来最亮眼的跑分，但更可能决定国产推理系统能否在未来几年内持续吸收新结构、新任务和新平台。

7 结语

本文围绕“生态格局与路线分化–分析骨架与核心架构–基于骨架的路线比较–由路线比较收束出的关键挑战–未来收敛方向”这一主线，对国产算力平台上的大模型推理引擎作了结构化梳理。其核心目的，不是再补一份项目清单，而是把讨论从“谁支持更多模型、谁局部更快”转回到“复杂度被压在哪里、哪些能力已经沉淀为契约、哪些问题仍在共享主链上外溢”这一更稳定的分析层面。

全文表明，评价一条技术路线是否成立，不能只看局部算子、单个模型或一次 benchmark 上的峰值结果，而应看它能否在同一架构中同时承受后端异构、复杂任务负载和上游快速演进三类压力，并在国产芯片、国产基础软件栈和本土部署环境中形成可持续的工程闭环。本文将现有路线概括为开源主干复用增强路线、平台一体化方案与国产原生/独立路线三类，并指出，长期竞争力并不取决于路线名称本身，而取决于平台差异能否被收敛为能力契约，长上下文、多模态、语音和 Agent 负载能否被统一状态机承接，压缩配置与单位 token 成本能否进入运行时闭环，以及评测与交付工具链能否真实反映生产代价。国产推理引擎下一阶段的竞争重点，也不在单点峰值还能提高多少，而在于谁能更快、更稳、更低成本地吸收模型变化并完成生产交付。对产业侧而言，这意味着建设重点应从围绕单模型做专项优化转向围绕持续演进负载建设系统能力；对研究侧而言，则需要把平台迁移成本、长周期维护成本、复杂工作负载复用性和近真实服务指标放到与性能同等重要的位置。

概括而言，本文的核心判断可进一步压缩为四点：

(1) 平台异构本身不是问题的终点，真正决定路线可持续性的，是硬件差异能否被收敛为稳定、可验证、可回退的能力契约。

(2) 长上下文、多模态、结构化输出、工具调用与 Agent 负载的共同约束，正在把推理引擎从 token 生成器推向状态中心化运行时，因而状态治理能力将比局部算子优化更能决定真实上限。

(3) 量化、KV 压缩与低比特执行只有进入统一的观测、调度和成本闭环，才可能从离线实验结果转化为可持续交付的单位 token 成本优势。

(4) 国产推理引擎的路线选择，本质上是复杂度分配与责任边界选择：复杂度由 backend、runtime、service state machine、toolchain 还是控制面承担，将直接决定系统的演进速度、交付确定性和长期维护成本。

进一步看，后续研究与工程推进更适合落实为若干能够持续积累的核心问题：平台差异需要被沉淀为稳定、可回退、可验证的能力边界，而不是长期滞留在共享主链中的隐式特判；长上下文、多模态、结构化输出、工具调用与 Agent 负载正在持续改写状态对象的种类与流动方式，因此状态生命周期管理将成为比局部算子优化更稳定的竞争点；压缩收益、评测证据与交付控制面只有进入同一闭环，国产推理系统的成本优势和工程确定性才可能被真实兑现。围绕这些问题持续推进，国产推理引擎的演进才更可能从阶段性适配走向可持续积累。

参考文献

[1] Woosuk Kwon et al. Efficient memory management for large language model serving with pagedattention. *Proceedings of the ACM Symposium on Operating Systems Principles*, 2023.

[2] vLLM Team. vllm project. https://github.com/vllm-project/vllm, 2024.

[3] SGLang Team. Sglang project. https://github.com/sgl-project/sglang, 2024.

[4] Gyeong-In Yu et al. Orca: A distributed serving system for transformer-based generative models, 2023. arXiv:2309.06180.

[5] Amey Agrawal et al. Sarathi-serve: Efficient LLM inference by piggybacking decodes with chunked prefills, 2023. arXiv:2308.16369.

[6] Yongzhe Zhong et al. Distserve: Disaggregating prefill and decoding for goodput-optimized LLM serving, 2024. arXiv:2401.09670.

[7] Harsh Patel et al. Splitwise: Efficient generative LLM inference using phase splitting, 2024. arXiv preprint.

[8] Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models, 2024. MLSys 2025, arXiv:2411.15100.

[9] Zedong Liu, Shenggan Cheng, Guangming Tan, Yang You, and Dingwen Tao. Elasticmm: Efficient multimodal llms serving with elastic multimodal parallelism, 2025. NeurIPS 2025 Oral.

[10] Huawei Ascend. Mindie documentation. https://www.hiascend.com/document/detail/zh/mindie/100/index/index.html, 2026. Accessed: 2026-05-07.

[11] vLLM Project. vLLM Ascend: Community-maintained hardware plugin for ascend npu. https://github.com/vllm-project/vllm-ascend, 2026. Accessed: 2026-05-07.

[12] SGLang Project. SGLang Kernel NPU: Kernel library for ascend npu. https://github.com/sgl-project/sgl-kernel-npu, 2026. Accessed: 2026-05-07.

[13] Ascend. Triton-Ascend: Triton backend for ascend npu. https://github.com/Ascend/triton-ascend, 2026. Accessed: 2026-05-07.

[14] DeepLink Org. DLinfer: Connecting domestic hardware to llm inference frameworks. https://github.com/DeepLink-org/dlinfer, 2026. Accessed: 2026-05-07.

[15] InternLM / OpenMMLab. LMDeploy: A toolkit for compressing, deploying, and serving llms. https://github.com/InternLM/lmdeploy, 2026. Accessed: 2026-05-07.

[16] InternLM / OpenMMLab. LMDeploy installation documentation. https://lmdeploy.readthedocs.io/en/latest/get_started/installation.html, 2026. Accessed: 2026-05-07.

[17] PaddlePaddle. FastDeploy: Production-ready deployment toolkit for llms and vlms. https://github.com/PaddlePaddle/FastDeploy, 2026. Accessed: 2026-05-07.

[18] Baidu. vLLM-Kunlun: vllm hardware plugin for kunlun xpu. https://github.com/baidu/vLLM-Kunlun, 2026. Accessed: 2026-05-07.

[19] Cambricon. vLLM-MLU: vllm plugin for cambricon mlu. https://github.com/Cambricon/vllm-mlu, 2026. Accessed: 2026-05-07.

[20] Cambricon. MagicMind: Inference acceleration engine for cambricon mlu. https://github.com/Cambricon/magicmind_cloud, 2026. Accessed: 2026-05-07.

[21] Moore Threads. vLLM-MUSA: vllm hardware plugin for moore threads musa gpus. https://github.com/MooreThreads/vllm-musa, 2026. Accessed: 2026-05-07.

[22] Moore Threads. vLLM-MTT: vllm backend based on mt-transformer. https://docs.mthreads.com/mtt/mtt-doc-online/, 2026. Accessed: 2026-05-07.

[23] MetaX-MACA. vLLM-MetaX: vllm hardware plugin for metax gpus. https://github.com/MetaX-MACA/vLLM-metax, 2026. Accessed: 2026-05-07.

[24] Enflame Technology. vLLM-GCU: vllm adaptation for enflame gcu. https://github.com/EnflameTechnology/vllm-gcu, 2026. Accessed: 2026-05-07.

[25] Enflame Technology. vLLM-GCU online documentation. https://support.enflame-tech.com/onlinedoc_dev_3.6/3-model/infer/vllm0.9/content/source/index.html, 2026. Accessed: 2026-05-07.

[26] Iluvatar / DeepSpark. DeepSparkInference: Inference examples on iluvatar hardware. https://openi.pcl.ac.cn/iluvatar/ZhiKai100, 2026. Accessed: 2026-05-07.

[27] PaddlePaddle. FastDeploy installation on iluvatar corex. https://paddlepaddle.github.io/FastDeploy/get_started/installation/iluvatar_corex/, 2026. Accessed: 2026-05-07.

[28] PaddlePaddle. FastDeploy deployment on hygon dcu. https://paddlepaddle.github.io/FastDeploy/zh/get_started/installation/hygon_dcu/, 2026. Accessed: 2026-05-07.

[29] Biren Technology. Biren developer community. https://developer.birentech.com/, 2026. Accessed: 2026-05-07.

[30] EngineX-Biren. EngineX-Biren vLLM: vllm adaptation for biren supa. https://dev.modelhub.org.cn/EngineX-Biren/enginex-biren-vllm, 2026. Accessed: 2026-05-07.

[31] Vastai Technologies. Vastai developer center. https://vastai-tech.github.io/, 2026. Accessed: 2026-05-07.

[32] Xorbits. Xinference: Heterogeneous inference platform documentation. https://inference.readthedocs.io/zh-cn/latest/, 2026. Accessed: 2026-05-07.

[33] GPUStack. GPUStack: Open-source gpu cluster manager for ai model deployment. https://github.com/gpustack/gpustack, 2026. Accessed: 2026-05-07.

[34] Zirui Liu et al. Kivi: A tuning-free asymmetric 2bit quantization for KV cache, 2024. arXiv:2402.02750.

[35] Yuhui Li et al. Snapkv: Llm knows what you are looking for before generation, 2024. arXiv:2404.14469.

[36] Tri Dao et al. Flashattention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 2022.

[37] Tri Dao et al. Flashattention-2: Faster attention with better parallelism and work partitioning. *International Conference on Learning Representations*, 2024.

[38] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision, 2024. NeurIPS 2024, arXiv:2407.08608.

[39] Lei Wang et al. Qserve: W4a8kv4 quantization and system co-design for efficient LLM serving, 2024. arXiv:2405.04532.

[40] Reiner Pope et al. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 2023.

[41] Ying Sheng et al. Flexgen: High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning*, 2023.

[42] Guangxuan Xiao et al. Streamingllm: Efficient streaming language models with attention sinks, 2023. arXiv:2309.17453.

[43] LongServe Team. Longserve: Efficient LLM serving for long-sequence requests, 2024. arXiv preprint.

[44] DeepSeek-AI et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. arXiv:2405.04434.

[45] DeepSeek-AI. Deepseek-v3 technical report, 2024. arXiv technical report.

[46] Qwen Team. Qwen2-vl technical report, 2024. arXiv technical report.

[47] Zhe Chen et al. Internvl2: Better multimodal foundation models with stronger vision-language alignment, 2024. arXiv preprint.

[48] Aohan Zeng et al. Glm-4-voice: Towards intelligent and human-like end-to-end spoken chatbot, 2024. arXiv:2412.02612.

[49] Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flexattention: A programming model for generating fused attention variants. *Proceedings of Machine Learning and Systems*, 2025.

[50] Hanqing Jiang et al. Minference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention, 2024. NeurIPS 2024 Spotlight, arXiv:2407.02490.

[51] Guangxuan Xiao et al. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads, 2024. arXiv:2410.10819.

[52] Woosuk Kwon et al. vattention: Dynamic memory management for serving LLMs without pagedattention, 2024. arXiv:2405.04437.

[53] Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. Lserve: Efficient long-sequence llm serving with unified sparse attention, 2025. MLSys 2025, arXiv:2502.14866.

[54] Zhenyu Zhang et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. arXiv:2306.14048.

[55] Siyuan Yang et al. Pyramidinfer: Pyramid KV cache compression for high-throughput LLM inference, 2024. arXiv preprint.

[56] Yibo Zhang et al. Cachegen: KV cache compression and streaming for fast large language model serving, 2024. SIGCOMM 2024, arXiv:2310.07240.

[57] Zhiqiang Xie, Hao Kang, Ying Sheng, Tushar Krishna, Kayvon Fatahalian, and Christos Kozyrakis. Ai metropolis: Scaling large language model-based multi-agent simulation with out-of-order execution, 2025. MLSys 2025.

[58] NVIDIA. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM, 2025.

[59] Microsoft DeepSpeed Team. DeepSpeed-FastGen. https://github.com/microsoft/DeepSpeed, 2024.

[60] FlashInfer Team. FlashInfer. https://github.com/flashinfer-ai/flashinfer, 2025.

[61] NVIDIA. Multimodal support in tensorrt llm. NVIDIA TensorRT-LLM documentation, 2026.

[62] MIT HAN Lab. Omniserve: Unified and efficient inference engine for large-scale llm serving. GitHub repository, 2025.

[63] Haichen Zhang, Chang Liu, and Woosuk Kwon. vllm x amd: Efficient llm inference on amd instinct MI300X gpus. AMD developer technical article, 2024.

[64] Sean Song, David Silverstone, and Mohit Deopujari. Llm inference optimization using amd gpu partitioning. AMD ROCm blog post, 2026.

[65] Chandrish Ambati and Trung Diep. Amd mi300x gpu performance analysis, 2025. arXiv:2510.27583.

[66] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving, 2024. OSDI 2024, arXiv:2406.03243.

[67] Ke Hong, Xiuhong Li, Lufang Chen, Qiuli Mao, Guohao Dai, Xuefei Ning, Shengen Yan, Yun Liang, and Yu Wang. Sola: Optimizing slo attainment for large language model serving with state-aware scheduling, 2025. MLSys 2025.

[68] Hugging Face. Transformers. GitHub repository, 2025.

[69] Ray Team. Ray serve documentation. Ray Serve documentation, 2025.

[70] NVIDIA. Nvidia triton inference server. GitHub repository, 2025.

[71] LMSYS Org. Fastchat. GitHub repository, 2025.

[72] Lianmin Zheng et al. Sglang: Efficient execution of structured language model programs, 2023. arXiv:2312.07104.

[73] vLLM Ascend Team. vllm ascend project. https://github.com/vllm-project/vllm-ascend, 2025.

[74] OpenMMLab. LMDeploy. https://github.com/InternLM/lmdeploy, 2025.

[75] Huawei Ascend. MindIE: Model inference engine. https://www.hiascend.com/, 2025.

[76] vLLM-HUST Team. vLLM-HUST. GitHub repository, 2026.

[77] Cambricon. vLLM-MLU. https://github.com/Cambricon/vllm-mlu, 2026. Apache-2.0 GitHub repository, accessed 2026-05-08.

[78] SGLang Team. Ascend NPUs in SGLang. https://docs.sglang.ai/platforms/ascend_npu.html, 2026. Official platform documentation, accessed 2026-05-08.

[79] SGLang Team. SGLang ascend NPU development roadmap. https://github.com/sgl-project/sglang/issues/13664, 2026. GitHub roadmap issue, accessed 2026-05-08.

[80] LightLLM Team. Lightllm. https://github.com/ModelTC/lightllm, 2025.

[81] Hugging Face. Text generation inference. https://github.com/huggingface/text-generation-inference, 2025.

[82] MLC Team. Mlc-llm. https://github.com/mlc-ai/mlc-llm, 2025.

[83] Ollama Team. Ollama. https://github.com/ollama/ollama, 2025.

[84] llama.cpp Contributors. llama.cpp. GitHub repository, 2025.

[85] ExLlama Team. Exllamav2. GitHub repository, 2025.

[86] mistral.rs Contributors. mistral.rs. https://github.com/EricLBuehler/mistral.rs, 2025.

[87] Aphrodite Team. Aphrodite engine. https://github.com/aphrodite-engine/aphrodite-engine, 2025.

[88] KTransformers Team. Ktransformers. https://github.com/kvcache-ai/ktransformers, 2025.

[89] Predibase. Lorax. https://github.com/predibase/lorax, 2025.

[90] Huawei Technologies. Mindspore. Official website, 2025.

[91] Huawei Ascend. Compute architecture for neural networks (CANN). Official website, 2025.

[92] DeepSeek-AI. Flashmla. GitHub repository, 2025.

[93] Xupeng Qin et al. Mooncake: A KVcache-centric disaggregated architecture for LLM serving, 2024. arXiv:2407.00079.

[94] Xupeng Miao et al. Mooncake store: A KVcache-centric distributed store for LLM serving, 2024. technical report.

[95] thu-pacman. Chitu 「赤兔」. https://github.com/thu-pacman/chitu, 2025. GitHub repository, accessed 2026-05-07.

[96] xLLM Team. xLLM. https://xllm.readthedocs.io/zh-cn/latest/, 2026. 官方文档，accessed 2026-05-07.

[97] JD Open Source. xLLM-Service. https://github.com/jd-opensource/xllm-service, 2026. GitHub repository, accessed 2026-05-08.

[98] xLLM Team. xLLM technical report, 2025. arXiv:2510.14686.

[99] FastDeploy Team. FastDeploy documentation. https://paddlepaddle.github.io/FastDeploy/, 2026. Official documentation, accessed 2026-05-08.

[100] LMDeploy Authors. LMDeploy installation. https://lmdeploy.readthedocs.io/en/latest/get_started/installation.html, 2026. Official installation guide, accessed 2026-05-08.

[101] China Telecom. 中国电信完成业界首个面向国产算力的跨架构大模型推理技术验证. https://www.chinatelecom.com.cn/ct/news/jtxw/163649.html, 2025. Official news article on Triton-based cross-architecture LLM inference validation.

[102] LMDeploy Authors. LMDeploy supported models. https://lmdeploy.readthedocs.io/en/stable/supported_models/supported_models.html, 2026. Official supported-model matrix, accessed 2026-05-08.

[103] PaddlePaddle. FastDeploy. https://github.com/PaddlePaddle/FastDeploy, 2026. High-performance inference and deployment toolkit for LLMs and VLMs, accessed 2026-05-07.

[104] LightLLM Team. LightLLM documentation. https://lightllm-en.readthedocs.io/en/stable/, 2026. Official documentation, accessed 2026-05-08.

[105] Ruihao Gong, Shihao Bai, Siyu Wu, Yunqian Fan, Zaijun Wang, Xiuhong Li, Hailong Yang, and Xianglong Liu. Past-future scheduler for LLM serving under SLA guarantees. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2025.

[106] Lequn Chen et al. Punica: Multi-tenant lora serving, 2023. arXiv:2310.18547.

[107] S-LoRA Team. S-lora: Serving thousands of concurrent lora adapters, 2024. arXiv preprint.

[108] Guidance Team. Guidance. https://github.com/guidance-ai/guidance, 2025.

[109] Outlines Team. Outlines. https://github.com/dottxt-ai/outlines, 2025.

[110] MLC Team. Xgrammar. https://github.com/mlc-ai/xgrammar, 2025.

[111] PaddlePaddle. PaddleNLP LLM Server. https://github.com/PaddlePaddle/PaddleNLP/tree/develop/llm/server, 2026. PaddleNLP repository LLM service entry, accessed 2026-05-07.

[112] Qwen Team. Qwen2-vl. https://github.com/QwenLM/Qwen2-VL, 2025.

[113] OpenGVLab. Internvl. https://github.com/OpenGVLab/InternVL, 2025.

[114] THUDM. Cogvlm2. https://github.com/THUDM/CogVLM2, 2025.

[115] Weixiong Wang et al. Cogvlm2: Visual language models for image and video understanding, 2024. arXiv preprint.

[116] Yunfei Chu et al. Qwen2-audio technical report, 2024. arXiv:2407.10759.

[117] Junbo Cui et al. Minicpm-o 4.5: Towards real-time full-duplex omni-modal interaction, 2026. arXiv:2604.27393.

[118] Gursimran Singh, Xinglu Wang, Yifan Hu, Timothy Yu, Linzi Xing, Wei Jiang, Zhefeng Wang, Xiaolong Bai, Yi Li, Ying Xiong, Yong Zhang, and Zhenan Fan. Efficiently serving large multimodal models using EPD disaggregation, 2025. ICML 2025, arXiv:2501.05460.

[119] Zicong Hong, Yuyan Chen, Haoyue Zhang, Peng Li, Wuhui Chen, Song Guo, and Xiaowei Shen. Efficient multimodal serving via module multiplexing, 2026. EuroSys 2026.

[120] Xupeng Miao et al. Specinfer: Accelerating generative large language model serving with speculative inference and token tree verification, 2023. arXiv:2305.09781.

[121] Yaniv Leviathan et al. Fast inference from transformers via speculative decoding. *International Conference on Machine Learning*, 2023.

[122] Tianle Cai et al. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. arXiv:2401.10774.

[123] Yuhong Li et al. Eagle: Speculative sampling requires rethinking feature uncertainty, 2024. arXiv:2401.15077.

[124] Yichao Fu et al. Break the sequential dependency of LLM inference using lookahead decoding, 2024. arXiv:2402.02057.

[125] Sonali Singh, Karthik Sangaiah, Shenrun Zhang, Ryan Swann, and Ganesh Dasika. Accelerating llm inference: Up to 3x speedup on MI300X with speculative decoding. AMD ROCm blog post, 2025.

[126] Mohammad Mahdi Kamani, Parsa Fashi, Vikram Appia, and Emad Barsoum. Beyond text: Accelerating multimodal ai inference with speculative decoding on AMD instinct MI300X gpus. AMD ROCm blog post, 2025.

[127] OpenCompass Team. Opencompass. GitHub repository, 2025.

[128] FlagOpen. Flagperf. GitHub repository, 2025.

表 3: 国产推理引擎常用指标的统一口径

指标/对象	统一定义	使用边界与解释重点
TTFT	请求提交到首个可见 token 返回的时间，通常覆盖排队、前处理、prefill 与首次采样	适合衡量在线交互体验；若比较跨平台 TTFT，应同时说明 prompt 长度、batch 组织、是否包含多模态前处理与图编译预热
TPOT	稳态生成阶段相邻两个输出 token 之间的平均时间	主要反映 decode 主路径效率；应与 batch 大小、采样策略和 speculative decoding 配置一并解释，避免脱离负载单独比较
吞吐/goodput	单位时间完成的 token、请求或满足 SLO 的有效工作量	吞吐适合描述系统上限，goodput 更适合描述真实服务收益；若只报告吞吐，容易掩盖尾时延和回退开销
KV 命中率/迁移开销	前缀复用成功比例，以及状态跨层级、跨实例或跨节点搬移所付出的流量与时延	适合评估长上下文、多轮会话和 PD/EPD 路径；必须结合请求分布、缓存策略和驱逐规则解释
图模式命中率/回退率	请求在 graph 模式稳定执行的比例，以及回退到 eager 或其他安全路径的频率	适合评估 shape 敏感后端的稳定性；若不同时给出回退原因，单独的图命中率意义有限
单位 token 成本	在既定质量和 SLO 约束下，生成单位 token 所需的综合资源成本	不应被理解为静态硬件单价，而应同时吸收量化配置、状态容量、迁移流量、实例利用率与运维开销
可复现性证据	支撑结论的代码、配置、版本矩阵、trace、日志与工作负载描述是否完整	不是单一性能指标，而是路线比较的证据门槛；缺少该项时，结论应降格为趋势判断或工程观察

表 4: 混合式开源推理引擎组件分层与国产算力潜在优化点

组件层	主要职责	可结合国产算力的潜在优化方向
入口与启动护栏	CLI、OpenAI 兼容服务、启动时序控制、运行时预检与环境初始化	针对国产驱动/编译链构建更稳定的启动护栏，在引擎启动前完成 NPU 预检、环境变量自动补齐、依赖矩阵校验与故障前移，减少平台问题侵入执行主路径
配置与能力装配	参数收敛、统一配置对象组装、不同能力开关与默认值选择	为国产后端提供平台感知的默认配置策略，如图模式开关、bucket 粒度、KV 容量预算、低比特路径和分布式后端选择，使优化从零散参数经验转为可复用配置模板
引擎编排与 EngineCore	请求对象转换、流式生命周期管理、调度与 core 通信	面向 shape 敏感后端优化连续批处理、prefill/decode 解耦、SLO 感知调度与前缀局部性管理，并在多进程/多实例形态下减少编译抖动和跨阶段干扰
模型执行与热点算子	模型注册、权重加载、worker/runner、attention 与量化等热点路径	围绕国产硬件补齐 fused attention、MLA/MoE 相关算子、低比特 kernel、图安全回退路径和异步搬运策略，把平台特征转化为稳定吞吐和时延收益
平台插件与运行时治理	平台探测、插件注入、后端激活、环境修复与外围治理工具衔接	把国产平台差异收敛为能力契约，通过插件化方式承载 dtype、attention backend、compile backend、distributed backend 与 runtime 修复逻辑，降低对共享主干的侵入深度
多模态、Reasoning 与 Tool 横切层	多模态注册、IO 处理、reasoning parser、tool parser、复杂输出协议适配	把视觉 encoder、语音前端、结构化输出与工具调用状态统一纳入生命周期管理，并结合国产后端对 encode/decode 分段执行、跨模态缓存和 parser 前后处理开销做协同优化
编译、KV Cache、分布式与 tracing 基础设施	graph/compile、缓存管理、并行执行、监控与链路追踪	针对国产编译栈强化 graph cache 命中、piecewise backend 选择、KV 驻留层次、通信路径调优与近真实负载观测，使性能优化、容量管理和运行时诊断形成闭环

表 5: 国产算力推理路线的复杂度落点

技术路线	代表对象	复杂度主要落点	对国产算力适配的关键意义
插件式后端适配	vLLM-Ascend、vLLM-MLU、SGLang Ascend[73, 77, 78]	plugin backend、attention backend、分布式运行时、版本矩阵	以较低上层接口改动把 Ascend NPU、寒武纪 MLU 等国产硬件纳入 vLLM/SGLang 语义，但成熟度受 CANN、torch_npu、厂商 SDK 与上游版本节奏约束
平台一体化	MindIE[75]	SDK/runtime/compiler、服务模板、监控与交付链路	把 Ascend 平台的编译、运行时和服务化交付收敛到平台栈内部，适合强调统一交付的场景，但不宜写成完全开源的通用引擎
国产独立引擎	Chitu、xLLM[95–98]	service state machine、PD/EPD、KV Cache、低精度与多硬件部署闭环	说明国产推理系统不只是在主流框架中补后端，也可以围绕状态治理、缓存治理和部署闭环重构引擎边界
工具链型部署	FastDeploy、LMDeploy[99, 100, 102]	模型接入、OpenAI/vLLM 风格接口、量化路径、硬件安装矩阵	将国产算力适配沉淀为“部署工具链 + 服务接口 + 硬件适配层”，但不同后端和不同硬件上的 feature parity 需要逐项核验
跨架构编译/运行时探索	中国电信 Triton 跨架构框架 [101]	compiler/runtime 抽象、统一算子库、透明运行时插件	试图缓解多国产芯片重复适配问题；当前更适合作为跨架构抽象探索，不能写成已开源成熟 serving 引擎

表 6: 结合社区实战信号的国产推理引擎路线选型提示

代表对象	更适合的前提	社区中反复暴露的工程摩擦	更值得关注的系统收益	更接近的组织选择
vLLM-Ascend	已有 vLLM 或 OpenAI 兼容服务栈，希望优先继承上游模型支持与接口形态	上游主干、插件分支、CANN 与 torch_npu 需要严格对齐；PD 分离、多机组网和环境预检常成为首轮障碍	以较低上层改动成本验证国产迁移路径，较快吸收上游 serving 特性	生态跟进速度优先、能承受版本协同与环境治理成本的团队
MindIE	更重视镜像、模板、监控、服务组件和平台统一交付，希望把部署复杂度收敛到平台栈内部	配置参数、服务模板、平台组件依赖和网络环境准备通常会显著影响部署成功率	更有机会获得交付责任集中、运维路径清晰和平台确定性较强的交付形态	行业部署、政企交付或平台统一性优先的组织
Chitu	希望直接采用独立推理引擎组织多元算力兼容、弹性部署和企业落地能力 [95]	独立引擎需要同时经营模型支持矩阵、服务接口和交付模板，若模块边界不清晰，演进成本会快速上升	更容易把多硬件兼容、生产可用性和部署弹性放到同一套自研内核里统筹	既希望摆脱单一上游主干依赖、又具备持续经营自研内核能力的团队
xLLM	希望围绕国产芯片同步建设推理框架与部署入口，把一键拉起、OpenAI 兼容与预验证能力前置到工程闭环中 [96, 97]	需要同时维护推理内核、分布式执行与部署入口的一致性，避免把入口便利性建立在脆弱的环境假设上	更容易形成面向国产芯片的“框架 + 部署入口”一体化落地路径	希望把国产化部署经验沉淀为统一操作入口和服务闭环的团队
LMDeploy	同时看重推理性能、模型支持矩阵和工程化工具链，希望保留较强部署灵活性	国产 GPU 适配成熟度、量化路径、不同硬件上的最佳实践需要逐平台核验	部署效率和高性能工具链相对均衡，适合快速工程试验与模型接入	追求性能与工程效率折中、愿意做平台侧验证的团队
SGLang	更重视复杂 Agent、长上下文前缀复用、结构化输出和请求编排，希望把复杂控制流直接写进 serving 运行时	国产化部署模板和硬件侧现成经验相对少，工程门槛更多落在负载理解与执行模式选择上	在复杂控制流、长 system prompt 复用和结构化输出场景下更可能体现运行时组织优势	复杂工作负载优先、愿意为执行模型与调度抽象投入理解成本的团队
vLLM-HUST 类混合式路线	不愿放弃上游接口兼容，同时希望把国产平台护栏、插件与治理工具逐步沉淀为独立工程层	需要持续经营模块边界，避免既继承上游复杂度又堆积本地特化补丁；对工程架构能力要求更高 [76]	在接口兼容、平台特化和运行时治理之间形成可持续折中	既要长期跟进上游生态，又要持续建设本地化治理能力的团队

表 7: 国内部署路线案例比较

代表对象	路线类型	工程落点	部署目标
vLLM-Ascend[73]	开源主干后端适配	围绕上游接口扩展后端、执行器与定制算子	快速跟进上游能力，优先保障生态兼容与模型覆盖
vLLM-MLU[77]	开源主干后端适配	依托 vLLM plugin system 接入寒武纪 MLU，并覆盖 Chunk Prefill、Prefix Caching、Spec Decode 等 vLLM 能力	以继承 vLLM 接口与调度语义为主，但复现性仍受 SDK/runtime 与公开 benchmark 粒度约束
SGLang Ascend[78, 79]	主线后端支持	在 SGLang 主线中为 Ascend NPU 提供 attention backend、PD 分离部署和组件版本映射	体现国产 NPU 被主流 serving 框架吸纳，但高级能力成熟度需与 roadmap 区分
vLLM-HUST[76]	混合式架构	保持 vLLM 上层接口，外挂平台插件并补齐运行时护栏与环境治理	兼顾上游兼容、国产平台接入与部署运维闭环
MindIE[75]	一体化部署	与本地编译器、运行时和交付链路深度协同	平台统一交付、稳定运行和行业方案落地优先
Chitu[95]	独立自研引擎	面向多元算力组织推理、弹性部署与企业落地能力	同时重视多硬件兼容、部署弹性与生产可用性的场景
xLLM[96, 97]	自研框架 + 部署入口	围绕国产芯片组织开源智能推理框架，并以前端部署入口补齐一键拉起、OpenAI 兼容与预验证能力	希望同步建设推理内核与部署闭环的国产化落地场景
LMDeploy[74]	工具链强化	围绕模型支持、推理工具链和工程化部署体验持续扩展	兼顾高性能部署、模型覆盖和工程效率
FastDeploy / PaddleNLP Server[103, 111]	工具链强化 + 服务封装	以高性能推理与部署工具包衔接模型导出、服务封装和应用接入	希望围绕飞桨生态组织 LLM/VLM 推理与部署链路的场景
中国电信 Triton 统一跨架构框架 [101]	跨架构编译/运行时	通过 Triton 跨架构编译器、统一算子库和 vLLM-Triton 透明插件探索多芯迁移	用于缓解多芯重复适配问题，但未开源且许可证与复现脚本未明确

表 8: 国内团队多模态推理链路工程关注点比较

代表模型	主要链路	关键工程压力	对推理引擎的直接要求
Qwen2-VL、InternVL2[46, 47]	多图、文档与 OCR	动态分辨率切图、视觉 token 膨胀、图文混排带来 shape 波动	视觉前处理与文本 prefill 协同、前缀缓存复用、bucket 化与编译抖动控制
CogVLM2[115]	视频理解	帧采样与跨帧压缩策略影响 encode 与 decode 资源分配	视觉 encoder 批处理、跨阶段资源再平衡、视频前缀组织
Qwen2-Audio 等 [48, 116, 117]	语音与实时交互	音频前端、speech tokenizer、流式 chunk 与打断恢复共同影响时延	流式调度、语音文本交错生成、实时率控制与会话状态恢复

表 9: 国产推理引擎关键挑战比较

挑战来源	主要冲击层	典型后果
硬件异构与后端碎片化	执行层、算子接口、通信路径	共享路径分叉、正确性边界不一致、维护成本持续上升
复杂任务与多模态负载	调度、缓存、图模式与前处理链路	阶段争用加剧、时延波动放大、局部功能可用但端到端不稳定
压缩收益与系统成本错配	量化路径、KV 状态、调度与成本核算	低比特收益不稳定、精度与时延相互牵制、单位 token 成本难形成闭环
上游快速演进与本地分叉	平台抽象、插件接口与共享模块	patch 累积、合并困难、对新模型和新特性的跟进滞后
评测体系与真实业务脱节	benchmark 设计、运维指标与资源评估	优化方向失真、离线结论与生产表现偏离

表 10: 不同国产推理路线与关键挑战的映射关系

代表路线	版本矩阵/环境门槛	网络与分布式风险	缓存/调度压力	运维与回退焦点	更需要优先建设的系统能力
vLLM-Ascend 类开源后端适配	高。上游主干、插件分支、CANN 与 torch_npu 对齐要求强，环境预检缺失时很容易在启动阶段失败 [73]	中到高。PD 分离、多机 RoCE/HCCL 连通性和阶段解耦常在扩容时暴露问题	高。更容易先沿用上游连续批处理与 KV 组织，再逐步补本土平台约束，长上下文和复杂 shape 下调度抖动更敏感	高。需要围绕环境校验、插件失配、图模式回退与服务启动护栏建立可观测修复路径	平台能力探测、版本矩阵校验、分布式连通性检查和主链外侧治理工具
MindIE 类一体化方案	中。更多复杂度被平台镜像、模板和组件版本收敛，但外部团队对底层差异的显式控制较弱 [75]	中。网络与服务编排问题更多表现为平台部署参数、服务组件协同与交付流程约束	中。缓存、调度与编译优化更容易被平台内部统一处理，但也更依赖平台默认策略与产品节奏	高。监控、配置模板、回滚路径与交付责任集中，是一体化路线真正的成败点	模板化部署、监控告警、配置基线、统一回归与平台级问题定位闭环
LMDeploy 类工具链强化路线	中。模型支持矩阵、量化路径和不同硬件上的最佳实践需要逐平台核验 [74]	中。分布式风险通常不先表现为链路不可用，而是表现为不同部署形态下性能与稳定性的权衡不透明	中到高。工具链强调高性能部署与模型接入效率，但在国产 GPU/信创环境中，缓存策略、量化收益和调度参数常需要额外试配	中到高。选型时更关心部署脚本、量化回退、多模型兼容和工程试验效率	面向不同平台的性能基线、量化/非量化双路径、模型接入自动化和快速回归脚本
vLLM-HUST 类混合式路线	中到高。既要跟上上游接口变化，又要维持本地插件、护栏和治理层的边界稳定 [76]	中到高。网络与分布式问题不仅来自链路本身，还来自哪些能力留在主链、哪些能力下沉到外围治理层的架构选择	高。要同时承接上游 serving 主体与本地化能力扩展，缓存、调度、多模态和复杂控制流更容易形成横向耦合	高。若模块边界经营不好，最容易同时继承上游复杂度与本地特化补丁负担	模块化扩展面、插件契约、外围治理工具、复杂能力注册中心与 merge-safe 演进机制

表 11: 三条研究主线在国产推理生态中的主体分工与演进重点

主体	更应优先收敛的能力	近期更现实的建设重点	中期更关键的演进目标
芯片厂商与基础软件栈提供方	设备能力表达、编译/运行时边界、通信语义与低比特路径稳定性	减少版本矩阵碎片化，明确 graph/eager、attention backend、通信后端和量化 kernel 的可用边界	把平台差异沉淀为可验证的能力契约，而不是持续依赖项目级特判和临时 patch
开源推理框架维护者	插件接口、模型执行抽象、复杂能力扩展面与回退语义	为国产后端保留更稳定的 backend、cache、compile、tool/reasoning 扩展点，降低共享主链分叉压力	让多模态、结构化输出、Agent 与长上下文能力在主线抽象内持续演进，而不是被外围分支各自重写
平台交付方与云边部署团队	控制面、灰度回滚、监控告警、版本基线和故障注入流程	把镜像、驱动、依赖树、部署模板和回归脚本产品化，降低“能跑起来”对人工经验的依赖	形成统一的交付控制面，使性能回归、环境稳定性和单位 token 成本进入同一治理流水线
研究团队与工程集成团队	工作负载建模、评测协议、状态治理策略与证据链组织	用近真实 serving workload 替代单一展示型 benchmark，补齐 TTFT、TPOT、KV 命中率、回退率等指标口径	建立能连接模型效果、系统成本和交付代价的分层评测体系，使路线比较真正支持架构决策
模型公司与上层应用构建者	模型结构变化对运行时边界的约束表达、复杂请求生命周期抽象	在发布新模型能力时同步暴露对缓存、调度、约束解码和多阶段执行的系统需求	推动模型设计、推理内核与服务运行时更紧密 co-design，避免系统复杂度全部滞后暴露在部署阶段