TPU Inference Externalization Full Steam Ahead - InferenceX

TPU 推理外部化全面加速 - InferenceX

InferenceX, Up to 50% Better Performance per Dollar, Rapid Externalization of TPU stack, Growing Customer Base, Ironwood, TPUv8i, Reducing CUDA Moat

InferenceX，每美元性能提升高达 50%，TPU 技术栈快速外部化，客户群持续增长，Ironwood，TPUv8i，缩小 CUDA 护城河

ALEC BARRA, CAM QU L CI, BRYAN SHAN, AND 4 OTHERS

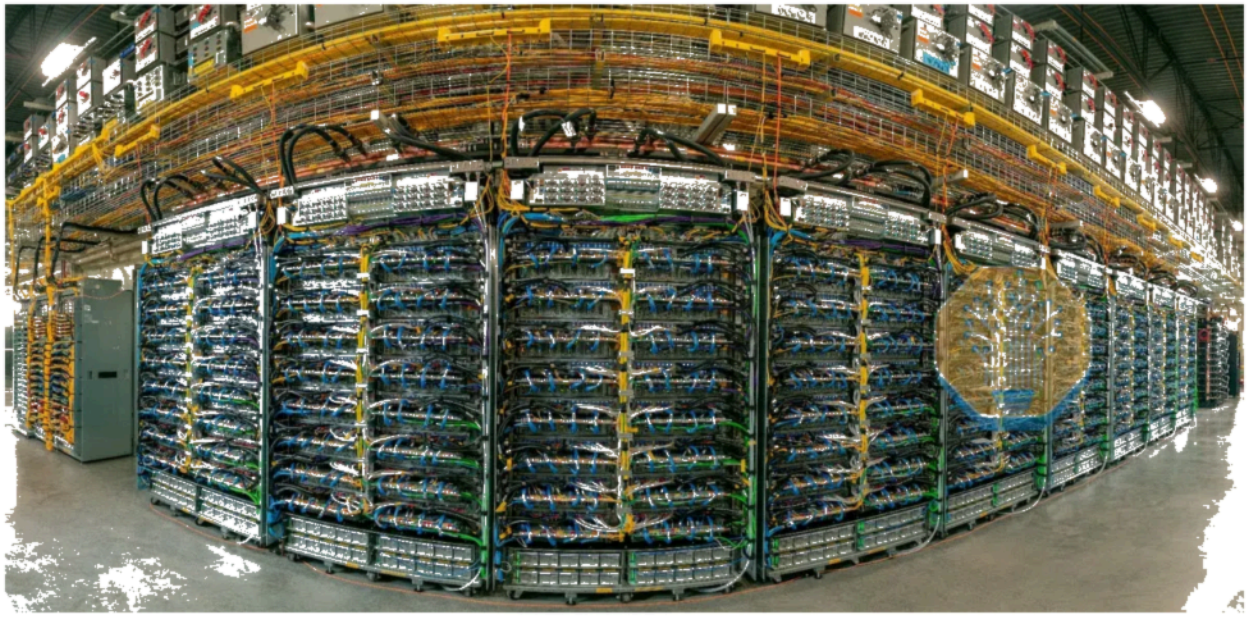
[illegible]

SEP 08, 2026 · PAID · 付费

68 6 Share 分享

For more than a decade, the industry has watched Google build an empire on its own silicon. Search, Ads, YouTube, and every generation of Gemini run on TPUs. Few accelerators have attracted as much architectural scrutiny or as much debate about what their performance and economics would look like outside the company that designed them. Anthropic being the biggest user of TPUs, surpassing Deepmind's own use by 2029.

十多年来，业界一直目睹谷歌凭借自家芯片构建起一个帝国。搜索、广告、YouTube，以及每一代 Gemini 都运行在 TPU 上。很少有加速器能像 TPU 这样，引发如此多的架构审视，以及关于其性能和成本效益在谷歌之外会如何表现的激烈争论。Anthropic 成为 TPU 最大的用户，到 2029 年其使用量将超过 Deepmind 自身。



Source: Google 来源：谷歌

Google's internal success was never the question. The question was how much of that advantage the rest of the industry could actually get. Could you take an open-weight model, serve it through a familiar inference engine, and beat NVIDIA on the economics that matter to your business?

谷歌内部的成功从来不是问题所在。问题在于，业界其他公司究竟能从中获得多少优势。你是否能拿一个开放权重模型，通过熟悉的推理引擎来部署，并在对你业务至关重要的成本效益上击败 NVIDIA？

Today, we are publishing the first third-party inference results for TPUv7 Ironwood on InferenceX Official Preview. In our apples-to-apples comparisons against B200/B300, Ironwood delivers up to 50% better performance per dollar. Its advantage extends across much of the Pareto curve, and we examine the economics from both sides: Google's internal total cost of ownership and the external TCO an actual customer pays.

今天，我们在 InferenceX 官方预览版上发布了 TPUv7 Ironwood 的首批第三方推理结果。在我们与 B200/B300 的同类对比中，Ironwood 每美元性能提升高达 50%。其优势延伸至帕累托曲线的大部分区域，我们从两个角度审视其成本效益：谷歌内部的总体拥有成本，以及实际客户支付的外部总体拥有成本。



TPUv7: Google Takes a Swing at the King

TPUv7：谷歌向王者发起冲击

DYLAN PATEL, MYRON XIE, AND 10 OTHERS

DYLAN PATEL、MYRON XIE 及其他 10 人 · 2025年11月28日 2025 年 11 月 28 日

Read full story 阅读全文 →

Ironwood (TPUv7) is the first generation in which Google is competing for others' inference workloads with chips that can be purchased outright or rented through its own cloud. In November 2025, we already said that Anthropic loves TPUs and committed to over one million of them (around 400k+ in direct purchases and 600k+ rented through GCP), used mainly for training but also for inference. Our Accelerator Model has the latest figures for Anthropic's TPU shipments and Google's overall TPU shipments by quarter, plus estimates for TPUv8i, v8t, and various v9 / v10, and more

Ironwood（TPUv7）是谷歌首代通过可直接购买或自家云租赁的芯片，来竞争他人推理工作负载的产品。2025 年 11 月，我们已提到 Anthropic 青睐 TPU，并承诺部署超过一百万颗（其中约 40 万+为直接购买，60 万+通过 GCP 租赁），主要用于训练，也用于推理。我们的加速器模型提供了 Anthropic TPU 出货量及谷歌整体 TPU 按季度出货的最新数据，并包含对 TPUv8i、v8t 及各类 v9/v10 的估算，以及更多信息。

We are excited by how quickly the new TorchTPU stack is developing, the external stack for TPUs. Later in the article, we will discuss the upcoming work needed for TPU software externalization, including optimizing speculative decoding,

disaggregated prefill, KV-cache offloading, multi-turn agentic workloads, and more. Even so, we at SemiAnalysis **strongly believe that TPU externalization is heading in the right direction and moving full steam ahead.** Furthermore, unlike AMD, which is still learning how to build a test-first software culture, Google has decades of software engineering experience and an extremely well established quality-driven culture, so we expect external TPU software to mature rapidly.

我们对新 TorchTPU 技术栈（TPU 外部技术栈）的快速发展感到振奋。在本文后续部分，我们将讨论 TPU 软件外部化所需的后续工作，包括优化投机解码、分离式预填充、KV 缓存卸载、多轮智能体工作负载等。尽管如此，SemiAnalysis 坚信 TPU 外部化正朝着正确方向全速前进。此外，与仍在学习构建测试优先软件文化的 AMD 不同，谷歌拥有数十年的软件工程经验和极为成熟的质量驱动文化，因此我们预期外部 TPU 软件将迅速成熟。

In this article, we will cover all the optimizations that went into TPU kernels and the serving stack for open-weight models, including DP attention optimization, MoE routing and kernel optimization, and reducing padding in GDN kernels. We will also take a deep dive into the TPU system and discuss the next steps the amazing TPU performance engineers are pursuing to make the stack widely available.

在本文中，我们将介绍针对开放权重模型在 TPU 内核和服务栈中所做的所有优化，包括 DP 注意力优化、MoE 路由与内核优化，以及减少 GDN 内核中的填充。我们还将深入探讨 TPU 系统，并讨论出色的 TPU 性能工程师为使该栈广泛可用而正在推进的后续步骤。

Google has spent more than a decade demonstrating what it can build with TPUs. Now we get to measure what the rest of the industry can do with them.

谷歌花了十多年时间展示其利用 TPU 所能构建的成果。现在，我们得以衡量行业其他参与者能利用 TPU 做些什么。

Shoutout to the Google (Chris Chan, Jahangir Hasan, Wangyuan Zhang, Anne Stern, Puneith Kaul, Ruizi Dong, Sangam Jindal, Qi Zhou, Madhan Jaganathan, Gang Ji, Jun Wan, Devanshu Jain, Jiaxin Cao, Srinath Mandalapu, Haowen Ning) and Inferact teams for this amazing TPU foundation and performance! Furthermore, shoutout to the RadixArk team that is also working on TorchTPU SGLang.

感谢谷歌（Chris Chan、Jahangir Hasan、Wangyuan Zhang、Anne Stern、Puneith Kaul、Ruizi Dong、Sangam Jindal、Qi Zhou、Madhan Jaganathan、Gang Ji、Jun Wan、Devanshu Jain、Jiaxin Cao、Srinath Mandalapu、Haowen Ning）和 Inferact 团队为这一出色的 TPU 基础与性能所做的贡献！此外，还要感谢同样致力于 TorchTPU SGLang 的 RadixArk 团队。

InferenceX Official Preview: TPUv7 Ironwood vs. Blackwell and Blackwell Ultra

InferenceX 官方预览：TPUv7 Ironwood 对比 Blackwell 与 Blackwell Ultra

We are already seeing strong results from the upcoming native TorchTPU vLLM stack in apples-to-apples comparisons against Nvidia GPUs. Google is using Qwen3.5 397B in FP8 as the initial bring-up model. Later sections take a deep dive into why the new TorchTPU approach is a marked improvement over the previous TorchAX path for external TPU vLLM/SGLang serving.

在与 Nvidia GPU 的同类对比中，我们已经看到即将推出的原生 TorchTPU vLLM 栈展现出了强劲成果。谷歌正使用 FP8 格式的 Qwen3.5 397B 作为初始启动模型。后续章节将深入探讨为何新的 TorchTPU 方法相较于之前的 TorchAX 路径，在外部 TPU vLLM/SGLang 服务方面有显著改进。

Once that foundation is in place, Google plans to extend support to other open-weight models, including Kimi K3 and GLM5.3. Once a handful of models are well optimized, we believe adding optimized support for a wide range of popular open models near day 0 becomes far easier. Today, vLLM and SGLang concentrate their day-0 support on Nvidia, with passable day-0 coverage for AMD. We expect TorchTPU to be stable enough in the near future that vLLM and SGLang maintainers may add TPUs to that day-0 list. The stack is expected to leave private beta & be open sourced around October.

一旦这一基础就绪，谷歌计划将支持扩展到其他开放权重模型，包括 Kimi K3 和 GLM5.3。一旦少数模型得到充分优化，我们相信为广泛流行的开源模型提供近乎首日即用的优化支持将变得容易得多。目前，vLLM 和 SGLang 将首日支持集中在 Nvidia 上，对 AMD 的首日覆盖尚可。我们预计 TorchTPU 在不久的将来会足够稳定，vLLM 和 SGLang 的维护者可能会将 TPU 加入首日支持列表。该技术栈预计将于十月左右退出私人测试阶段并开源。

In apples-to-apples comparisons of aggregated serving with FP8 and single-token prediction, we are seeing up to 50% better performance per dollar from TPU than from B200 and B300 running FP8 in aggregated serving. When serving models using FP4 on NVIDIA GPUs, there is quality loss verus FP8. TPUv7 does not have native FP4 computation thus on FP4, NVIDIA GPUs still maintains the lead. This will change with TPUv8i which has native FP4 support thus we strongly believe TPUv8i Boardfly

will be competitive to Rubin NVL72.

在 FP8 聚合推理和单 token 预测的同口径对比中，我们看到 TPU 相比运行 FP8 聚合推理的 B200 和 B300，每美元性能高出最多 50%。在 NVIDIA GPU 上使用 FP4 推理模型时，与 FP8 相比存在质量损失。TPUv7 不具备原生 FP4 计算能力，因此在 FP4 方面，NVIDIA GPU 仍保持领先。这一局面将随着 TPUv8i 的到来而改变——TPUv8i 原生支持 FP4，因此我们坚信 TPUv8i Boardfly 将对标 Rubin NVL72 并具备竞争力。

We believe the TPU performance-per-dollar advantage can persist once Google enables disaggregated serving and speculative decoding, with MTP and disaggregated serving enabled on both sides of the comparison. Later in the article, we discuss the optimizations Google plans to add once the TorchTPU foundation is in place. To keep the bring-up scope contained, Google has initially focused on benchmarking 8k1k. However, we believe that the TorchTPU stack will also perform amazingly well on agentic workloads & will be publishing results for those too later in the year. Later on in the article, we will talk about the next steps & roadmap of TPU external Inference.

我们相信，一旦谷歌启用分离式推理和推测解码，并在对比双方都启用 MTP 和分离式推理的情况下，TPU 在每美元性能上的优势可以持续保持。在文章后面部分，我们将讨论谷歌在 TorchTPU 基础就绪后计划添加的优化。为了控制初始部署范围，谷歌最初专注于 8k1k 的基准测试。然而，我们相信 TorchTPU 技术栈在智能体工作负载上也将表现出色，并将在今年晚些时候发布相关结果。在文章后续部分，我们将讨论 TPU 外部推理的下一步计划与路线图。

TPU is King on Performance per Dollar
TPU 在每美元性能上独占鳌头

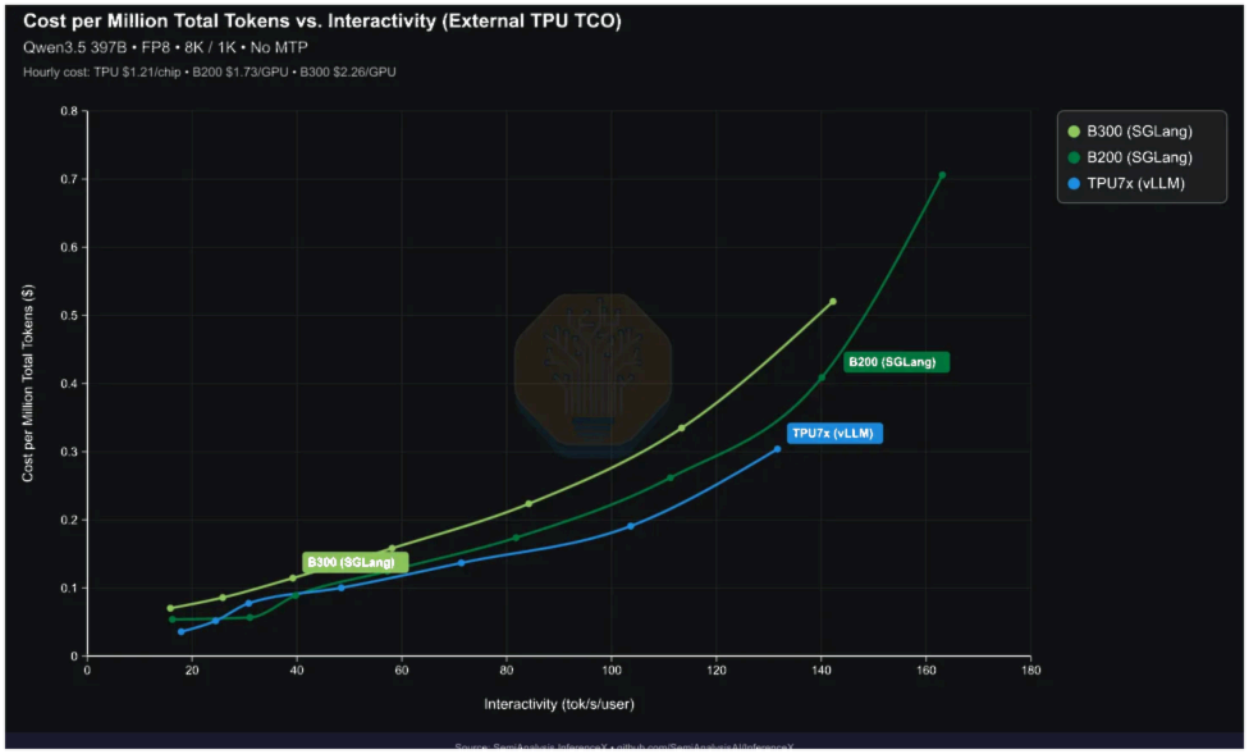
At an interactivity of 100 tokens/s/user, Ironwood costs approximately $0.181 per million total tokens, compared with $0.222 for B200 and $0.276 for B300. That is approximately 19% lower cost than B200 and 34% lower than B300, while delivering the same per-user generation speed.

在每用户每秒 100 个令牌的交互性下，Ironwood 每百万总令牌的成本约为 0.181 美元，而 B200 为 0.222 美元，B300 为 0.276 美元。这意味着成本比 B200 低约 19%，比 B300 低 34%，同时提供相同的每用户生成速度。

For this comparison, we use an external TPU TCO against B200/B300 TCO for an hyperscaler lab to buy TPU. Last year, SemiAnalysis Accelerator Model was the first to break that Google is selling TPUs instead of just renting out through Google Cloud. TPU's lower hourly cost offsets its lower throughput across much of the interactivity range. The SemiAnalysis TCO Model provides a full breakdown of the TPU BoM

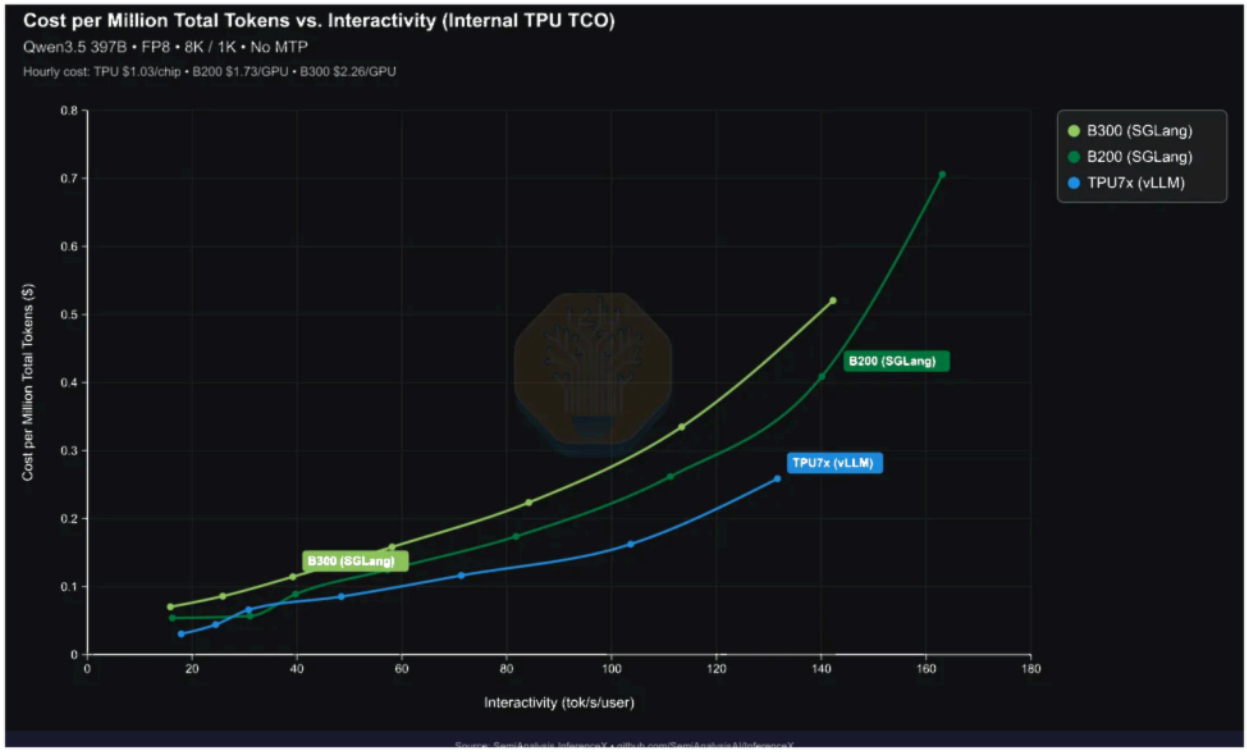
estimate and TCO estimates.

在此比较中，我们使用外部 TPU 的 TCO 与 B200/B300 的 TCO 进行对比，以评估超大规模实验室购买 TPU 的情况。去年，SemiAnalysis 加速器模型率先披露谷歌正在销售 TPU，而不仅仅是通过谷歌云出租。TPU 较低的每小时成本弥补了其在大部分交互性范围内吞吐量较低的问题。SemiAnalysis TCO 模型提供了 TPU 物料清单（BoM）估算和 TCO 估算的完整细分。



Source: SemiAnalysis InferenceX Preview

来源：SemiAnalysis InferenceX 预览



Source: InferenceX Preview

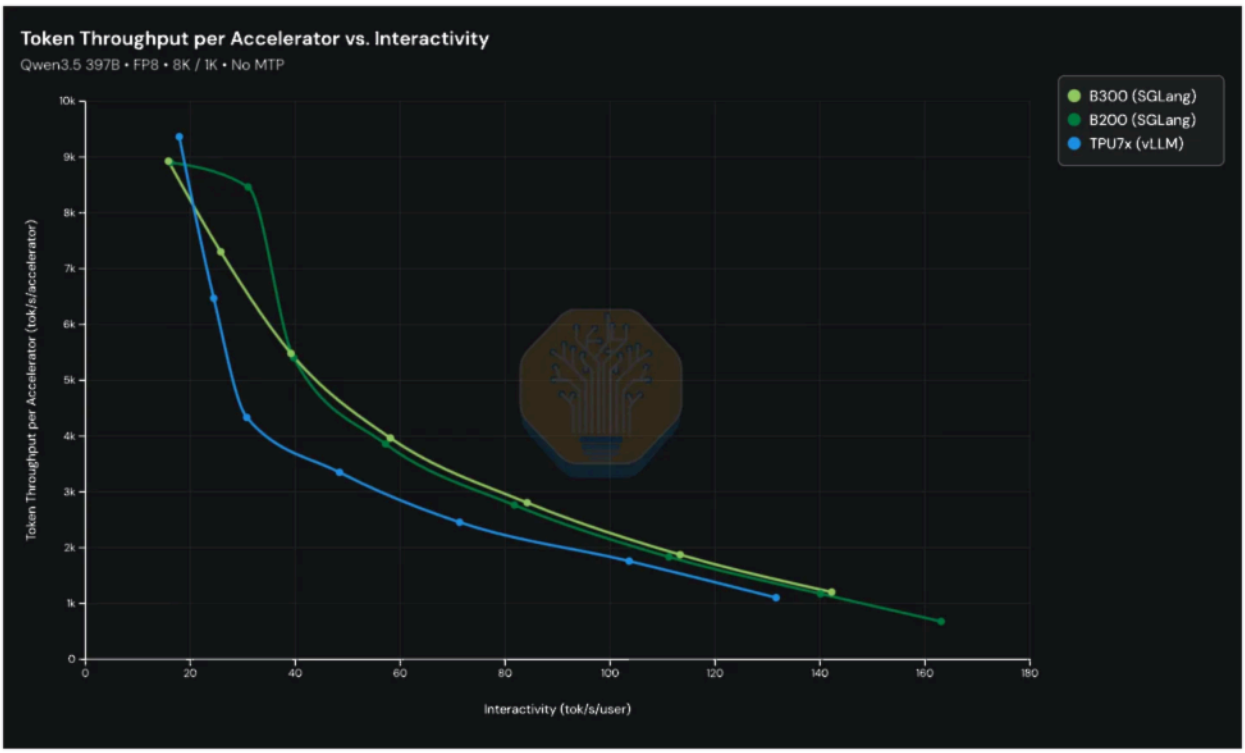
InferenceX 预览

Across most of the raw-performance curve, TPU does not outperform Nvidia GPUs. However, that comparison does not account for TPU's lower TCO. **Buyers of TPU are mainly concerned with how much revenue they can make, i.e. performance per dollar and performance per watt.**

在大部分原始性能曲线上，TPU 并未超越 Nvidia GPU。然而，这种比较并未考虑 TPU 更低的总体拥有成本（TCO）。TPU 的购买者主要关心他们能创造多少收入，即每美元性能和每瓦性能。

That being said, at interactivity of 20 tok/s/user, Ironwood also leads on raw throughput, reaching 9,364 total tokens/s/chip, compared with 8,903 on B200 and 8,925 on B300. This is approximately 5% higher throughput than either GPU in these runs. Combined with the lower modeled hourly cost, Ironwood delivers 50.4% more tokens per dollar than B200 and 96.0% more than B300.

话虽如此，在 20 tok/s/用户的交互性下，Ironwood 在原始吞吐量方面也领先，达到 9,364 总 token/s/芯片，而 B200 为 8,903，B300 为 8,925。这比这两种 GPU 在这些运行中的吞吐量高出约 5%。结合较低的建模每小时成本，Ironwood 每美元 token 数比 B200 多 50.4%，比 B300 多 96.0%。



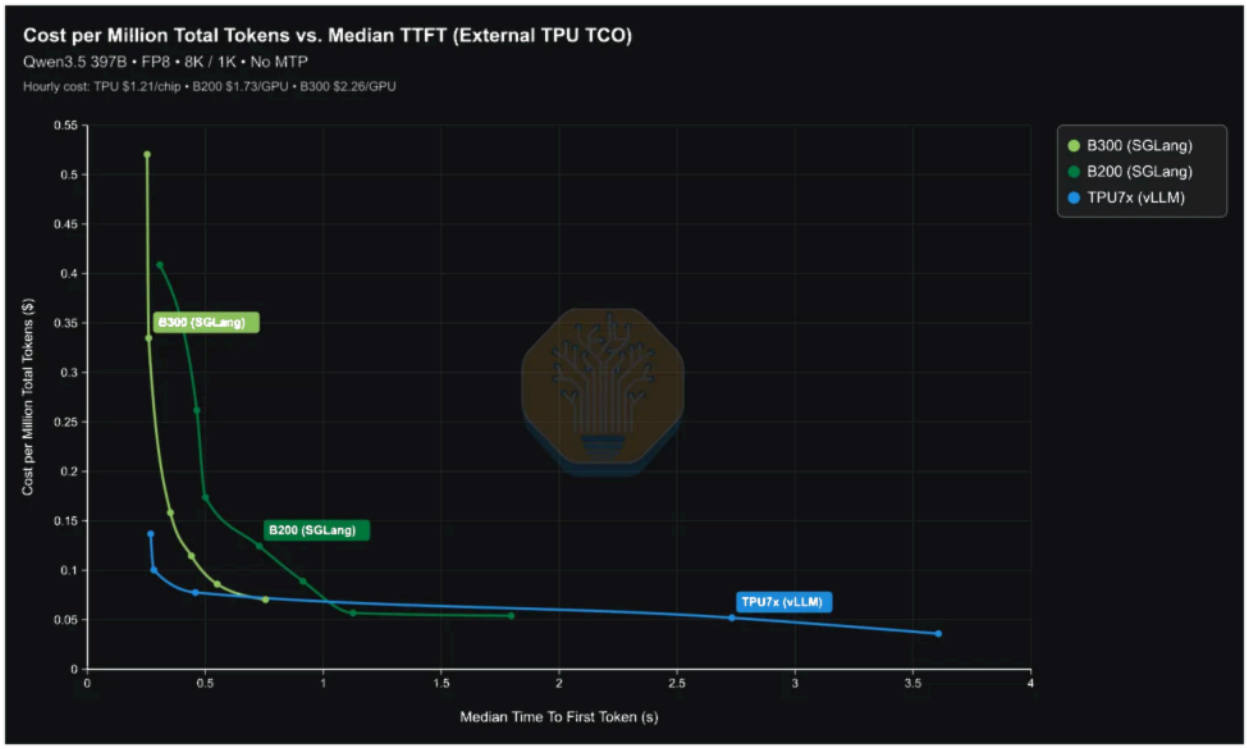
Source: SemiAnalysis InferenceX Preview

SemiAnalysis InferenceX 预览

If we consider TPU TCO for Google's internal workloads at $1.03/chip-hour, the performance-per-dollar advantage at concurrency 256 increases to 76.7% over B200 and 130.2% over B300. However, these high-concurrency results come with latency trade-offs. For example, at concurrency 256, TPU mean TTFT is 5.41 seconds, compared with 3.75 seconds on B200 and 2.40 seconds on B300. The previously

discussed 50% to 96% advantage applies to this datapoint specifically, rather than every latency target.

如果考虑 Google 内部工作负载的 TPU TCO 为 $1.03/芯片小时，则在并发度为 256 时，每美元性能优势相比 B200 提升 76.7%，相比 B300 提升 130.2%。然而，这些高并发结果伴随着延迟权衡。例如，在并发度为 256 时，TPU 的平均 TTFT 为 5.41 秒，而 B200 为 3.75 秒，B300 为 2.40 秒。之前讨论的 50% 至 96% 的优势适用于该特定数据点，而非每个延迟目标。

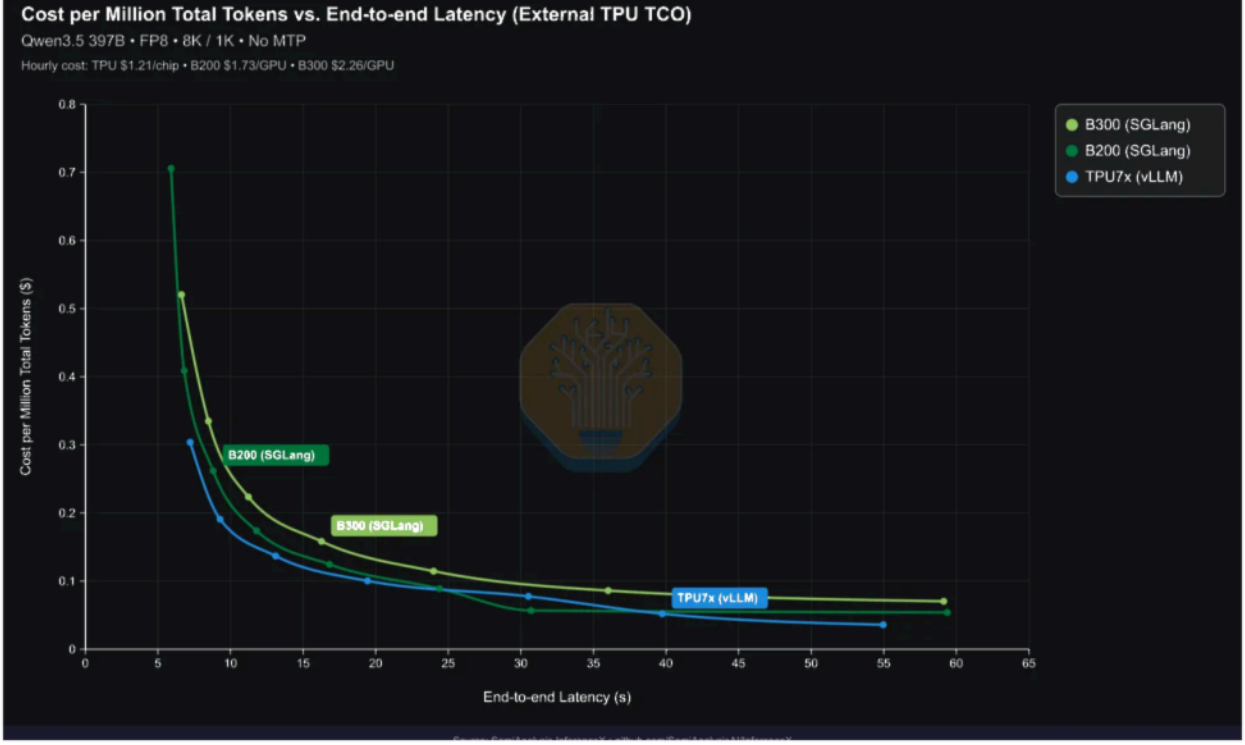


Source: SemiAnalysis InferenceX Preview

SemiAnalysis InferenceX 预览

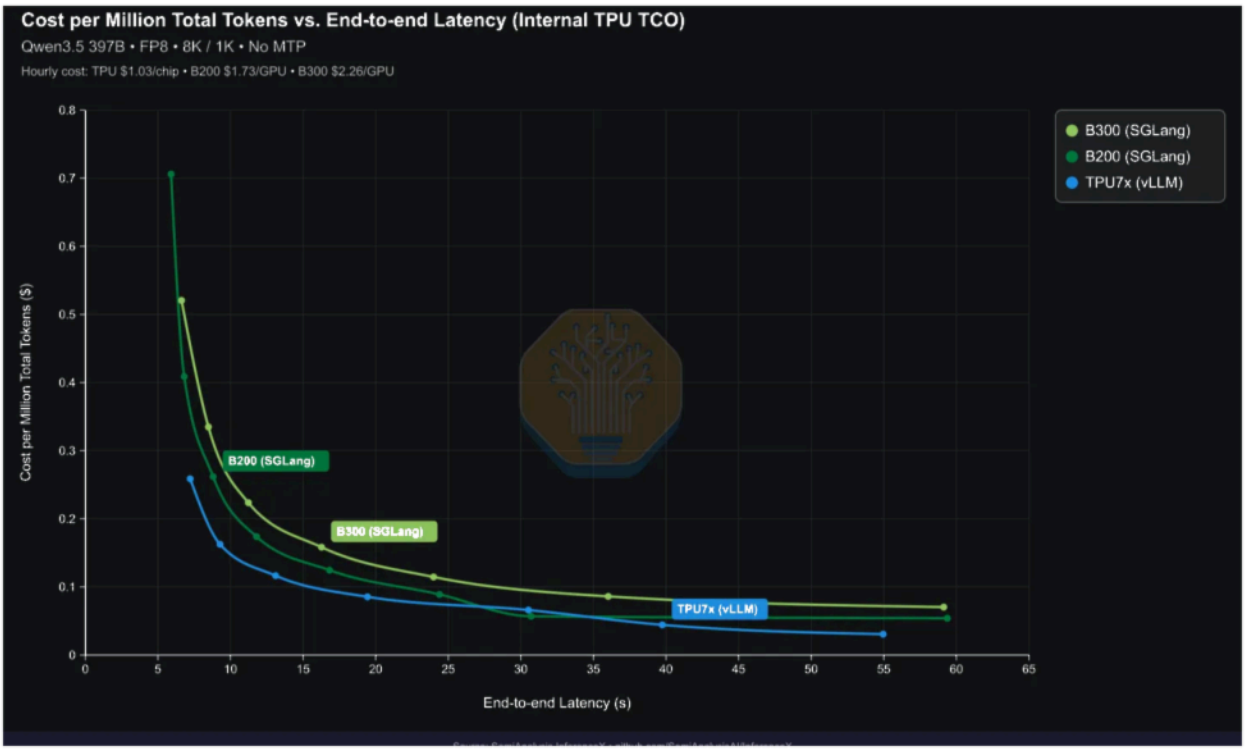
Looking at end-to-end latency, Ironwood remains competitive beyond that highest-throughput point. At a 20-second median response time, the Pareto curves put TPU at approximately $0.098 per million total tokens, compared with $0.106 for B200 and $0.132 for B300. TPU therefore costs approximately 8% less than B200 and 25% less than B300.

从端到端延迟来看，Ironwood 在最高吞吐量点之外仍保持竞争力。在 20 秒中位响应时间下，帕累托曲线显示 TPU 每百万总 token 成本约为 $0.098，而 B200 为 $0.106，B300 为 $0.132。因此，TPU 成本比 B200 低约 8%，比 B300 低 25%。



Source: SemiAnalysis InferenceX Preview

SemiAnalysis InferenceX 预览



Source: SemiAnalysis InferenceX Preview

SemiAnalysis InferenceX 预览

Despite all of the above, B200 can still come out ahead on a small part of the Apples to Apples curve, including around the 30-second median response time. However, TPU regains its lower cost per million tokens at longer response times, while maintaining a

cost advantage over B300 across the overlapping range shown here.

尽管存在上述种种情况，B200 在“苹果对苹果”对比曲线的一小部分上仍可能领先，包括大约 30 秒的中位响应时间。然而，在更长的响应时间下，TPU 重新获得了更低的每百万 token 成本，同时在所示的重叠范围内，相对于 B300 保持了成本优势。

SemiAnalysis is a reader-supported publication. To receive new posts and support our work, consider becoming a subscriber.

Apples to Bananas: GB300 NVL72 Disagg verus TPUv7 Agg

苹果对香蕉：GB300 NVL72 分离式架构对比 TPUv7 聚合式架构

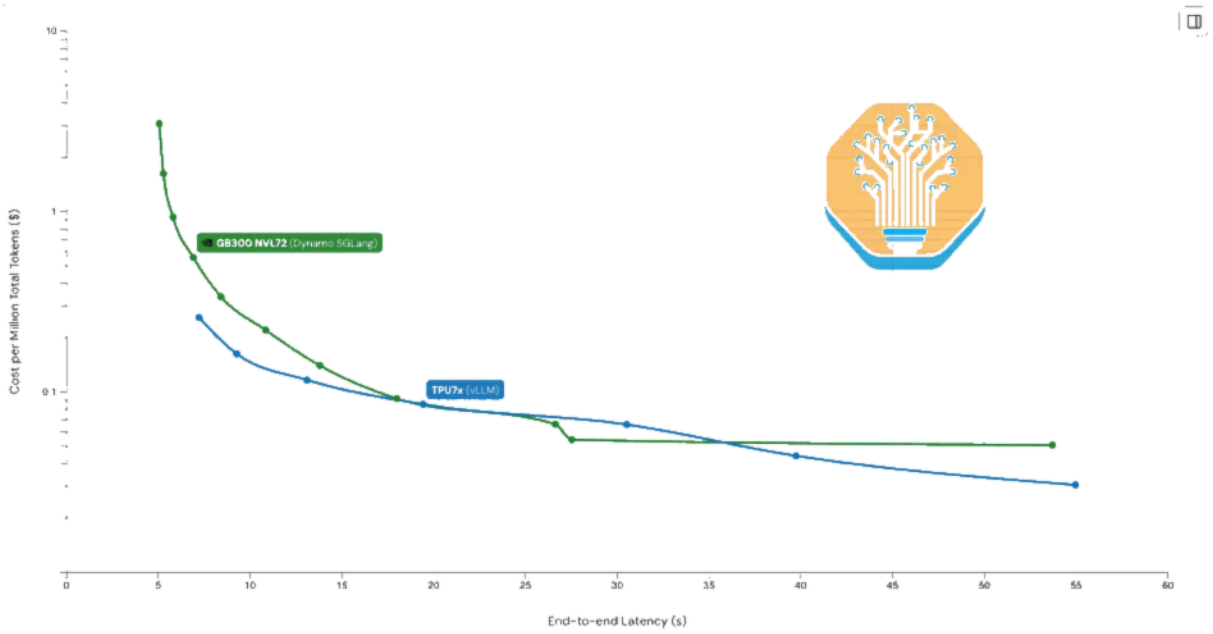
Google has been running disaggregated serving internally for years in production and that path is heavily optimized, but the external TPU serving stack does not have a fully optimized disagg path yet. So today, GB200/300 NVL72 is more competitive on perf per dollar in a disagg-vs-disagg comparison. But we expect that gap to close within a few months & TPUv7 to be competitive againist GB200/300 NVL72 as Google, Inferact, RadixArk, and SemiAnalysis land optimizations, and we will publish TPUv7 disagg vs GB200/300 NVL72 disagg in a follow-up article. TPUv7 pods can scale up to over 1k+ chips through their low latency ICI fabric thus can enable optimizations for large models disagg & ultra wideEP that NVIDIA NVL72 can’t perform.

Google 多年来一直在生产环境中内部运行分离式服务，该路径已高度优化，但外部 TPU 服务栈尚未拥有完全优化的分离式路径。因此，目前 GB200/300 NVL72 在分离式对分离式的比较中，在每美元性能方面更具竞争力。但我们预计这一差距将在几个月内缩小，随着 Google、Inferact、RadixArk 和 SemiAnalysis 落地各项优化，TPUv7 将再次与 GB200/300 NVL72 展开竞争，我们将在后续文章中发布 TPUv7 分离式与 GB200/300 NVL72 分离式的对比。TPUv7 Pod 可通过其低延迟 ICI 架构扩展到超过 1000+ 芯片，从而能够为大型模型的分离式服务和超宽 EP 实现优化，而这是 NVIDIA NVL72 无法做到的。

Even on an apples to bananas comparison between TPUv7 aggregated serving verus GB300 NVL72 disagg, using internal TCO, TPUv7 is decently competitive on high latency & low to left medium latency. But when comparing on the right medium e2e latency, GB300 NVL72 disagg has an single advantage against TPuv7 aggregrated

serving.

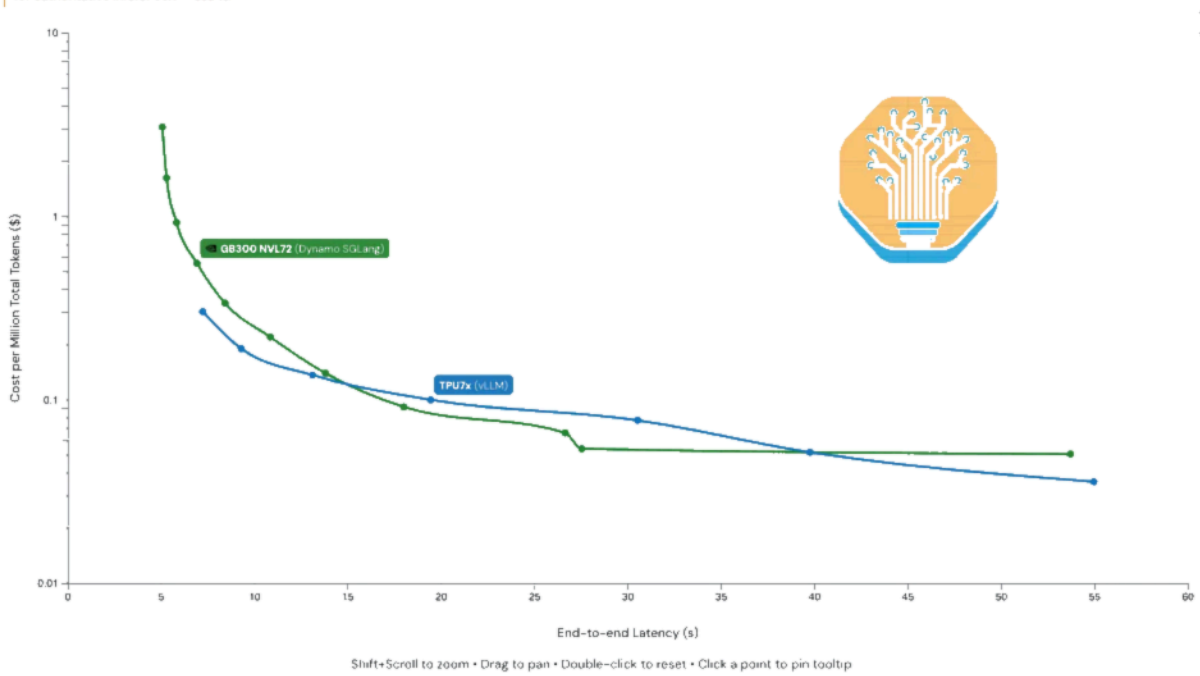
即使在 TPUv7 聚合式服务与 GB300 NVL72 分离式服务的“苹果对香蕉”对比中，使用内部 TCO 计算，TPUv7 在高延迟和低至左侧中等延迟区间也颇具竞争力。但在右侧中等端到端延迟的比较中，GB300 NVL72 分离式架构相对于 TPUv7 聚合式服务具有单一优势。



Source: SemiAnalysis 来源：SemiAnalysis

When looking at it GB300 NVL72 disagg verus TPUv7 agg, it is competitive at low and high e2e latency but in the middle, there is about an 30% perf per dollar advantage of GB300 against TPUv7 agg. We strongly believe once disaggregated serving on TPUv7 is optimized, it would be competitive across the full pareto against GB300 NVL72.

从表面来看，GB300 NVL72 的分离式架构与 TPUv7 的聚合式架构相比，在低端和高端端到端延迟方面具有竞争力，但在中间区间，GB300 相比 TPUv7 聚合式架构在每美元性能上约有 30% 的优势。我们坚信，一旦 TPUv7 上的分离式服务得到优化，它将在整个帕累托前沿上与 GB300 NVL72 展开全面竞争。



Source: SemiAnalysis 来源：SemiAnalysis

Inference Serving with the New Native, First-Class TorchTPU Backend vs. the Previous TorchAX Stack

使用新的原生一流 TorchTPU 后端进行推理服务，对比之前的 TorchAX 栈

Previous TorchAX vLLM Backend

之前的 TorchAX vLLM 后端

Google previously used the TorchAX backend to translate the PyTorch-based implementations in vLLM and SGLang into JAX. TorchAX was intended to let developers run PyTorch model definitions through JAX. However, issues with this approach have prompted Google and the PyTorch, vLLM, and SGLang communities to collaborate on a new approach called TorchTPU. It lets developers use TPUs as native PyTorch devices, while StableHLO, XLA, and TPU-optimized Pallas kernels handle the low-level execution behind the scenes. In the upcoming sections of the article, we will be showing the performance of this new TorchTPU backend compared to

NVIDIA Blackwell & Blackwell Ultra GPUs.

Google 此前使用 TorchAX 后端将 vLLM 和 SGLang 中基于 PyTorch 的实现转换为 JAX。TorchAX 旨在让开发者通过 JAX 运行 PyTorch 模型定义。然而，这种方法存在的问题促使 Google 与 PyTorch、vLLM 和 SGLang 社区合作，共同开发一种名为 TorchTPU 的新方法。它让开发者能够将 TPU 用作原生 PyTorch 设备，而 StableHLO、XLA 和 TPU 优化的 Pallas 内核则在后台处理底层执行。在本文接下来的部分，我们将展示这种新的 TorchTPU 后端与 NVIDIA Blackwell 和 Blackwell Ultra GPU 相比的性能表现。

vLLM TPU support has evolved through three different stages:

vLLM 的 TPU 支持经历了三个不同的阶段：

- Its first TPU prototype used PyTorch/XLA. Its lazy execution model collected operations into computation graphs for XLA to compile, rather than executing each PyTorch operation immediately.

  其首个 TPU 原型使用了 PyTorch/XLA。其惰性执行模型将操作收集到计算图中供 XLA 编译，而不是立即执行每个 PyTorch 操作。

- The current public backend, tpu-inference, uses a TPU-optimized JAX implementation when one is available. Otherwise, TorchAX translates the original PyTorch model into operations that JAX can execute. The goal is to reuse mature TPU primitives and parallelism support without requiring every PyTorch model to be rewritten.

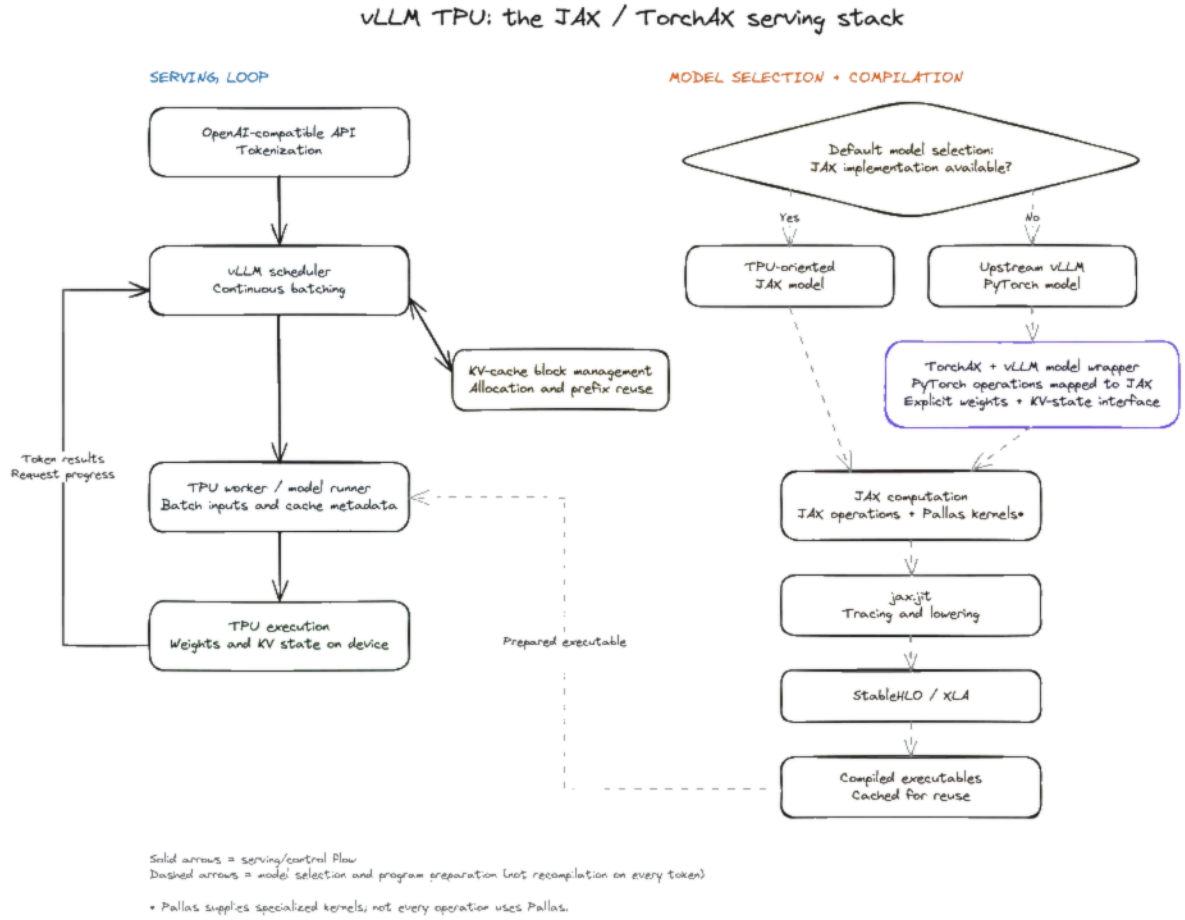
  当前公开的后端 tpu-inference，在可用时使用 TPU 优化的 JAX 实现。否则，TorchAX 会将原始 PyTorch 模型转换为 JAX 可以执行的操作。其目标是复用成熟的 TPU 原语和并行支持，而无需重写每个 PyTorch 模型。

- The upcoming third design, TorchTPU, lets vLLM treat TPUs as native PyTorch devices. It will be the go-to backend for SGLang and vLLM on TPUs going forward.

  即将推出的第三代设计 TorchTPU，让 vLLM 将 TPU 视为原生 PyTorch 设备。未来，它将成为 SGLang 和 vLLM 在 TPU 上的首选后端。

The diagram below zooms in on the second stage and shows how vLLM's tpu-inference backend combines direct JAX models and PyTorch models running through TorchAX.

下图放大了第二阶段，展示了 vLLM 的 tpu-inference 后端如何将直接 JAX 模型与通过 TorchAX 运行的 PyTorch 模型相结合。



Source: SemiAnalysis/vLLM

来源：SemiAnalysis/vLLM

The original choice of JAX was pragmatic. In the last announcement, Google described difficulties with low-level optimization, paged attention, and fitting vLLM's worker model to TPU execution. They thought that JAX would offer more mature TPU primitives and parallelism support. TorchAX let the team use those capabilities without rewriting every PyTorch model, while TPU-oriented JAX implementations could share the same kernel and compiler work.

最初选择 JAX 是出于务实考虑。在上一份公告中，Google 描述了在底层优化、分页注意力以及将 vLLM 的工作者模型适配到 TPU 执行方面遇到的困难。他们认为 JAX 能提供更成熟的 TPU 原语和并行支持。TorchAX 使团队无需重写每个 PyTorch 模型即可利用这些能力，同时面向 TPU 的 JAX 实现可以共享相同的内核和编译器工作。

With TorchAX, developers still write the model in PyTorch, but JAX executes it on the TPU. TorchAX acts as the translation layer between the two frameworks. As the TorchAX documentation explains, a torchax.tensor.Tensor behaves like an o PyTorch tensor but is backed by a jax.Array. When the model invokes a tens operation, known in PyTorch as an ATen operation, TorchAX intercepts it t __torch_dispatch__ hook and translates it into one or more JAX operations. model can therefore remain written in PyTorch even though JAX performs t



computation.

借助 TorchAX，开发者仍然使用 PyTorch 编写模型，但由 JAX 在 TPU 上执行。TorchAX 充当两个框架之间的转换层。正如 TorchAX 文档所解释的，torchax.tensor.Tensor 的行为类似于普通的 PyTorch 张量，但其底层由 jax.Array 支持。当模型调用张量操作（在 PyTorch 中称为 ATen 操作）时，TorchAX 通过 __torch_dispatch__钩子拦截该操作，并将其转换为一个或多个 JAX 操作。因此，即使计算由 JAX 执行，模型仍可保持以 PyTorch 编写。

vLLM must also manage state that changes during generation, especially the KV cache, which stores attention information from earlier tokens for reuse. Its model wrapper presents the model weights and KV cache to JAX as an explicit state. This process, known as functionalization, turns changes that would otherwise happen inside Python objects into inputs and outputs that the compiler can track. jax.jit can then capture each inference step as a computation graph and pass it through JAX's compilation pipeline for repeated execution.

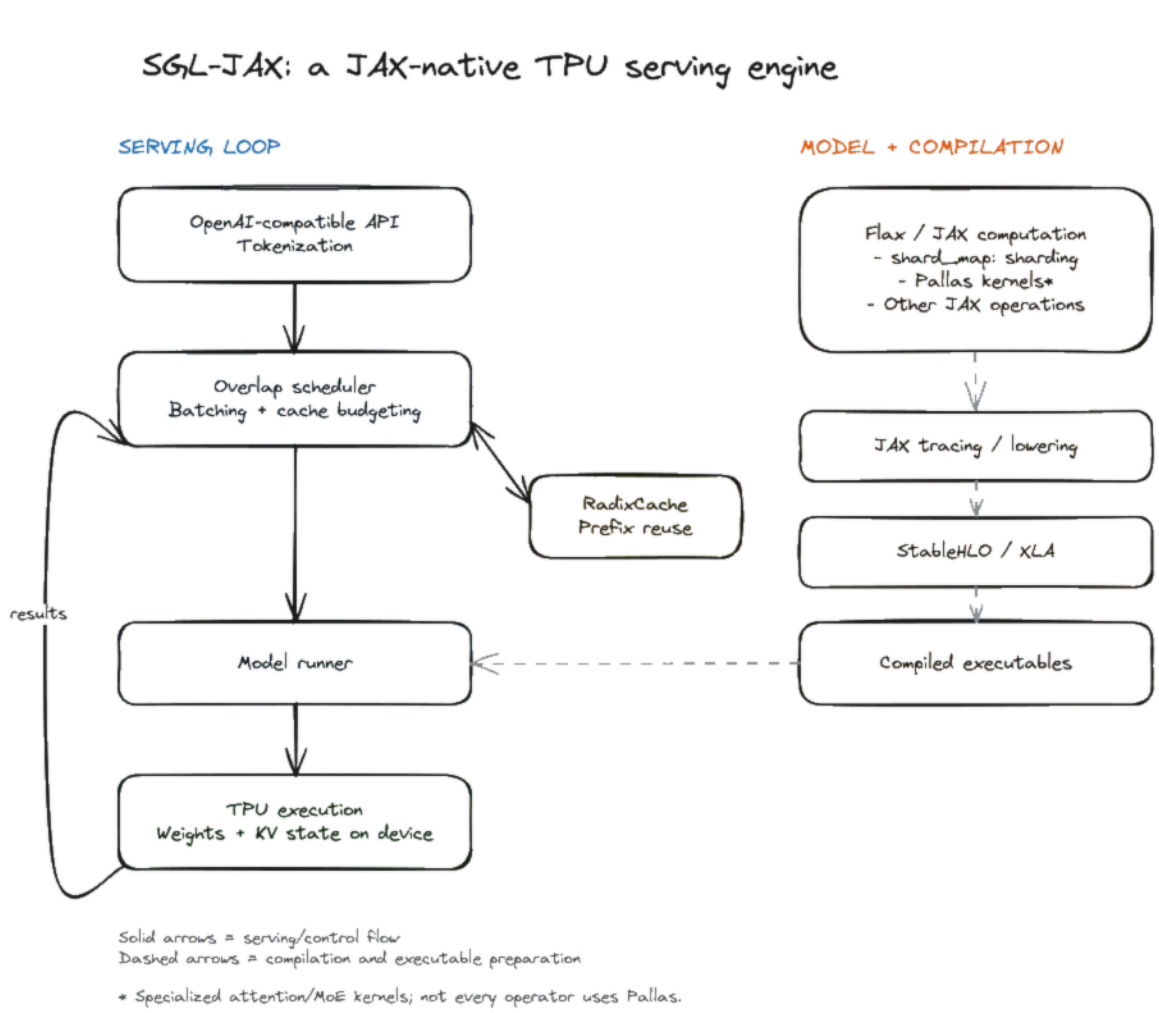
vLLM 还必须管理生成过程中变化的状态，尤其是 KV 缓存，它存储早期 token 的注意力信息以供重用。其模型包装器将模型权重和 KV 缓存作为显式状态呈现给 JAX。这个过程称为功能化，它将原本会在 Python 对象内部发生的变化转换为编译器可以跟踪的输入和输出。然后，jax.jit 可以将每个推理步骤捕获为计算图，并通过 JAX 的编译流水线进行重复执行。

From there, each component has a distinct role. Pallas supplies optimized TPU kernels for performance-critical operations. StableHLO represents the program in a standardized format that the compiler can process, and XLA converts it into executable code for the TPU. In vLLM's standard PyTorch path, torch.compile coordinates graph capture and compilation. In the TorchAX path, the JAX/XLA pipeline already performs those tasks, so vLLM bypasses its usual torch.compile route. These translation layers have caused many issues that the TorchTPU vLLM stack aims to solve.

从这里开始，每个组件都有其独特的角色。Pallas 为性能关键的操作提供优化的 TPU 内核。StableHLO 以编译器可以处理的标准格式表示程序，而 XLA 将其转换为可在 TPU 上执行的代码。在 vLLM 标准的 PyTorch 路径中，torch.compile 协调图捕获和编译。在 TorchAX 路径中，JAX/XLA 流水线已经执行了这些任务，因此 vLLM 绕过了其通常的 torch.compile 路线。这些转换层引发了许多问题，而 TorchTPU vLLM 技术栈旨在解决这些问题。

Previous SGLang JAX Backend

之前的 SGLang JAX 后端



Source: SemiAnalysis/SGLang

来源：SemiAnalysis/SGLang

SGLang's current public TPU stack uses SGLang-JAX. SGLang-JAX makes a similar trade-off to vLLM's TorchAX backend but takes a different approach. It is a separate JAX-native serving engine rather than the PyTorch SGLang runtime translated through TorchAX. It combines SGLang-style scheduling and prefix caching with JAX models and TPU-specific kernels. Google and RadixArk have announced SGL-torchtpu as an additional PyTorch-native backend to extend the available serving paths.

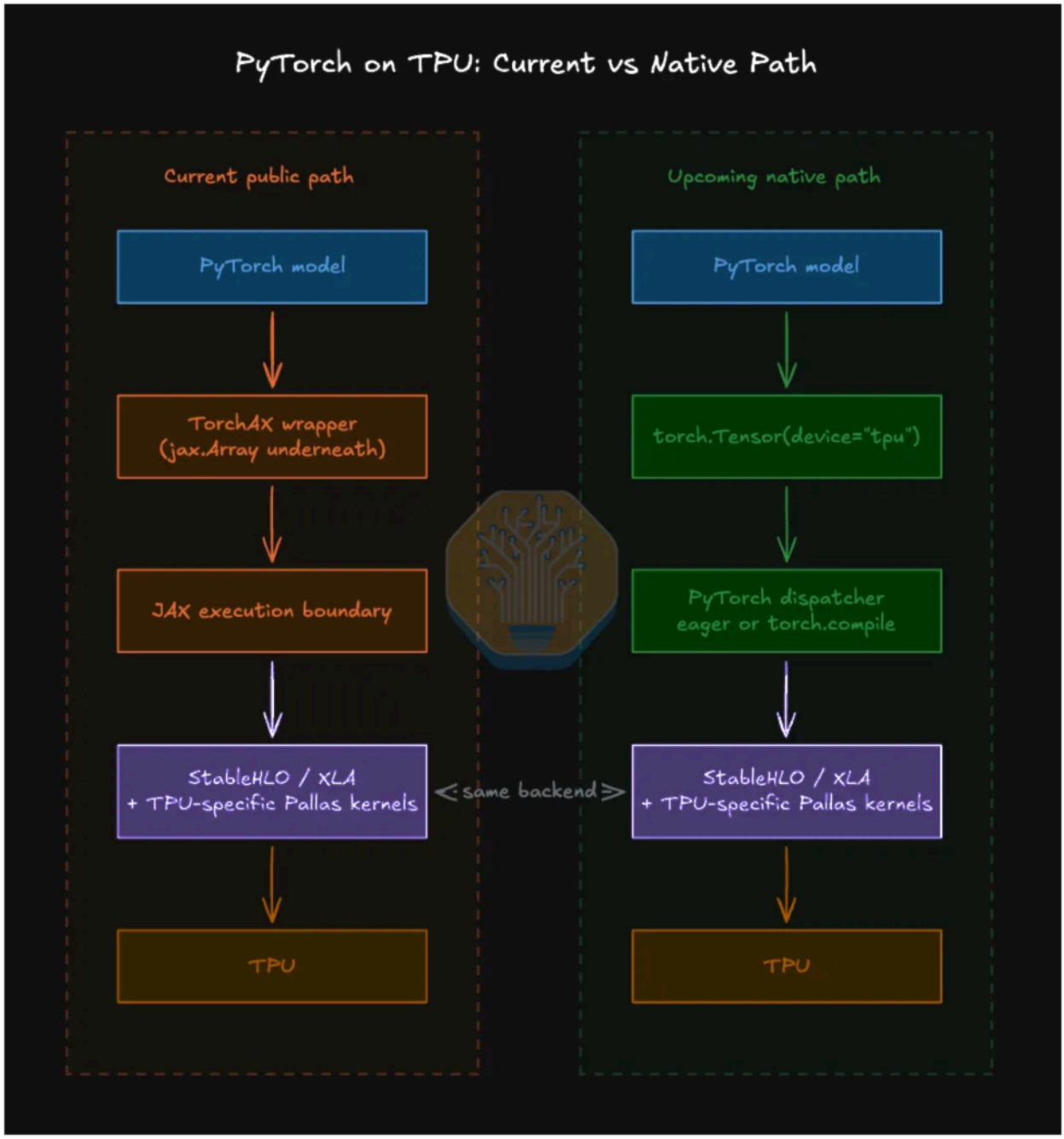
SGLang 当前公开的 TPU 技术栈使用 SGLang-JAX。SGLang-JAX 与 vLLM 的 TorchAX 后端做出了类似的权衡，但采取了不同的方法。它是一个独立的 JAX 原生服务引擎，而不是通过 TorchAX 转换的 PyTorch SGLang 运行时。它将 SGLang 风格的调度和前缀缓存与 JAX 模型及 TPU 特定内核相结合。谷歌和 RadixArk 已宣布 SGL-torchtpu 作为额外的 PyTorch 原生后端，以扩展可用的服务路径。

Shifting Toward the New Native, First-Class TorchTPU Backend

转向新的原生、一流的 TorchTPU 后端

TorchTPU moves the framework boundary into PyTorch itself. It uses PyTorch's PrivateUse1 backend extension point to expose an ordinary torch.Tensor on device="tpu" rather than a wrapper backed by a JAX array. The PyTorch dispatcher routes ATen operations to the TPU backend. Developers can run code eagerly for bring-up and debugging, or call torch.compile for graph capture. In the compiled path, TorchDynamo and AOTAutograd produce an FX graph, TorchTPU lowers it to StableHLO, and XLA produces the TPU executable. Google's conference deck makes the compiler choice explicit that this path uses XLA, not Inductor and Triton.

TorchTPU 将框架边界移入 PyTorch 自身。它利用 PyTorch 的 PrivateUse1 后端扩展点，在 device="tpu"上暴露一个普通的 torch.Tensor，而不是由 JAX 数组支持的包装器。PyTorch 调度器将 ATen 操作路由到 TPU 后端。开发者可以急切地运行代码以进行启动和调试，或调用 torch.compile 进行图捕获。在编译路径中，TorchDynamo 和 AOTAutograd 生成 FX 图，TorchTPU 将其降级为 StableHLO，XLA 则生成 TPU 可执行文件。谷歌的会议演示文稿明确指出了编译器选择，即此路径使用 XLA，而非 Inductor 和 Triton。



"Native" therefore has a specific scope. It covers PyTorch tensors, dispatch, eager execution, compilation entrypoints, and distributed APIs. XLA remains the compiler, and the kernel layer continues to use TPU-specific implementations. TorchTPU can call Pallas and JAX-backed custom kernels. Helion's TPU backend also emits Pallas. The user-facing framework becomes PyTorch-native, while the compiler and performance-critical kernels remain TPU-aware.

因此，"原生"具有特定的范围。它涵盖 PyTorch 张量、调度、即时执行、编译入口和分布式 API。XLA 仍是编译器，内核层继续使用 TPU 专用实现。TorchTPU 可以调用 Pallas 和基于 JAX 的自定义内核。Helion 的 TPU 后端也生成 Pallas。面向用户的框架变为 PyTorch 原生，而编译器和性能关键内核仍保持 TPU 感知。

Developers can use .to("tpu") while retaining familiar distributed interfaces and serving-engine code. Google documents support for DDP, FSDP2, DTensor, one process per device, and both MPMD and SPMD execution. Those interfaces fit the process and distributed model already assumed by PyTorch serving engines.

开发者可以使用 .to("tpu") 同时保留熟悉的分布式接口和服务引擎代码。谷歌文档支持 DDP、FSDP2、DTensor、每设备单进程，以及 MPMD 和 SPMD 执行。这些接口符合 PyTorch 服务引擎已采用的进程和分布式模型。

For vLLM and SGLang, the opportunity is to reuse more upstream model code, schedulers, continuous batching, APIs, and feature logic instead of rebuilding them across a PyTorch-to-JAX boundary. That should reduce the cost of bringing up models and engine features, even when the final TPU implementation still needs specialized kernels.

对于 vLLM 和 SGLang 而言，机会在于复用更多上游模型代码、调度器、连续批处理、API 和功能逻辑，而不是在 PyTorch 到 JAX 的边界上重新构建它们。这应能降低模型和引擎功能的引入成本，即使最终的 TPU 实现仍需要专用内核。

Native PyTorch support does not eliminate the need for TPU-specific optimization. Pallas and TorchAX serve different purposes, TorchAX allows PyTorch model code to execute through JAX, while Pallas is used to implement performance critical operations directly for TPU hardware. Removing TorchAX doesn't mean removing the Pallas kernels underneath it. Since TorchTPU allows native Pytorch code to invoke Pallas and JAX kernels, kernels developed for earlier stack are able to migrate into the

new stack.

原生 PyTorch 支持并不能消除对 TPU 特定优化的需求。Pallas 和 TorchAX 服务于不同的目的，TorchAX 允许 PyTorch 模型代码通过 JAX 执行，而 Pallas 则用于直接在 TPU 硬件上实现性能关键操作。移除 TorchAX 并不意味着移除其底层的 Pallas 内核。由于 TorchTPU 允许原生 PyTorch 代码调用 Pallas 和 JAX 内核，为早期技术栈开发的内核能够迁移到新技术栈中。

Similar to GPU PyTorch vLLM, Custom kernels will still be needed for high performance inferencing. Tensor shapes and layouts still need to be tuned to use TPU's matrix multiplication units efficiently. We believe that most of the existing Pallas kernels are easily transferable to the new native TorchTPU serving stack but that does not mean that every existing kernel will transfer unchanged. Many things like wrappers, tensor layouts, and runtime integration etc may still need to be slightly adapted and validated.

与 GPU 上的 PyTorch vLLM 类似，高性能推理仍然需要自定义内核。张量形状和布局仍需调整，以高效利用 TPU 的矩阵乘法单元。我们认为，现有的大多数 Pallas 内核可以轻松迁移到新的原生 TorchTPU 服务技术栈中，但这并不意味着每个现有内核都能原封不动地迁移。诸如包装器、张量布局和运行时集成等许多方面可能仍需要稍作调整和验证。

Strong Ecosystem Collaboration with Google's TPU Team

与谷歌 TPU 团队的强大生态合作

vLLM and SGLang are the two major open-source production inference serving stacks. Inferact, RadixArk, and Red Hat are investing heavily in collaboration with Google to make the TorchTPU backend a first-class experience in both stacks. This community support is extremely important for externalizing TPU inference.

vLLM 和 SGLang 是两大主流的开源生产级推理服务栈。Inferact、RadixArk 和 Red Hat 正大力投入与 Google 的合作，致力于让 TorchTPU 后端在这两个栈中成为一流体验。这种社区支持对于实现 TPU 推理外部化至关重要。



PREVIEW

A New Unified Backend Supporting PyTorch and JAX on TPU

Source: Inferact 来源：Inferact

As of early September, TorchTPU remains in private beta and will be open-sourced around mid-October during the PyTorch Conference. We are excited to see it move TPUs toward a first-class experience in PyTorch, vLLM, and SGLang. Once TorchTPU vLLM and TorchTPU SGLang are open-sourced, we will move TPU benchmarking for InferenceX/AgentX from our fork to our public repo.

截至九月初，TorchTPU 仍处于私有测试阶段，并将在十月中旬的 PyTorch 大会期间开源。我们很高兴看到它推动 TPU 在 PyTorch、vLLM 和 SGLang 中成为一流体验。一旦 TorchTPU vLLM 和 TorchTPU SGLang 开源，我们将把 InferenceX/AgentX 的 TPU 基准测试从我们的分支迁移到公共仓库中。



together contributing to



Source: RadixArk 来源：RadixArk

TPU Inference Optimizations Deep Dive

TPU 推理优化深度解析

The sprint to optimize TPU inference for the native TorchTPU serving stack's first bring-up model involved hundreds of engineering hours and numerous PRs. As described later in the article, TPU architecture differs from GPU architecture and therefore requires its own set of optimizations to run models efficiently. Below, we describe the optimizations that improved (or in some cases, enabled) performance for Qwen3.5-397B and other models on Ironwood. Most of the Pallas kernel optimizations are transferable to the new native TorchTPU stack.

为原生 TorchTPU 服务栈的首个启动模型优化 TPU 推理的冲刺工作，投入了数百小时的工程时间和大量 PR。正如文章后续所述，TPU 架构与 GPU 架构不同，因此需要一套自己的优化方案才能高效运行模型。下面，我们描述了在 Ironwood 上提升（或在某些情况下实现）Qwen3.5-397B 及其他模型性能的优化措施。大多数 Pallas 内核优化可迁移到新的原生 TorchTPU 栈中。

DP Attention Optimization
DP 注意力优化

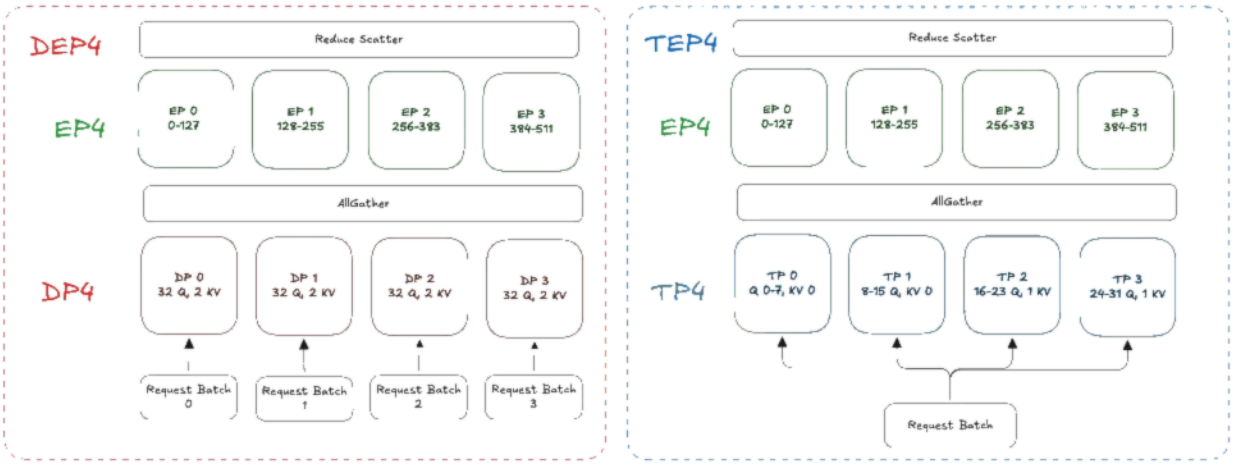
One issue that is not necessarily specific to TPU is how to effectively map parallelism from model architecture to hardware. For instance, Qwen3.5's GQA attention layer has 32 query heads but only 2 shared KV heads. With TP8 (eight logical devices), query computation can be evenly distributed across the 8 devices, with 4 query heads per device. The KV heads, however, do not divide evenly across those devices. Each KV head is shared by 16 query heads, meaning four devices need the same KV data to compute their local attention. When TP exceeds the KV-head count, standard practice is to replicate KV heads across TP ranks. vLLM already supports this. The TPU backend needed a compatibility fix to activate that existing behavior and avoid unnecessary All-to-All communication.

一个并非 TPU 特有的问题是，如何有效地将模型架构中的并行性映射到硬件上。例如，Qwen3.5 的 GQA 注意力层有 32 个查询头，但只有 2 个共享的 KV 头。在 TP8（八个逻辑设备）下，查询计算可以均匀分布到 8 个设备上，每个设备处理 4 个查询头。然而，KV 头无法在这些设备上均匀分配。每个 KV 头由 16 个查询头共享，这意味着四个设备需要相同的 KV 数据来计算其本地注意力。当 TP 规模超过 KV 头数量时，标准做法是在 TP 各等级间复制 KV 头。vLLM 已经支持这一点。TPU 后端需要进行兼容性修复，以激活这一现有行为，避免不必要的 All-to-All 通信。

For higher-concurrency serving, the TPU backend also added support for combining eight-way attention data parallelism (DP8) with eight-way expert parallelism (EP8), simply referred to as "DP attention" or DEP8. Instead of splitting every request's attention across all eight devices, each device processes a different subset of requests and keeps both KV heads for those requests locally. The attention weights are replicated, but the KV caches hold different request histories rather than repeated

copies of the same history. Meanwhile, the 512 routed experts remain distributed across the devices, avoiding replication of the much larger expert weight pool. Making this work required coordinating request assignment, recurrent-state slots, and block tables in the serving engine.

为了支持更高并发度的服务，TPU 后端还增加了将八路注意力数据并行（DP8）与八路专家并行（EP8）相结合的支持，简称为“DP 注意力”或 DEP8。并非将每个请求的注意力计算分散到全部八个设备上，而是每个设备处理不同的请求子集，并在本地保留这些请求的 KV 头。注意力权重是复制的，但 KV 缓存中保存的是不同的请求历史，而非同一历史的重复副本。与此同时，512 个路由专家仍分布在各个设备上，避免复制规模更大的专家权重池。实现这一功能需要在服务引擎中协调请求分配、循环状态槽和块表。



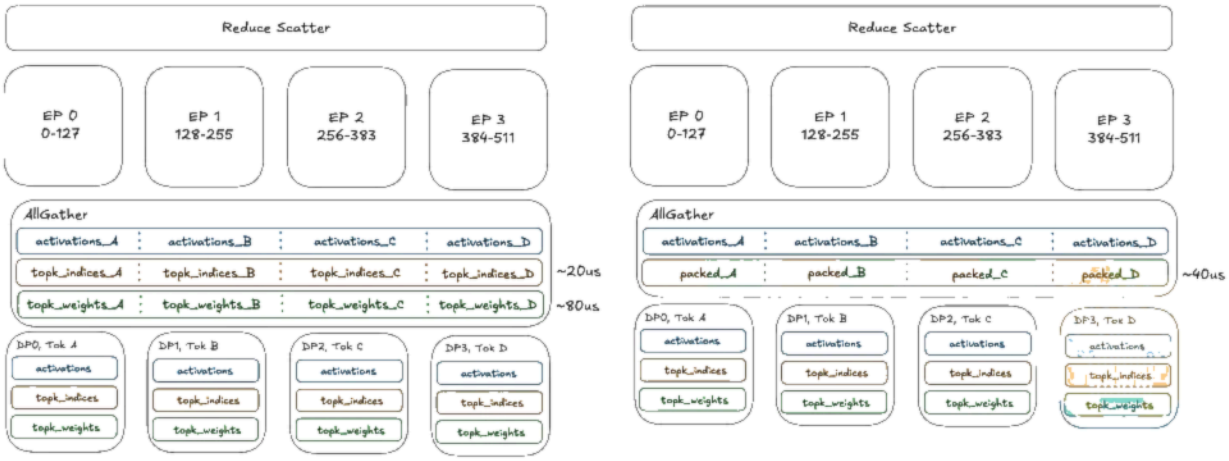
Source: SemiAnalysis 来源：SemiAnalysis

Optimizing Communications
优化通信

With DP attention and EP, the TPU GroupedGEMM implementation all-gathers token activations and routing metadata before the experts run, then uses reduce-scatter to sum their weighted outputs and return each token's result to its attention rank. The backend originally gathered the selected expert IDs and routing weights separately. Google combined these into one all-gather, removing an extra collective whose latency can dominate the transfer time for such small arrays. The PR reports roughly 80 microseconds per layer saved in its DeepSeek-V3 measurements. While 80 µs may sound small, DeepSeek-V3/R1 has 58 layers that require this AllGather. Across those layers, the savings add up to around 4.64 ms per forward pass in isolation (80 µs ×

58), reducing TPOT and increasing interactivity.

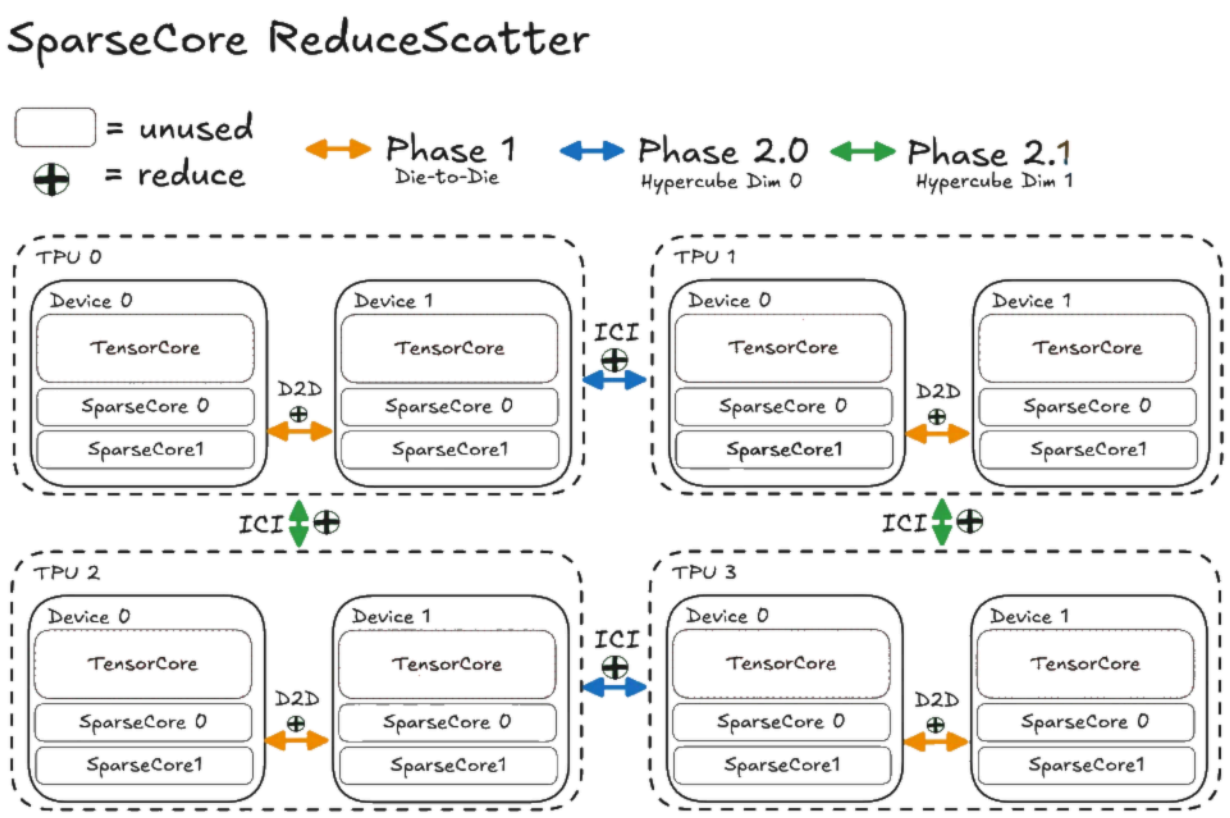
通过 DP 注意力和 EP，TPU GroupedGEMM 实现在专家运行之前先对 token 激活和路由元数据执行 all-gather，然后使用 reduce-scatter 对加权输出求和，并将每个 token 的结果返回给其注意力 rank。后端最初分别收集选定的专家 ID 和路由权重。Google 将这两者合并为一次 all-gather，省去了一次额外的集合通信，而对于此类小数组，该集合通信的延迟可能主导传输时间。该 PR 报告称，在其 DeepSeek-V3 测量中，每层节省约 80 微秒。虽然 80 微秒听起来可能很小，但 DeepSeek-V3/R1 有 58 层需要此 AllGather。跨这些层累计，单独计算时每次前向传播可节省约 4.64 毫秒（80 微秒×58），从而降低 TPOT 并提高交互性。



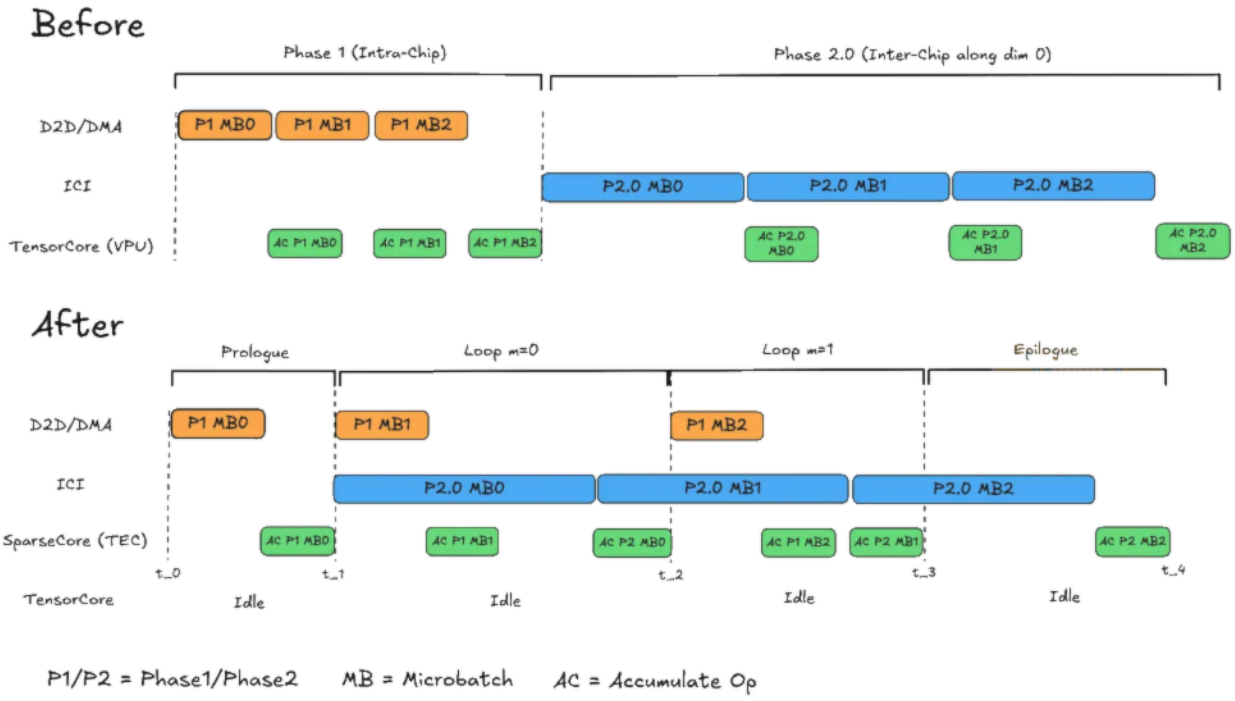
Source: SemiAnalysis 来源：SemiAnalysis

Google also [implemented the ReduceScatter collective on SparseCore] (which is better suited for irregular operations like data movement) and used Ironwood's faster die-to-die links to combine contributions within each chip before exchanging partial sums across chips. This two-device-per-chip layout and the chip-to-chip ICI network are covered in the next section. Double buffering lets the kernel transfer one chunk while accumulating another, overlapping local reductions and die-to-die transfers with the slower chip-to-chip traffic. Running the collective on SparseCore also frees TensorCore execution for other operations.

Google 还在 SparseCore 上实现了 ReduceScatter 集合通信（SparseCore 更适合处理数据移动等不规则操作），并利用 Ironwood 更快的 die-to-die 链路，先在每个芯片内合并贡献，再跨芯片交换部分和。这种每芯片双设备布局以及芯片间 ICI 网络将在下一节介绍。双缓冲允许内核在累加一个数据块的同时传输另一个数据块，将本地归约和 die-to-die 传输与较慢的芯片间通信重叠执行。在 SparseCore 上运行集合通信还释放了 TensorCore，使其可以执行其他操作。



Source: SemiAnalysis 来源：SemiAnalysis



Source: Google and SemiAnalysis

来源：Google 和 SemiAnalysis

In the diagram above, communication stages overlap, shortening end-to-end execution time. For instance, consider the point labeled t_1. At this time, the intra-chip DMA ScatterReduce (P1) has completed for MB (microbatch) 1. ScatterReduce across ICI dim 0 (P2.0) can now occur for MB0 *concurrently with* P1 for the subsequent MB. Against a baseline, this optimization results in [4.1 to 14.2% higher throughput on 8k1k](#) across concurrency 64 to 512, including 8.5% at concurrency 256, and 26.1% on 1k8k at

concurrency 512.

在上图中，通信阶段相互重叠，缩短了端到端的执行时间。例如，考虑标记为 t_1 的时间点。此时，芯片内的 DMA ScatterReduce（P1）已完成对 MB（微批次）1 的处理。现在，针对 MB0 的跨 ICI 维度 0 的 ScatterReduce（P2.0）可以与后续 MB 的 P1 同时进行。与基线相比，此优化在 8k1k 配置下、并发度从 64 到 512 的范围内，吞吐量提升了 4.1% 至 14.2%，其中在并发度为 256 时提升 8.5%；在 1k8k 配置下、并发度为 512 时，吞吐量提升了 26.1%。

Thanks for reading SemiAnalysis! This post
is public so feel free to share it.

The movement of this collective from TensorCore to SparseCore also means that additional work can be pipelined on the TensorCore while the collective runs. However, not every collective is always worth moving. In some cases, offloading a collective to SparseCores actually *worsens* performance. For example, for Qwen3.5 a threshold now decides when all-reduce and all-gather operations are offloaded to the SparseCore, **keeping small collectives on the TensorCore when they fit in VMEM and SparseCore offload overhead would make them slower**. The default threshold is derived from VMEM capacity. The PR reports 8k1k throughput gains of 2.7% at concurrency 64 and 5.7% at concurrency 128.

将该集合通信从 TensorCore 迁移到 SparseCore 也意味着，在集合通信运行期间，TensorCore 上可以流水线处理额外的工作。然而，并非每个集合通信都值得迁移。在某些情况下，将集合通信卸载到 SparseCore 实际上会降低性能。例如，对于 Qwen3.5，现在通过一个阈值来决定何时将 all-reduce 和 all-gather 操作卸载到 SparseCore，当小型集合通信适合放入 VMEM 且 SparseCore 卸载开销会使其变慢时，则将其保留在 TensorCore 上。默认阈值由 VMEM 容量推导得出。该 PR 报告称，在 8k1k 配置下，并发度为 64 时吞吐量提升 2.7%，并发度为 128 时提升 5.7%。

Optimizing MoE Routing and Expert Kernels
优化 MoE 路由与专家内核

A mixture-of-experts layer produces irregular groups of tokens for each expert, and these ragged groups have to be reshaped into something the TPU's matrix units can consume efficiently. Some of the changes below are general improvements to the routing and grouped-matmul backend that benefit any MoE model. Others were measured directly on Qwen3.5.

混合专家层会为每个专家生成不规则的 token 分组，这些参差不齐的分组需要被重塑为 TPU 矩阵单元能够高效处理的形式。以下部分改动是对路由和分组矩阵乘法后端的通用优化，适用于任何 MoE 模型。其他改动则直接在 Qwen3.5 上进行了测量验证。

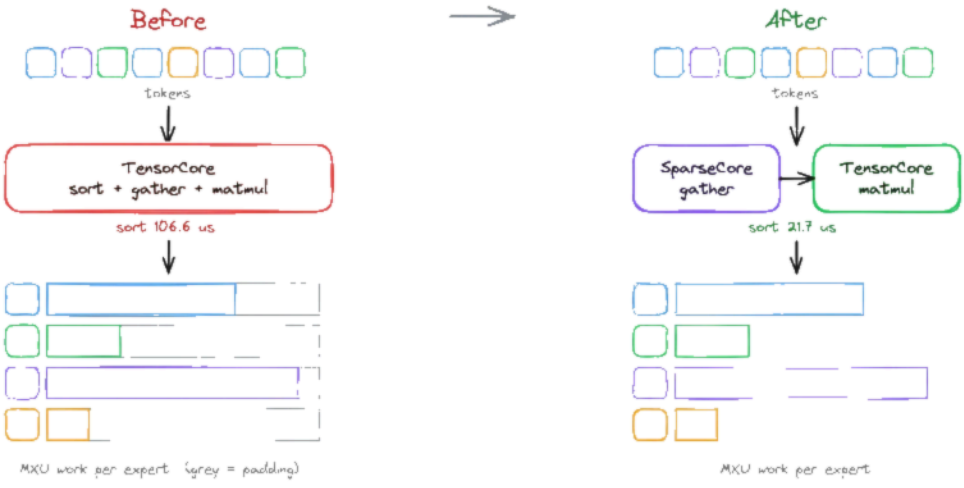
SemiAnalysis is a reader-supported publication. To receive new posts and support our work, consider becoming a free or paid subscriber.

The second version of the grouped matmul reworks how expert inputs are fed to the MXU. It removes redundant tile computation, sizes transfers to the number of valid rows rather than the padded maximum, triple-buffers expert weights so the next group's weights are already in flight while the current group is computed, and fuses group metadata generation into the kernel.

第二版分组矩阵乘法重新设计了专家输入馈送到 MXU 的方式。它移除了冗余的 tile 计算，将传输大小调整为有效行数而非填充后的最大值，对专家权重进行三重缓冲，使得当前分组计算时下一组的权重已在传输中，并将分组元数据生成融合到内核中。

The irregular rearrangement of expert inputs was then moved onto the SparseCore. The SparseCore handles data movement, gathering each expert's tokens into contiguous groups, while the TensorCore is left to run the expert matrix multiplications. A subsequent rewrite improved memory-read pipelining and split the combine work across tokens and hidden dimensions. The PR reports a 12% increase in 8k1k serving throughput compared with the original SparseCore kernels, along with lower TTFT and TPOT.

专家输入的不规则重排随后被移至 SparseCore 上执行。SparseCore 负责数据处理，将每个专家的 token 收集到连续的分组中，而 TensorCore 则专注于执行专家矩阵乘法。后续的重写优化了内存读取流水线，并将合并工作分散到 token 和隐藏维度上。该 PR 报告称，与原始 SparseCore 内核相比，8k1k 服务吞吐量提升了 12%，同时 TTFT 和 TPOT 也有所降低。



Source: SemiAnalysis 来源：SemiAnalysis

A further optimization moved the top-k weight gather in the ragged gather-reduce path onto the SparseCore. Here, gather means reading selected entries from memory

on one device, not an inter-device all-gather. Previously, the TensorCore gathered routing weights and source indices during preprocessing, limiting TensorCore/SparseCore overlap. Moving these gathers into the SparseCore kernel reduces TensorCore overhead from 29 µs to 14 µs and overall operation latency from 146 µs to 137 µs in a DeepSeek-V3 microbenchmark at batch 2k with 16-way expert parallelism.

进一步的优化将不规则聚合-归约路径中的 top-k 权重收集操作移至 SparseCore 上执行。此处，“收集”指从单个设备的内存中读取选定条目，而非设备间的全收集操作。此前，TensorCore 在预处理阶段负责收集路由权重和源索引，限制了 TensorCore/SparseCore 的重叠执行。将这些收集操作移入 SparseCore 内核后，在 DeepSeek-V3 微基准测试中（批次大小为 2k，采用 16 路专家并行），TensorCore 开销从 29 微秒降至 14 微秒，整体操作延迟从 146 微秒降至 137 微秒。

For small batches, the general ragged path costs more than the work it is arranging. A dedicated small-batch permutation therefore builds one-hot matrices and uses ordinary matrix multiplication to permute tokens to their experts and unpermute the results. On the 8k1k workload, this improved throughput by 7.3% at concurrency 64 and 5.1% at concurrency 128.

对于小批次场景，通用不规则路径的开销超过了其实际组织工作的价值。因此，专用的小批次置换方案构建 one-hot 矩阵，并利用普通矩阵乘法将令牌置换到对应专家处，再将结果反置换回来。在 8k1k 工作负载下，该优化在并发度为 64 时提升了 7.3% 的吞吐量，在并发度为 128 时提升了 5.1%。

A WIP change packs the expert ID and token index into a single sort key, so XLA performs a simpler sort while preserving the required ordering. Sort latency drops from 106.6 µs to 21.7 µs. The PR reports 8k1k serving gains of 0.6 to 8.5%, but the same change also enables an FP8 all-gather.

一项进行中的改动将专家 ID 和令牌索引打包为单一排序键，使 XLA 能够执行更简单的排序，同时保持所需的排序顺序。排序延迟从 106.6 微秒降至 21.7 微秒。该 PR 报告了 8k1k 服务场景下 0.6%至 8.5%的性能提升，但同一改动还支持了 FP8 全收集操作。

Optimizing Gated DeltaNet Pallas Kernels
优化 Gated DeltaNet Pallas 内核

Gated DeltaNet (GDN) is a recurrent computation. That is, at each step the running state is decayed, updated with a rank-one term, and projected to produce an output. Check out the following article for a deeper dive into linear attention mechanisms

such as GDN:

Gated DeltaNet（GDN）是一种循环计算。也就是说，在每一步中，运行状态被衰减，用秩一项更新，并投影以生成输出。请查看以下文章，深入了解线性注意力机制（如 GDN）：



Kimi K3, The Manos, The Mythos, The Legendos

Kimi K3，传奇之人，神话之身，传说之魂

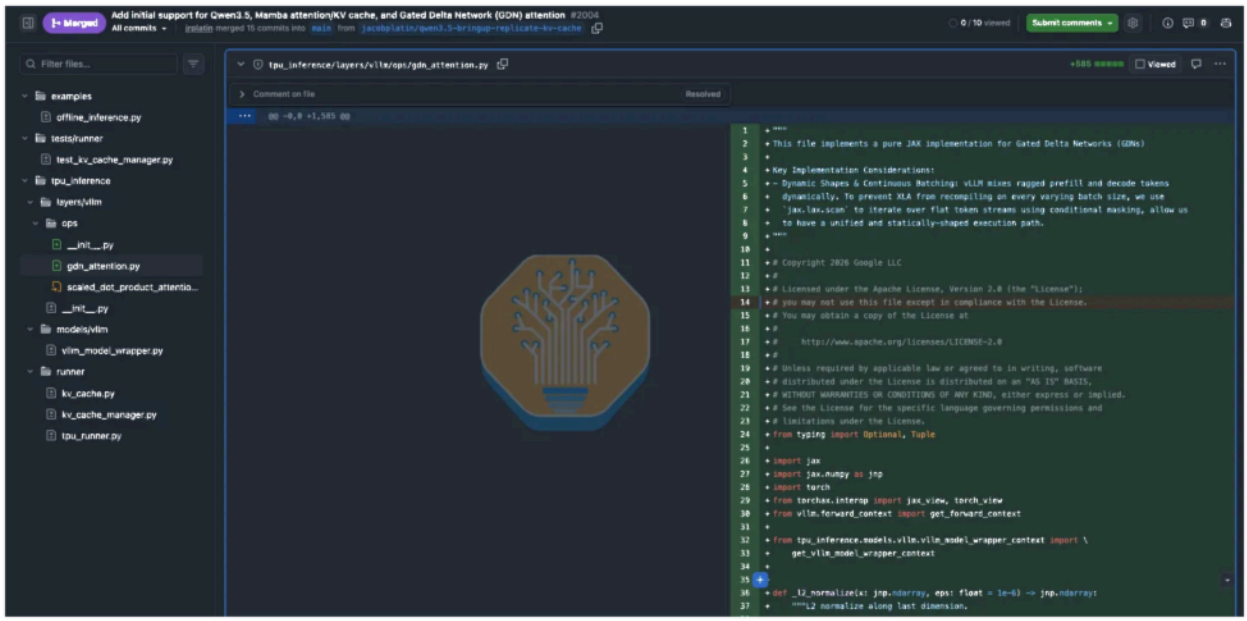
KIMBO CHEN, SHUBHAM CHOUDHARI, AND 2 OTHERS

KIMBO CHEN、SHUBHAM CHOUDHARI 及其他 2 人 · 8月4日 8 月 4 日

Read full story 阅读全文 →

The changes below show how its matrix operations, vector updates, and state transfers are scheduled across the MXU, VPU, VMEM, and HBM. Initial Qwen3.5 support added pure JAX implementations of causal Conv1D and GDN, connected them to TPU operator dispatch, and enabled recurrent-state caching. Later PRs optimized these implementations.

以下更改展示了其矩阵运算、向量更新和状态传输如何在 MXU、VPU、VMEM 和 HBM 之间调度。初始 Qwen3.5 支持添加了因果 Conv1D 和 GDN 的纯 JAX 实现，将其连接到 TPU 算子调度，并启用了循环状态缓存。后续的拉取请求优化了这些实现。



Source: GitHub 来源：GitHub

Another low-hanging fruit optimization is achieved by rearranging the algebra in the output projection calculation to overlap MXU and VPU work. Previously, the VPU first applied the rank-one update to the decayed state, and the MXU then multiplied the updated state by the query to produce the output. That dependency forced the

MXU to wait.

另一个容易实现的优化是通过重新排列输出投影计算中的代数运算，使 MXU 和 VPU 的工作重叠。此前，VPU 首先对衰减状态应用秩一更新，然后 MXU 将更新后的状态与查询相乘以生成输出。这种依赖关系迫使 MXU 等待。

Writing the decayed state as S, the correction vector as Δ, and the key and query as k and q, the output can be expanded:

将衰减状态记为 S，修正向量记为Δ，键和查询分别记为 k 和 q，输出可以展开为：

$$o = \left(S + \Delta k^{\mathrm{T}}\right) q = Sq + \Delta \left(k^{\mathrm{T}} q\right)$$

Source: SemiAnalysis 来源：SemiAnalysis

The MXU can now compute Sq directly from the decayed state while the VPU constructs the updated state for the next token. The current update's contribution to the output is calculated separately using $k^{\intercal}q$, a scalar dot product per head, followed by a small scaled-vector addition. This removes the state update from the MXU's dependency path. Reported 8k1k throughput gains were 2.79% at concurrency 64 and 4.48% at concurrency 512.

现在，MXU 可以直接从衰减状态计算 Sq，而 VPU 则为下一个令牌构建更新后的状态。当前更新对输出的贡献使用 $k^{\intercal}q$（每个头的标量点积）单独计算，随后进行小规模的缩放向量加法。这将状态更新从 MXU 的依赖路径中移除。报告的 8k1k 吞吐量提升在并发 64 时为 2.79%，在并发 512 时为 4.48%。

The next change reduces vector-register spills by slicing Q and K inside the decode loop so fewer values stay live at once. The decode-64 kernel becomes roughly 20% faster, but end-to-end gains are smaller, at 0.8% on 8k1k and 3.8% on 1k8k at concurrency 512, since the kernel is only one part of a decode step.

下一个更改通过在解码循环内对 Q 和 K 进行切片来减少向量寄存器溢出，从而使更少的数值同时保持活跃。解码-64 内核的速度大约提升了 20%，但端到端的收益较小，在并发度为 512 时，8k1k 的收益为 0.8%，1k8k 的收益为 3.8%，因为该内核只是解码步骤的一部分。

Asynchronous state transfers overlap DMA with computation using double buffering. The extra VMEM consumed by the second set of buffers initially regressed data-parallel attention. Reusing existing scratch buffers and shortening the lifetimes of temporaries recovered that capacity and removed the regression. The reported 8k1k

throughput gain is 11.3% at concurrency 512.

异步状态传输通过双缓冲将 DMA 与计算重叠。第二组缓冲区额外消耗的 VMEM 最初导致数据并行注意力的性能回退。通过重用现有的暂存缓冲区并缩短临时变量的生命周期，恢复了该容量并消除了回退。所报告的 8k1k 吞吐量增益在并发度为 512 时为 11.3%。

GDN v3 fuses Conv1D and GDN into a single kernel to cut HBM round trips, improves prefill layouts, and unifies mixed prefill/decode execution into one path. Reported kernel-level speedups are 1.41× for decode, 1.60× for prefill, and 2.14× for mixed batches. These measurements cover the kernel alone and do not establish end-to-end serving gains.

GDN v3 将 Conv1D 和 GDN 融合为单个内核，以减少 HBM 往返，改进预填充布局，并将混合预填充/解码执行统一为一条路径。所报告的内核级加速比为：解码 1.41 倍，预填充 1.60 倍，混合批次 2.14 倍。这些测量仅涵盖内核本身，并未确立端到端的服务收益。



Source: SemiAnalysis 来源：SemiAnalysis

Significant performance gains can be achieved simply by improving the overlap between operations within a kernel!

仅通过改善内核内操作之间的重叠，就能实现显著的性能提升！

Managing Hybrid State and Paged Attention
管理混合状态与分页注意力

Qwen3.5 is a hybrid model with two kinds of state: GQA layers accumulate a KV history that grows with every token, while GDN layers hold a fixed-size recurrent state per request. State allocation, storage precision, attention page size, and physical data layout all determine how much HBM is actually usable and how efficiently attention

runs.

Qwen3.5 是一种混合模型，具有两种状态：GQA 层累积随每个 token 增长的 KV 历史，而 GDN 层为每个请求持有固定大小的循环状态。状态分配、存储精度、注意力页大小和物理数据布局共同决定了实际可用的 HBM 容量以及注意力运行的效率。

SemiAnalysis is a reader-supported publication. To receive new posts and support our work, consider becoming a subscriber.

Batched ragged paged attention batches sequences together, precomputes page metadata, and triple-buffers to improve pipelining and reduce padding. This change provides shared attention-backend groundwork. The PR's workload examples use Qwen3-32B, so its measurements should be kept separate from the Qwen3.5-397B results.

批量不规则分页注意力将序列批量组合，预计算页元数据，并采用三重缓冲以改善流水线并减少填充。此更改提供了共享的注意力后端基础。该 PR 的工作负载示例使用 Qwen3-32B，因此其测量结果应与 Qwen3.5-397B 的结果分开看待。

Recurrent state is now allocated compactly, with roughly one slot per active request instead of `num_blocks` slots for every layer group. In the reported configuration, this reclaims about 76 GiB of HBM and expands the attention block pool by 71%, improving 1k8k output throughput by 18% at concurrency 64.

循环状态现在被紧凑地分配，每个活动请求大约占用一个槽位，而不是每个层组占用 `num_blocks` 个槽位。在所报告的配置中，这回收了约 76 GiB 的 HBM，并将注意力块池扩大了 71%，在并发度为 64 时，1k8k 输出吞吐量提升了 18%。

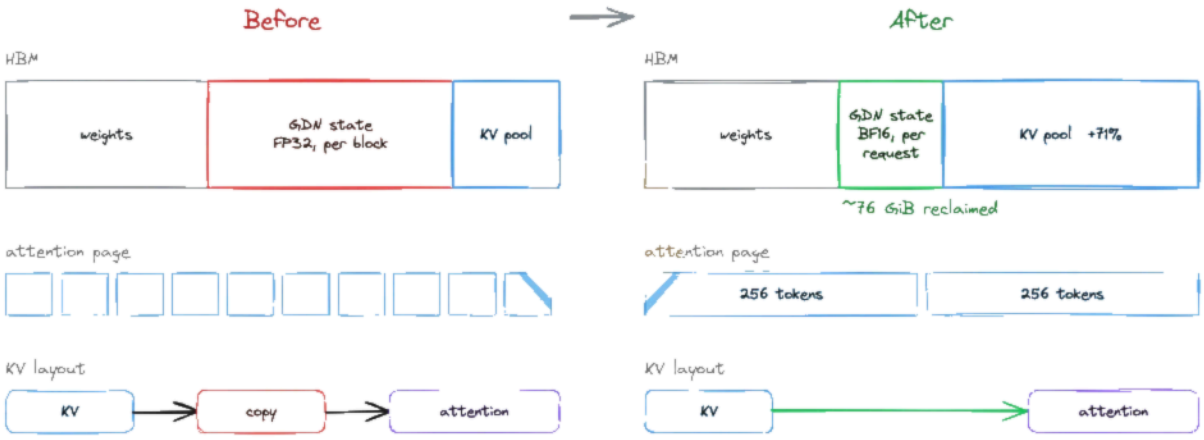
Storing the recurrent state in BF16 halves its HBM footprint while keeping the arithmetic in FP32 inside VMEM. This saves memory capacity and transfer bandwidth while preserving FP32 arithmetic. The reported 1k8k throughput gain is 15% at concurrency 512.

将循环状态以 BF16 存储，可将其 HBM 占用减半，同时在 VMEM 内部保持 FP32 算术运算。这节省了内存容量和传输带宽，同时保留了 FP32 算术精度。报告显示，在并发度为 512 时，1k8k 吞吐量提升了 15%。

Removing the old hybrid page-size alignment constraint lets batched attention use a suitable power-of-two page size, including 256 tokens. 1k8k throughput improves by about 7% at concurrency 512. This applies to the non-prefix-caching path and is

distinct from the aligned checkpoint mode introduced later.

移除旧的混合页大小对齐约束，使批量注意力能够使用合适的 2 的幂次页大小，包括 256 个 token。在并发度为 512 时，1k8k 吞吐量提升约 7%。这适用于非前缀缓存路径，与后续引入的对齐检查点模式不同。



Source: SemiAnalysis 来源：SemiAnalysis

Unnecessary KV-layout copies are avoided by selecting the reshape path based on KV-head count, rather than imposing a layout constraint that only other shapes need. Qwen3.5 8k1k throughput improves by about 4.1% at concurrency 512.

通过根据 KV 头数量选择重塑路径，而非强加仅其他形状需要的布局约束，避免了不必要的 KV 布局复制。在并发度为 512 时，Qwen3.5 8k1k 吞吐量提升约 4.1%。

Reducing Padding and Low-Concurrency Overhead

减少填充与低并发开销

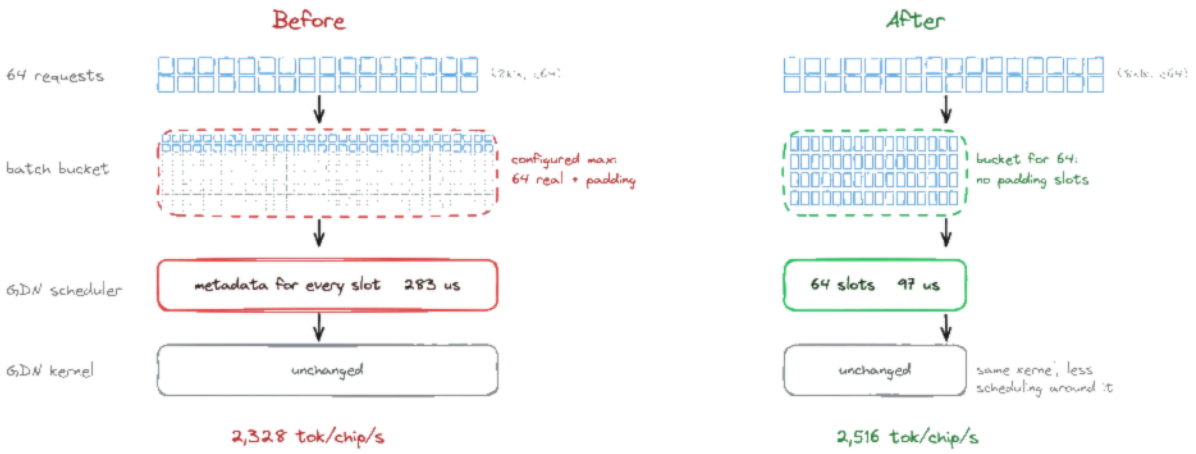
When only four or eight requests are in flight, a configuration tuned for hundreds of concurrent requests wastes work. Compiled shape buckets are too large, metadata is sized for the configured maximum, and padding tokens trigger dummy expert work. The InferenceX operating points sit at exactly these low concurrencies, which motivated the following changes.

当只有四个或八个请求在途时，为数百个并发请求调优的配置会浪费工作。编译后的形状桶过大，元数据按配置的最大值来设定大小，填充令牌会触发无用的专家工作。InferenceX 的工作点恰好处于这些低并发水平，这促使了以下更改。

Request metadata is now bucketed by the number of active requests instead of always using the configured maximum. In the 8k1k concurrency-64 test, GDN scheduling overhead falls from 283 µs to 97 µs and throughput rises from 2,328 to 2,516

tokens/chip/s.

请求元数据现在按活动请求数量进行分桶，而不是始终使用配置的最大值。在 8k1k 并发 64 测试中，GDN 调度开销从 283 微秒降至 97 微秒，吞吐量从每秒每芯片 2,328 个令牌提升至 2,516 个令牌。



Source: SemiAnalysis 来源：SemiAnalysis

Explicit Qwen3.5 tuning for InferenceX shrinks rotary tables, sizes sequence limits per DP rank, and adds a dedicated attention bucket for concurrency four. The combined gains at concurrency four are 13.3% on 8k1k and 15.5% on 1k1k.

针对 InferenceX 的显式 Qwen3.5 调优缩小了旋转表，按 DP 排名调整序列长度限制，并为并发四添加了专用的注意力桶。在并发四时，8k1k 的综合增益为 13.3%，1k1k 为 15.5%。

A further round of low-concurrency tuning switches to TP8 attention plus expert parallelism, reduces the minimum token bucket, and routes padding tokens to expert zero so they do not trigger additional expert weight loads. Combined 1k1k gains are 22.9% at concurrency four and 18.1% at eight. On 8k1k, throughput rises by 9.2% at concurrency four but falls by 5.3% at concurrency eight.

进一步的低并发调优轮次切换到 TP8 注意力加专家并行，减少了最小令牌桶，并将填充令牌路由到专家零，这样它们就不会触发额外的专家权重加载。1k1k 的综合增益在并发四时为 22.9%，在并发八时为 18.1%。在 8k1k 上，并发四时吞吐量提升 9.2%，但并发八时下降 5.3%。

Enabling Prefix Caching for Hybrid Models
为混合模型启用前缀缓存

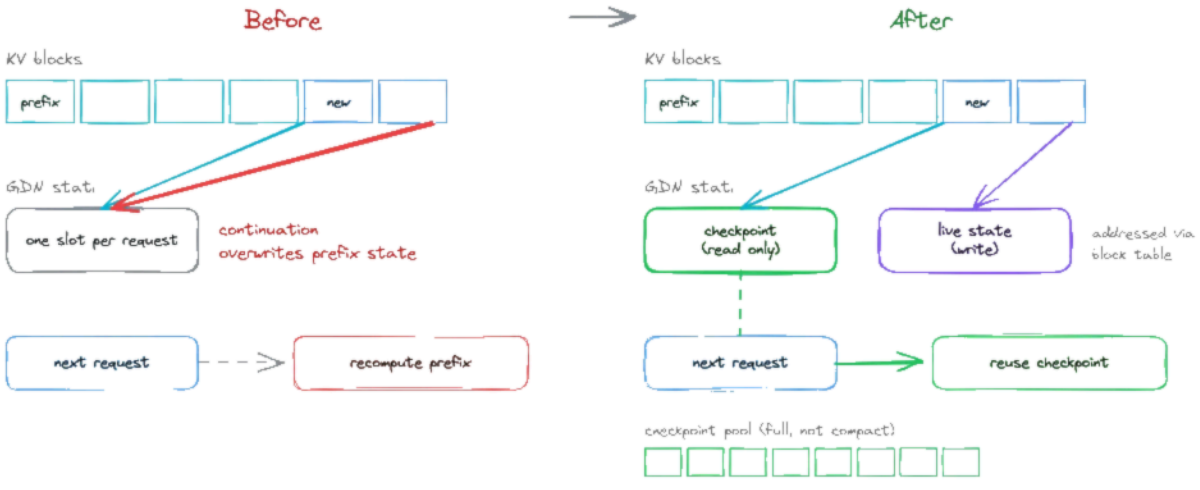
The gains above were measured on random-input benchmarks, where nothing is shared between requests. Prefix caching is crucial for agentic and multi-turn workloads, which reuse long system prompts and conversation histories. For a hybrid

model, a cached prefix must retain both its KV blocks and the GDN recurrent state at the end of the prefix. That recurrent state is normally overwritten as soon as the request continues, making prefix caching harder than for "regular" attention models.

上述增益是在随机输入基准测试中测得的，此时请求之间没有任何共享内容。前缀缓存对于智能体及多轮工作负载至关重要，这些负载会复用较长的系统提示和对话历史。对于混合模型，缓存的前缀必须同时保留其 KV 块以及前缀末尾的 GDN 循环状态。该循环状态通常在请求继续时立即被覆盖，这使得前缀缓存比"常规"注意力模型更难实现。

Hybrid prefix caching with DP support resolves this by giving GDN separate slots for reading a checkpoint and writing the live state. Continuing a request therefore no longer overwrites the checkpoint associated with its cached prefix. State addresses are derived from the block table, the same structure that already locates KV blocks. Checkpoints are taken at an aligned cache granularity so a saved state always lines up with a KV block boundary. This mode needs a full checkpoint pool rather than the compact per-request allocation described earlier, trading some HBM for reuse across requests. The benefit shows up when prefixes are actually shared.

支持 DP 的混合前缀缓存通过为 GDN 提供独立的读取检查点和写入实时状态的槽位来解决此问题。因此，继续处理请求不再覆盖与其缓存前缀关联的检查点。状态地址从块表中派生，该结构与定位 KV 块的结构相同。检查点以对齐的缓存粒度进行保存，因此保存的状态始终与 KV 块边界对齐。此模式需要完整的检查点池，而非前述的紧凑型按请求分配，以牺牲部分 HBM 换取跨请求的复用。当前缀实际被共享时，这种优势便会显现。



Source: SemiAnalysis 来源：SemiAnalysis

Lane Layout and Pipelining Depth in Paged Attention

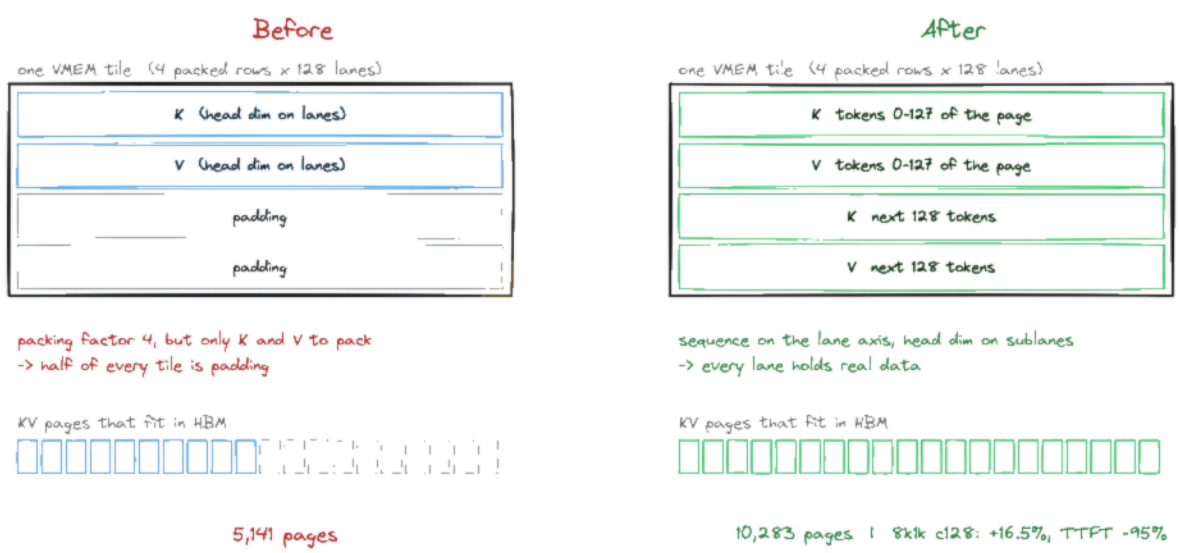
分页注意力中的车道布局与流水线深度

Two hardware facts shape how a KV cache is laid out on TPU. We discuss TPU hardware in more detail in the following section. First, the vector unit works on tiles whose last dimension is 128 lanes wide, so any array whose trailing dimension is smaller than that is padded up. Second, a Pallas kernel hides HBM latency through double-buffering. It fetches the next block while computing the current one. Both blocks must fit in VMEM, so the size of the compute block determines how deep the prefetch can be. Both constraints were costing capacity and throughput.

两个硬件事实决定了 KV 缓存如何在 TPU 上布局。我们将在下一节更详细地讨论 TPU 硬件。首先，向量单元处理的是最后一维为 128 通道宽的瓦片，因此任何尾维小于该值的数组都会被填充。其次，Pallas 内核通过双缓冲来隐藏 HBM 延迟。它在计算当前块的同时获取下一个块。两个块都必须适合 VMEM，因此计算块的大小决定了预取可以有多深。这两个约束都在消耗容量和吞吐量。

In the batched attention kernel, the KV cache packed keys and values along the head dimension, and for FP8 that packing factor is four. A model with a single KV head per device only has two things to pack, so half of every tile was wasted on padding. A sequence-on-lane layout fixes this by putting the page's tokens on the 128-lane axis and the head dimension on the sublane axis instead. Usable KV pages double (from 5,141 to 10,283 in the reported configuration) and the head dimension only needs to be a multiple of 32 rather than 128, which allows models with a head dim of 64 to use the kernel as well. The layout costs about 3% in per-token latency at low concurrency, but at concurrency 128 on 8k1k the extra capacity lifts throughput 16.5% and cuts median TTFT by 95%, because requests no longer wait for KV space.

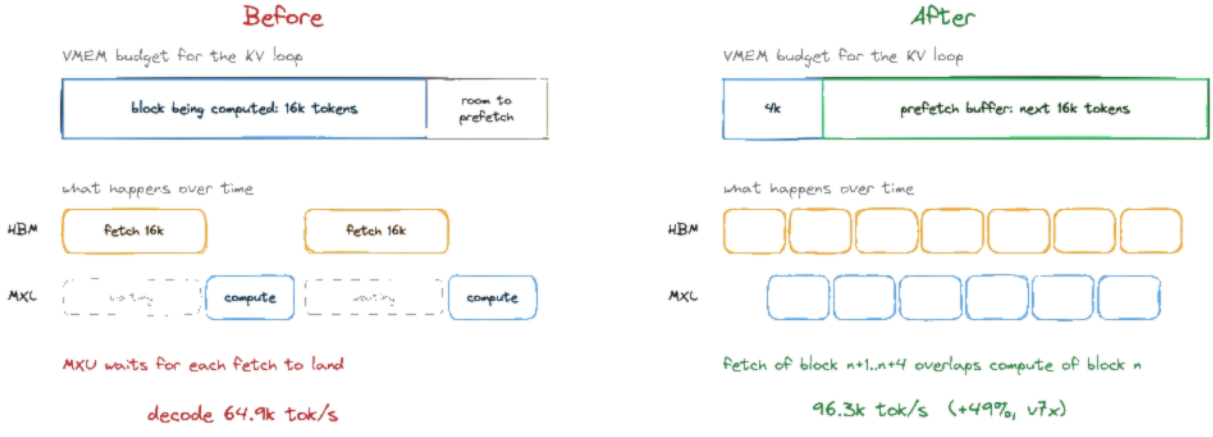
在批量注意力内核中，KV 缓存沿头维度打包键和值，对于 FP8，打包因子为四。每个设备只有一个 KV 头的模型只有两个需要打包的元素，因此每个瓦片的一半都浪费在填充上。序列在通道上的布局通过将页面的令牌放在 128 通道轴上，而将头维度放在子通道轴上来解决这个问题。可用的 KV 页面翻倍（在报告的配置中从 5,141 增加到 10,283），并且头维度只需要是 32 的倍数而不是 128，这使得头维度为 64 的模型也能使用该内核。这种布局在低并发下每个令牌的延迟成本约为 3%，但在 8k1k 配置下并发为 128 时，额外的容量将吞吐量提升了 16.5%，并将中位 TTFT 降低了 95%，因为请求不再需要等待 KV 空间。



Source: SemiAnalysis 来源：SemiAnalysis

The RPA v3 block-size heuristic had a similar blind spot. During v7x decode, it set the KV compute block equal to the KV fetch block, around 16k tokens, which left almost no VMEM for the prefetch buffer. Splitting the two block sizes kept the KV fetch block at 16k tokens while reducing the compute block to 4k. This gave the pipeline room to run ahead of the MXU and increased decode throughput from 64.9k to 96.3k tokens per second, a 49% gain reproduced across four runs on Qwen3-0.6B. Throughput followed a clean inverted-U curve across the block-size sweep. The tuned-parameter table could not represent the improvement because it stored a single block size per shape, which is why the change first landed as an environment override. A follow-up extends the table to hold the fetch and compute sizes separately.

RPA v3 的块大小启发式算法也存在类似的盲区。在 v7x 解码过程中，它将 KV 计算块设置为与 KV 获取块相同，约为 16k 个令牌，这几乎未给预取缓冲区留下任何 VMEM 空间。将两个块大小分开后，KV 获取块保持在 16k 个令牌，而计算块则减少到 4k。这为流水线提供了在 MXU 之前运行的余量，并将解码吞吐量从每秒 64.9k 个令牌提升至 96.3k 个令牌，在 Qwen3-0.6B 上四次运行中均实现了 49%的提升。吞吐量在块大小扫描中呈现清晰的倒 U 形曲线。调优参数表无法体现这一改进，因为它按形状仅存储单个块大小，这也是该更改最初以环境覆盖形式落地的原因。后续工作扩展了该表，以分别存储获取和计算大小。



Source: SemiAnalysis 来源：SemiAnalysis

TPU System-Level Co-Design & Networking Explainer

TPU 系统级协同设计与网络解析

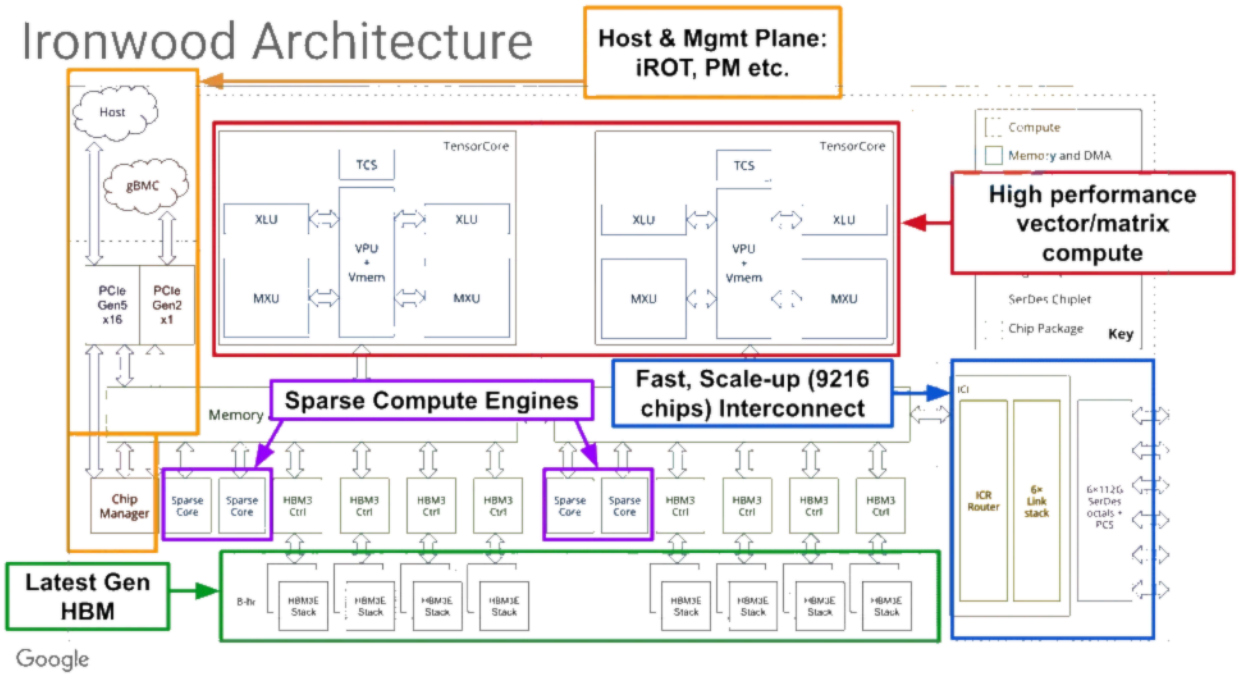
As we said in the TPU King article, Google's cost-per-token advantage on inference is a result of co-design. Instead of focusing on maximum single-chip performance, Google designs the compute die, chip-to-chip fabric, and compiler together, allowing it to optimize computation and communication as a single system. That joint optimization helps explain Ironwood's performance per dollar. The following sections examine how the chip and network work together. Anthropic is a big fan of TPUs and uses them heavily for training.

正如我们在 TPU King 文章中所说，谷歌在推理方面的每令牌成本优势源于协同设计。谷歌并未专注于最大化单芯片性能，而是将计算芯片、芯片间互连结构和编译器一同设计，从而能够将计算和通信作为一个整体系统进行优化。这种联合优化有助于解释 Ironwood 的每美元性能表现。以下章节将探讨芯片与网络如何协同工作。Anthropic 是 TPU 的忠实用户，并大量使用 TPU 进行训练。

The Ironwood Chip Ironwood 芯片

TPUv7 Ironwood breaks from the "MegaCore" convention that defined TPU v4 and TPU v5p, where two physical cores were fused into a single logical accelerator sharing one memory space. Ironwood instead has two separate compute dies, each running its own independent logical device. These dies are joined by a high-bandwidth die-to-die link rather than a unified memory fabric. JAX and other frameworks now expose them as two distinct devices per chip. Each Ironwood chip carries 2 TensorCores and 4 third-generation SparseCores. The SparseCores accelerate embedding lookups and other sparse operations that would otherwise choke a dense matrix engine.

TPUv7 Ironwood 打破了定义 TPU v4 和 TPU v5p 的"MegaCore"惯例，该惯例将两个物理核心融合为一个共享单一内存空间的逻辑加速器。Ironwood 反而采用两个独立的计算芯片，每个芯片运行自己的独立逻辑设备。这些芯片通过高带宽的芯片间链路连接，而非统一的内存架构。JAX 和其他框架现在将它们暴露为每个芯片上的两个独立设备。每个 Ironwood 芯片包含 2 个 TensorCore 和 4 个第三代 SparseCore。SparseCore 加速嵌入查找和其他稀疏操作，这些操作否则会阻塞密集矩阵引擎。



Source: Google 来源：谷歌

On the memory side, each chip has about 6x Trillium's HBM capacity, a jump that matters directly for KV-cache headroom and batch size. Ironwood is notably the first TPU generation with native FP8 hardware support, whereas prior generations had to emulate FP8 in software.

在内存方面，每个芯片的 HBM 容量约为 Trillium 的 6 倍，这一跃升直接关系到 KV 缓存余量和批处理大小。Ironwood 是首个原生支持 FP8 硬件的 TPU 代际，而前几代必须在软件中模拟 FP8。

Google TPU Chip Specifications					
	Unit	TPU v5e (Viperlite)	TPU v5p (Viperfish)	TPU v6 (Trillium)	TPU v7 (Ironwood)
Chip Specifications					
Logic Die Configuration	-	1x Compute Die, 1x IOD, 2x HBM2E 4-Hi	1x Compute Die, 1x IOD, 6x HBM2E 8-Hi	1x Compute Die, 1x IOD, 2x HBM3 8-Hi	2x Compute Dies, 1x IOD, 8x HBM3E 8-Hi
System availability	-	4Q23	1Q24	4Q24	4Q25
Foundry Node (compute)	-	N5		N5P	N3E
HBM Capacity	GB	16GB HBM2E	95GB HBM2E	32GB HBM3	192GB HBM3E
HBM Bandwidth	GB/s	819	2,765	1,640	7,380
Packaging	-	CoWoS-S			
FP8 TFLOPs (Dense)	TFLOPs	-	459	918	4,614
INT8 TFLOPs (Dense)	TFLOPs	394	918	1,836	4,614
BF16 TFLOPs (Dense)	TFLOPs	197	459	918	2,307
Chip TDP	W	~300	~550	~390	~980
Scale Up Networking					
Scale Up Topology	-	2D Torus	3D Torus	2D Torus	2D/3D Torus
Interchip Interconnect Bandwidth Uni-directional	Gb/s	1,600	4,800	3,200	4,800
TPU Pod Size	#	256	8,960	256	9,216

Source: SemiAnalysis 来源：SemiAnalysis

Read more on Ironwood in our TPUv7 article

在我们的 TPUv7 文章中了解更多关于 Ironwood 的信息



TPUv7: Google Takes a Swing at the King

TPUv7：谷歌向王者发起冲击

DYLAN PATEL, MYRON XIE, AND 10 OTHERS

DYLAN PATEL、MYRON XIE 及其他 10 人

· 2025年11月28日 2025 年 11 月 28 日

Read full story 阅读全文 →

The MXU and Why Shapes Matter
MXU 与形状为何重要

The Matrix Multiply Unit, the engine that actually does the multiplying, is a systolic array: a two-dimensional grid of multiply-accumulate cells. Weights are loaded into the array and held stationary. Activations stream in from the edge, and partial sums ripple through the grid cell by cell, accumulating as they go. The finished result streams out the other side without ever touching memory mid-computation. Every TPU generation through v5 used a 128x128 MXU, good for 16,384 MACs per cycle. Starting with TPU v6e and carrying through to Ironwood, the array doubled to 256x256, or 65,536 MACs per cycle, delivering 4x more FLOPs per cycle than the previous design.

矩阵乘法单元（MXU）是实际执行乘法运算的引擎，它是一种脉动阵列：由乘法累加单元组成的二维网格。权重被加载到阵列中并保持固定。激活值从边缘流入，部分和逐单元在网格中传播并不断累加。最终结果从另一侧流出，整个计算过程中无需访问内存。从每一代 TPU 到 v5，都采用了 128x128 的 MXU，每个周期可执行 16,384 次 MAC 运算。从 TPU v6e 开始，并延续至 Ironwood，阵列尺寸翻倍至 256x256，即每个周期 65,536 次 MAC 运算，相比前代设计，每个周期的 FLOPs 提升了 4 倍。

Please read the scaling book (aka the ML systems bible) for excellent top-tier information on TPUs and scaling in general.

请阅读扩展指南（又称 ML 系统圣经），以获取关于 TPU 和扩展的顶级优质信息。

SemiAnalysis is a reader-supported publication. To receive new posts and support our work, consider becoming a subscriber.

The catch is that a bigger systolic array is only free if you can keep it full. Matrix dimensions need to be padded up to at least the MXU's side length in both directions, 128 on older generations or 256 on v6e and v7, and the XLA compiler dutifully pads any smaller axis to fill the tile. Every padded cell still occupies a MAC unit for that cycle, multiplying by a zero that contributes nothing to the answer. Llama 3 8B illustrates this with an attention head dimension of 128. On Ironwood's 256x256 MXU, that head dimension is exactly half the array's native width, which caps the two attention matmuls at a maximum of 50% MXU utilization (not 75%, because only the

head dimension is 128).

关键在于，更大的脉动阵列只有在保持满载时才是免费的。矩阵维度需要在两个方向上都填充至至少 MXU 的边长，旧代次为 128，v6e 和 v7 则为 256，而 XLA 编译器会尽职尽责地将任何较小的轴填充以填满整个块。每个填充的单元在该周期仍占用一个 MAC 单元，乘以一个对结果毫无贡献的零。Llama 3 8B 就是一个例子，其注意力头维度为 128。在 Ironwood 的 256x256 MXU 上，该头维度恰好是阵列原生宽度的一半，这使两个注意力矩阵乘法运算的 MXU 利用率最高只能达到 50%（而非 75%，因为只有头维度是 128）。

The implication runs deeper than one model's attention layer. Architectural choices that used to be arbitrary hyperparameters (such as head dimensions, the KV head count that survives after tensor-parallel sharding, and MoE expert widths) increasingly need to be made with TPU tile geometry in mind, because a shape mismatch there is a direct tax on delivered throughput regardless of how good the rest of the stack is. Kernel authors also need to account for this explicitly. Production attention kernels like ragged paged attention use explicit packing dimensions to reduce padding when XLA's default tiling produces an inefficient layout.

其影响远不止单个模型的注意力层。以往被视为任意超参数的架构选择（如头维度、张量并行分片后保留的 KV 头数量，以及 MoE 专家宽度）越来越需要结合 TPU 瓦片几何来考量，因为那里的形状不匹配会直接造成吞吐量损失，无论堆栈其余部分多么优秀。内核开发者也需要明确考虑这一点。生产级注意力内核（如 ragged paged attention）使用显式打包维度来减少填充，当 XLA 的默认分块产生低效布局时尤为关键。

TPUs Are Picky TPU 很挑剔

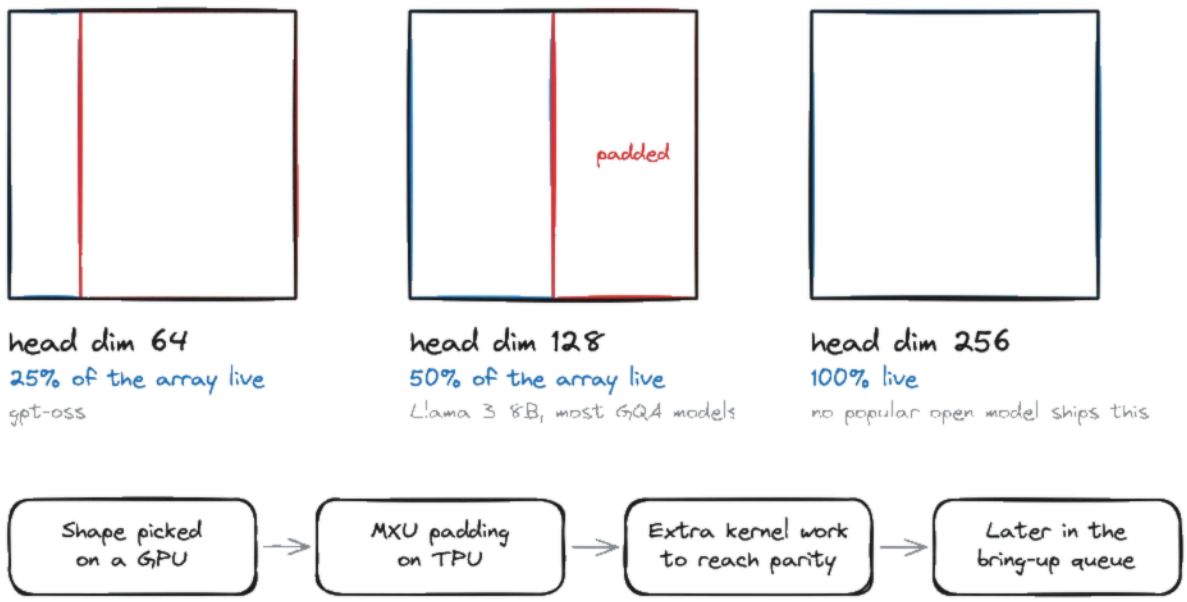
GPU matrix cores consume small tiles, so a wide range of head dimensions, expert widths, and post-sharding KV head counts all land near peak. A dimension of 64 instead of 128 pays close to nothing on an H100 or a B200 and gets a cheaper attention layer for it. GPU architecture allowed researchers to prioritize evaluations performance without a loss in inference performance. But for TPUs, this hyperparameter becomes a tradeoff

GPU 矩阵核心处理小块数据，因此各种头维度、专家宽度以及分片后的 KV 头数量都能接近峰值性能。在 H100 或 B200 上，使用 64 而非 128 的维度几乎不产生额外开销，同时还能获得更经济的注意力层。GPU 架构使研究人员能够优先考虑评估性能，而不会牺牲推理性能。但对于 TPU 而言，这一超参数则成为了一种权衡。

As previously mentioned, these choices are not free on a 256-wide systolic array. By the same arithmetic that caps Llama 3 8B's attention matmuls at 50%, a head dimension of 64 caps them at 25% before a single line of kernel code is written. gpt-oss ships at 64. DeepSeek's MLA splits its query/key dimension into 128 plus 64 for a total

of 192, ungainly against any power-of-two array and worse against a wide one.

如前所述，在 256 宽度的脉动阵列上，这些选择并非没有代价。根据限制 Llama 3 8B 注意力矩阵乘法效率为 50%的相同计算逻辑，头维度为 64 时，在编写任何一行内核代码之前，其效率就被限制在 25%。gpt-oss 采用 64 的头维度。DeepSeek 的 MLA 将其查询/键维度拆分为 128 加 64，总计 192，这在任何二次幂阵列上都显得不协调，在宽阵列上则更为不利。



The consequence is that bring-up cost varies enormously, and it correlates poorly with how popular a model is. A model whose shapes fall out cleanly needs scheduling, serving, and tuning work, which is measured in weeks. A model that fights the tile geometry needs new kernels before it is even at parity, let alone winning on performance per dollar. Kernel engineers are finite, so sequencing externalization toward models where the hardware is not starting a lap down is the rational move, and we would expect the same behavior from any accelerator vendor in this position.

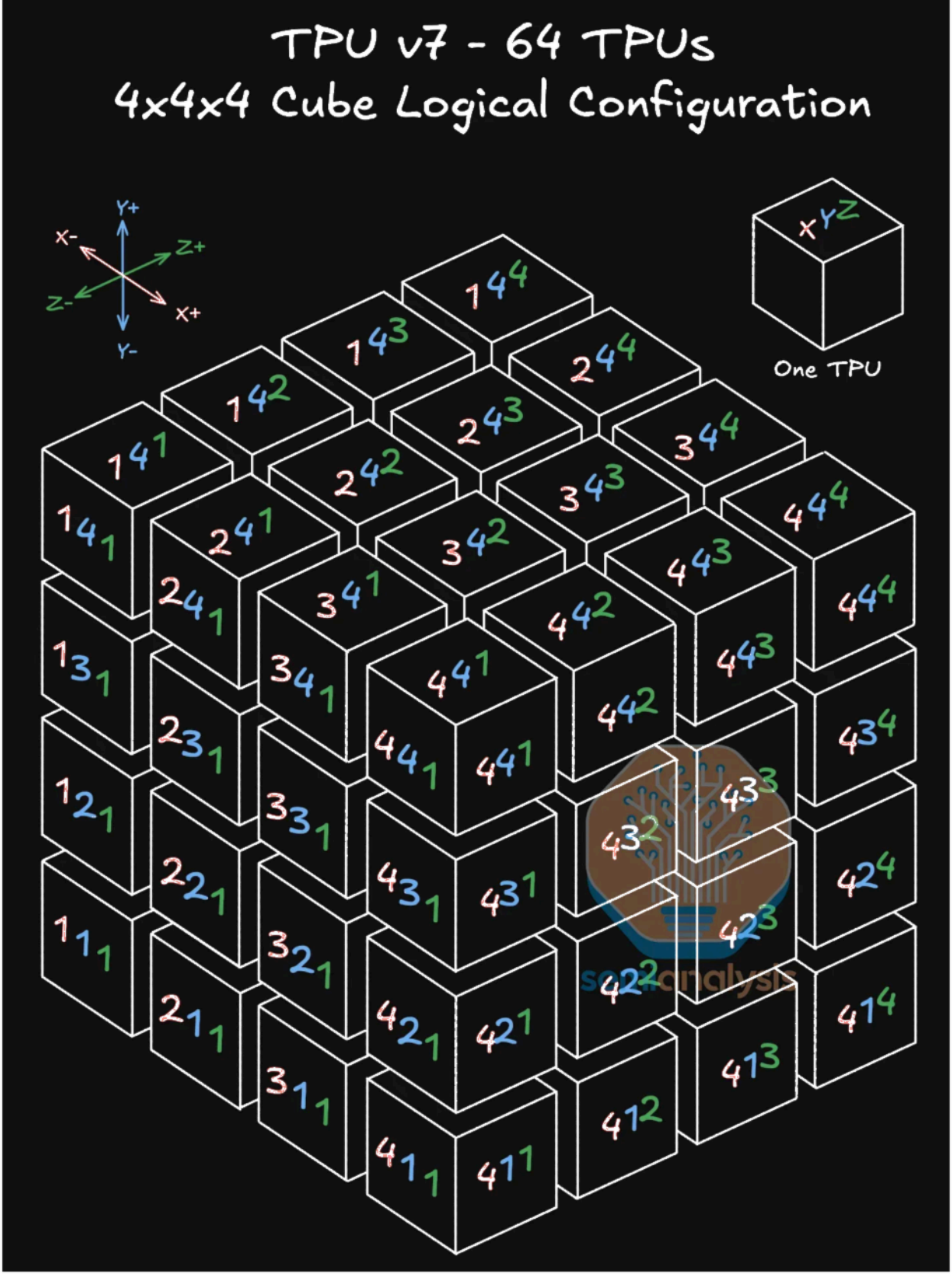
其结果是，模型的上线成本差异巨大，且与模型的流行程度关联甚微。形状参数恰好匹配的模型，只需进行调度、服务和调优工作，耗时以周计。而与瓦片几何结构不匹配的模型，在达到同等性能之前就需要开发新的内核，更不用说在性价比上取得优势了。内核工程师资源有限，因此将外部化工作优先安排给硬件本身不处于劣势的模型是合理之举，我们预计任何处于此位置的加速器供应商都会采取相同策略。

Torus Topology 环形拓扑结构

Inference and training at scale run across thousands of chips that need to talk to each other constantly. TPUs connect P2P over a custom network called ICI (Inter-Chip Interconnect). ICI bypasses the host CPU entirely, allowing chips to exchange activations and gradients directly instead of routing them through PCIe and a general-purpose NIC. The topology behind ICI has evolved generation over generation: TPU v2 and v3 used a 2D torus with each chip wired to 4 neighbors, and starting with TPU

v4 and v5p, Google moved to a 3D torus with each chip wired to 6 neighbors along the +/-X, +/-Y, and +/-Z axes. Ironwood (TPUv7) keeps that 3D torus. The fundamental building block is a 4x4x4 cube of 64 chips, sized to map cleanly onto one physical server rack.

大规模推理和训练需要在数千块芯片上运行，这些芯片必须持续相互通信。TPU 通过名为 ICI（芯片间互连）的自定义网络进行点对点连接。ICI 完全绕过了主机 CPU，使芯片能够直接交换激活值和梯度，而无需通过 PCIe 和通用网卡进行路由。ICI 背后的拓扑结构逐代演进：TPU v2 和 v3 采用二维环形拓扑，每块芯片与 4 个邻居相连；从 TPU v4 和 v5p 开始，Google 转向三维环形拓扑，每块芯片沿+/-X、+/-Y 和+/-Z 轴与 6 个邻居相连。Ironwood（TPUv7）延续了这种三维环形拓扑。其基本构建单元是一个 4x4x4 的 64 芯片立方体，尺寸设计为可整齐映射到一个物理服务器机架上。

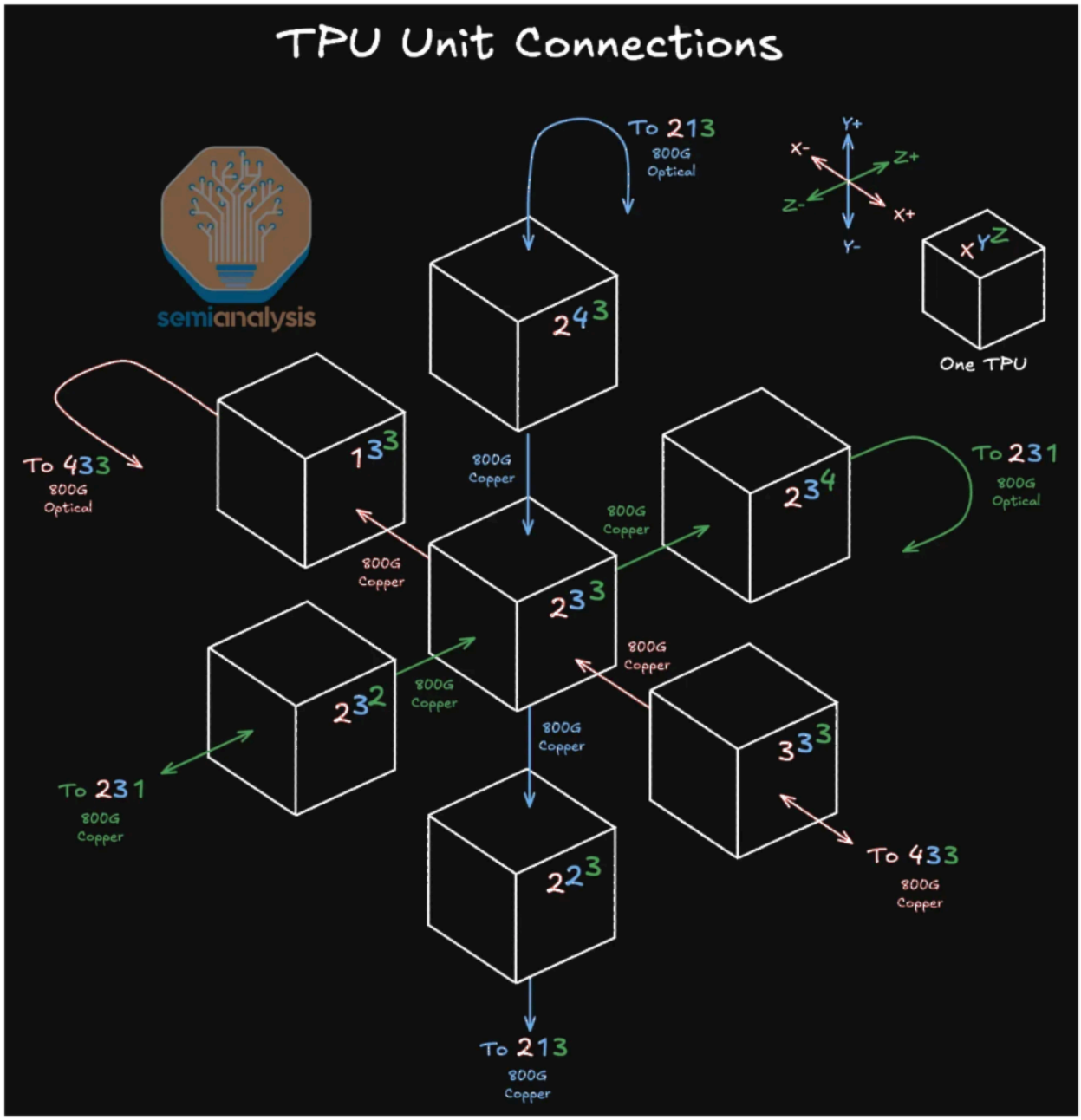


Source: SemiAnalysis 来源：SemiAnalysis

The difference between a torus and a plain grid is its wraparound links. Connecting the ends of a line forms a ring, cutting the worst-case hop distance from N to N/2. It's the same trick that makes Pac-Man's maze feel smaller than it is. Google pushes this further with a "twisted torus," a Mobius-strip-like wraparound that shaves the average hop count down even more. To scale beyond a single 4x4x4 cube, Google stitches cubes together using Optical Circuit Switches, which preserve the wraparound property across a much larger reconfigurable topology, scaling all the way up to Ironwood's full 9,216-chip superpod and its 42.5 FP8 exaflops of aggregate compute. The operational payoff of OCS is that Google can physically rewire around a failed link or a dead chip

in seconds, using mirrors rather than sending a technician to re-splice copper in a live data center.

环面与普通网格的区别在于其环绕链接。将一条线的两端连接形成环，可将最坏情况下的跳数距离从 N 缩短至 N/2。这正是让《吃豆人》迷宫感觉比实际更小的巧妙手法。谷歌通过“扭曲环面”进一步推进这一概念，这是一种类似莫比乌斯带的环绕结构，能进一步降低平均跳数。为了扩展到单个 4x4x4 立方体之外，谷歌使用光路交换机将立方体拼接在一起，在更大的可重构拓扑中保持环绕特性，可一直扩展到 Ironwood 完整的 9,216 芯片超级集群及其 42.5 FP8 exaflops 的聚合算力。OCS 在运营上的优势在于，谷歌可以通过镜子在数秒内物理重连绕过故障链路或失效芯片，而无需派技术人员在运行中的数据中心重新拼接铜缆。



Source: SemiAnalysis 来源：SemiAnalysis

This scale-out design was revolutionary at the time. Before NVL72 racks, models that could not fit within a single 8-GPU node had to use pipeline parallelism because inter-node InfiniBand was slow. On a TPU pod, the ICI torus gives NVLink-class bandwidth across the entire pod, up to 8,960 chips on v5p. With this bandwidth, a

DSV3-scale model can be sharded across the whole pod using tensor parallelism, expert parallelism, and data parallelism (FSDP-style weight sharding), without splitting layers across pipeline stages.

这种横向扩展设计在当时具有革命性意义。在 NVL72 机架出现之前，无法容纳在单个 8-GPU 节点内的模型不得不采用流水线并行，因为节点间的 InfiniBand 速度较慢。而在 TPU Pod 上，ICI 环形拓扑在整个 Pod 内提供 NVLink 级别的带宽，在 v5p 上最高支持 8,960 个芯片。凭借这一带宽，DSV3 规模的模型可以通过张量并行、专家并行和数据并行（FSDP 式权重分片）在整个 Pod 上进行分片，而无需将层拆分到流水线阶段中。

Although a TPU torus has more hops than a single-hop NVLink switched design, our upcoming CollectiveX/NetworkingX results show that, in many cases, the TPU torus has lower latency than a single-hop NVSwitch for small EP messages.

尽管 TPU 环形拓扑的跳数多于单跳 NVLink 交换设计，但我们即将发布的 CollectiveX/NetworkingX 结果表明，在许多情况下，对于小型 EP 消息，TPU 环形拓扑的延迟低于单跳 NVSwitch。

Looking Ahead: TPUv8i's Boardfly Network
展望未来：TPUv8i 的 Boardfly 网络

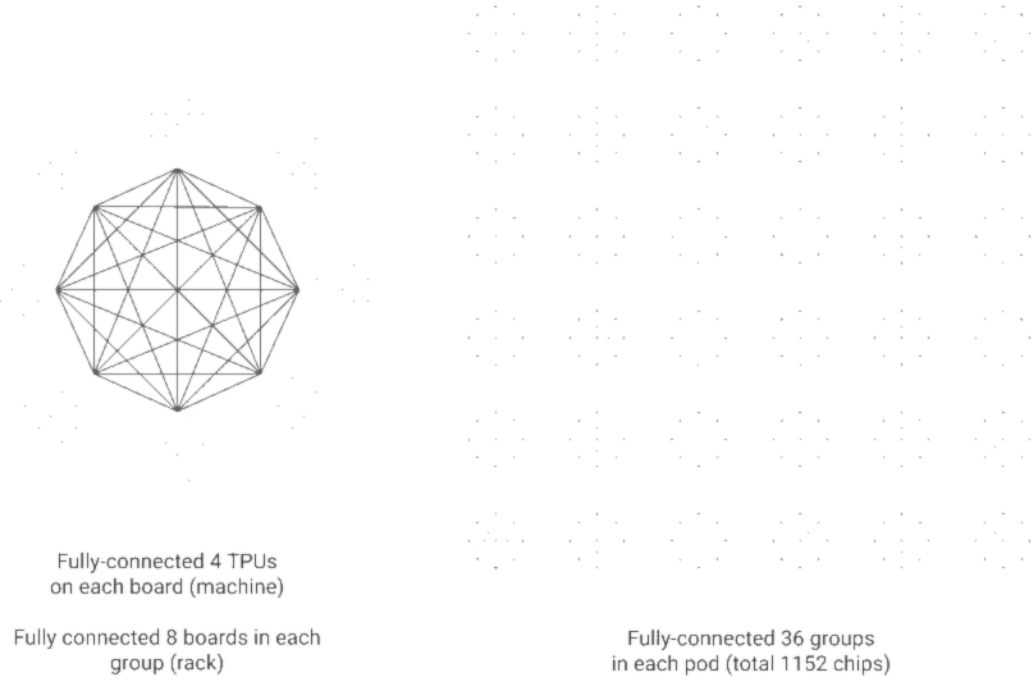
Google's newly announced eighth-generation TPU lineup has two purpose-built chips: TPU 8t for training and TPU 8i for inference. This is the first time Google has separated training and inference into distinct architecture designs rather than shipping one architecture tuned for both. TPU 8t keeps the 3D torus lineage alive for scale-up, but TPU 8i replaces the torus with a new topology called "Boardfly." The name nods to dragonfly-style high-radix network designs long used in supercomputing. Boardfly uses a flatter, hierarchical fabric of high-radix switches rather than a nearest-neighbor mesh. The Boardfly topology affects networking attachment capex per chip.

谷歌最新发布的第八代 TPU 产品线包含两款专用芯片：用于训练的 TPU 8t 和用于推理的 TPU 8i。这是谷歌首次将训练和推理分离为不同的架构设计，而非推出一种兼顾两者的统一架构。TPU 8t 延续了 3D 环形拓扑用于扩展，而 TPU 8i 则用名为"Boardfly"的新拓扑取代了环形拓扑。该名称借鉴了超级计算中长期使用的高基数蜻蜓式网络设计。Boardfly 采用更扁平、分层的多基数交换结构，而非最近邻网格。Boardfly 拓扑影响了每颗芯片的网络连接资本支出。

The payoff is a network diameter cut by more than 50% versus a similarly sized 3D torus, roughly 16 hops down to about 7 hops at comparable scale in the 1,024 to 1,152-chip range. Fewer hops mean materially lower tail latency on collective operations, and that matters enormously once you are routing tokens across MoE layers or

running multi-turn agentic workloads where every extra hop compounds into user-visible latency. TPU 8i backs this up with 19.2 Tb/s of ICI bandwidth, double the prior generation, and 384 MB of on-chip SRAM, 3x the previous generation, sized specifically to hold the KV cache of reasoning and agentic models on-chip rather than round-tripping to HBM.

其回报是，与同等规模的 3D 环形拓扑相比，网络直径缩短超过 50%，在 1024 至 1152 芯片范围内，跳数从约 16 跳降至约 7 跳。更少的跳数意味着集合通信操作上的尾部延迟显著降低，而当你跨 MoE 层路由 token 或运行多轮智能体工作负载时，这一点至关重要——每增加一跳都会累积成用户可感知的延迟。TPU 8i 以 19.2 Tb/s 的 ICI 带宽（为上一代的两倍）和 384 MB 的片上 SRAM（为上一代的 3 倍）为此提供支撑，其容量专门设计用于将推理和智能体模型的 KV 缓存保留在片上，而非往返于 HBM 之间。



Source: Google 来源：谷歌

None of this hardware headroom pays off on its own, though. Turning a lower-diameter network and a bigger on-chip cache into cost-per-token gains still requires the software stack to catch up. The next section covers the roadmap for doing so, including speculative decoding, prefill-decode disaggregation, and the broader model support needed to take full advantage of TPUv8 in production.

然而，这些硬件余量本身并不会自动转化为收益。要将更低直径的网络和更大的片上缓存转化为每 token 成本优势，仍需软件栈同步跟进。下一节将介绍实现这一目标的路线图，包括投机解码、预填充与解码分离，以及在生产环境中充分利用 TPUv8 所需的更广泛的模型支持。

Next Steps in Laying the TPU Foundation

奠定 TPU 基础的后续步骤

These preview results establish a starting point for further optimization on the same hardware. We expect performance to improve as TPU externalization continues, with substantial work still needed across the software stack.

这些预览结果为在同一硬件上的进一步优化奠定了起点。随着 TPU 外部化的持续推进，我们预计性能将不断提升，但软件栈方面仍有大量工作有待完成。

Enabling & Optimizing Spec Decoding (MTP)

启用并优化推测解码（MTP）

The first area the Google team needs to focus on is on optimizing speculative decoding. A cheap, small drafter guesses several tokens ahead, the main model verifies all of them in a single forward pass, and any guess that doesn't match is thrown away. Everything that survives is identical to what the model would have produced on its own. Speculative decoding is lossless, with zero quality degradation.

谷歌团队需要关注的首要领域是优化推测解码。一个廉价的小型草稿模型会提前猜测多个 token，主模型在单次前向传播中验证所有这些猜测，任何不匹配的猜测都会被丢弃。所有保留下来的结果与模型自行生成的结果完全一致。推测解码是无损的，质量零下降。

The reason it works is that decode is bandwidth bound, not compute bound. To emit a single token for a single user, you stream the entire model's weights out of HBM. That weight read is the dominant cost, and the cost barely changes whether you verify one token or five. The matrix units sit mostly idle while the memory system does all the work. Speculative decoding spends that idle compute to amortize one very expensive weight read across several tokens, which is why methods that draft multiple tokens at once, like MTP or DSpark, are so cheap and so effective.

其工作原理在于，解码过程受带宽限制，而非计算限制。要为单个用户生成单个 token，需要将整个模型的权重从 HBM 中流式读出。权重读取是主要成本，而无论验证一个 token 还是五个 token，成本几乎不变。矩阵单元大多处于空闲状态，而内存系统承担了全部工作。推测解码利用这些空闲计算资源，将一次昂贵的权重读取分摊到多个 token 上，这正是 MTP 或 DSpark 等多 token 同时草拟的方法如此廉价且高效的原因。



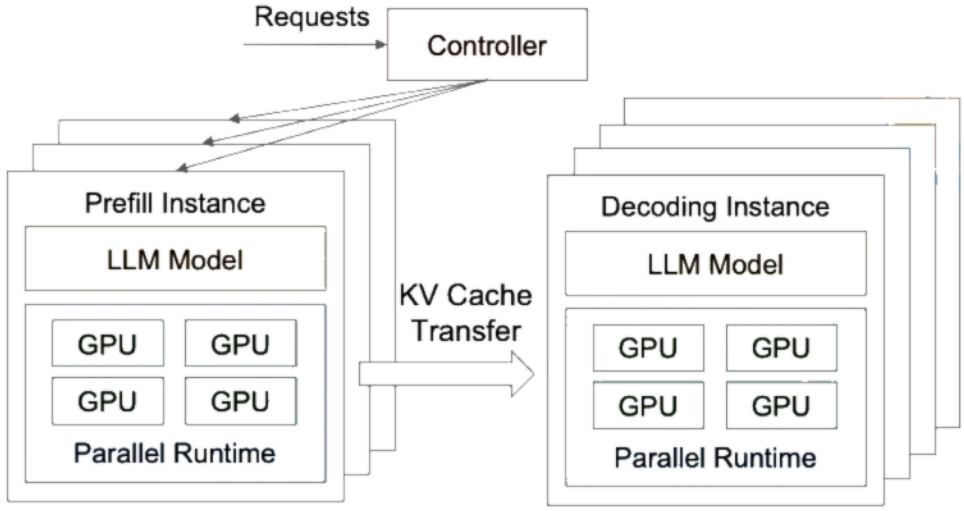
Source: vLLM 来源：vLLM

Prefill-Decode Disaggregation & KV-cache Offloading

预填充-解码分离与 KV 缓存卸载

Prefill-decode (PD) disaggregation is another optimization the TPU team is working to externalize. It separates prefill and decode across distinct TPU pools, allowing each pool to be tuned and scaled independently to match the workload.

预填充-解码（PD）分离是 TPU 团队正在推进外部化的另一项优化。它将预填充和解码分配到不同的 TPU 池中，使每个池能够根据工作负载独立调整和扩展。



Source: DistServe 来源：DistServe

Google has been running PD disaggregation internally for Gemini serving for ages, but work to externalize it began only a couple of months ago. That work includes TPU support in llm-d and the open-sourcing of TPU-Sync (formerly TPU-raiden), Google's disaggregated KV-cache transfer library. TPU-Sync works natively with JAX and the native TorchTPU stack, and can perform zero-copy transfers by extracting native PJRTBuffer hardware descriptors. With disaggregated PD, we believe TPUv7 can be a

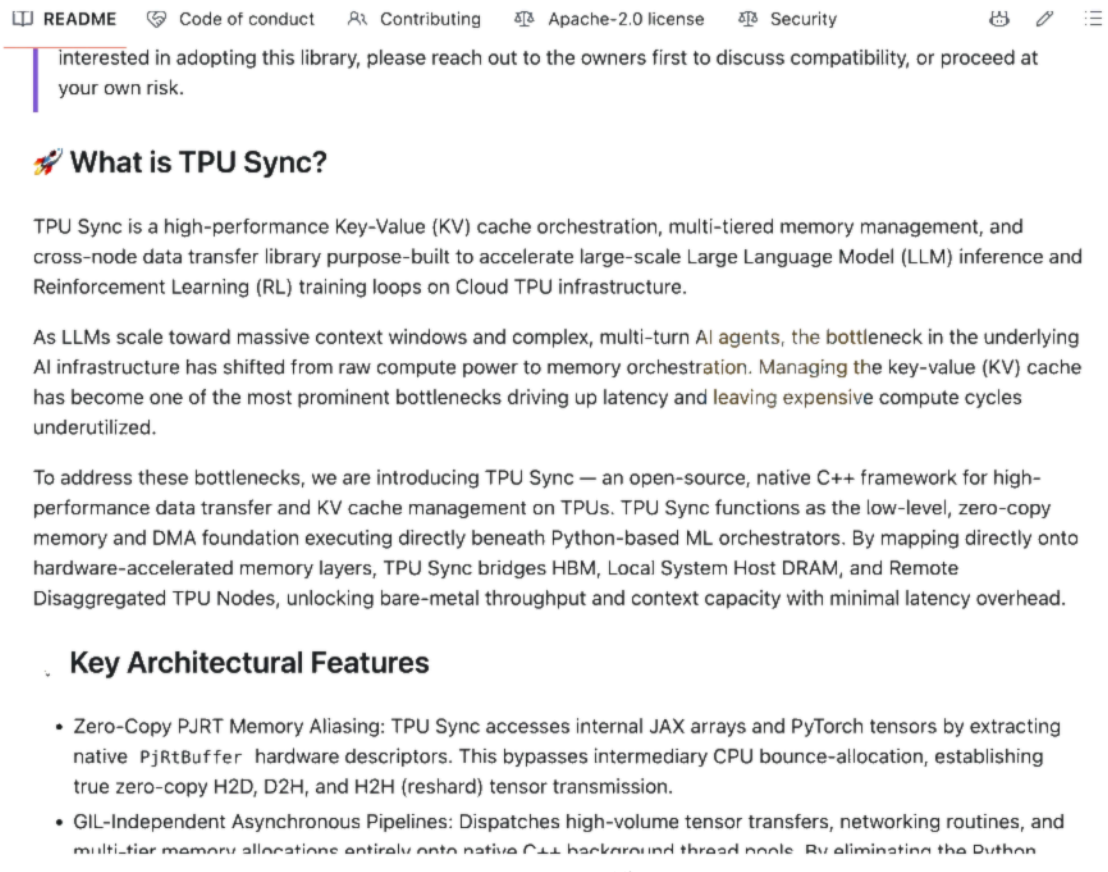
strong competitor to GB200/GB300 and even beat them on performance per dollar.

谷歌在 Gemini 服务中内部运行 PD 分离已有多时，但将其外部化的工作仅始于几个月前。这项工作包括在 llm-d 中支持 TPU，以及开源 TPU-Sync（前身为 TPU-raiden），这是谷歌的分离式 KV 缓存传输库。TPU-Sync 原生支持 JAX 和原生 TorchTPU 栈，并可通过提取原生 PJRTBuffer 硬件描述符实现零拷贝传输。借助分离式 PD，我们相信 TPUv7 可以成为 GB200/GB300 的有力竞争者，甚至在每美元性能上超越它们。

SemiAnalysis is a reader-supported publication. To receive new posts and support our work, consider becoming a subscriber.

TPU-Sync also supports native TPU KV-cache DRAM offloading, which is needed for large models and medium-to-large batch sizes, where HBM can no longer hold the KV cache for all users. It is great to see Google externalizing its DRAM offloading optimizations to the public as well.

TPU-Sync 还支持原生 TPU KV 缓存 DRAM 卸载，这对于大型模型和中到大批次大小是必需的，因为在这些情况下 HBM 无法再容纳所有用户的 KV 缓存。很高兴看到谷歌也将其 DRAM 卸载优化对外公开。


README Code of conduct Contributing Apache-2.0 license Security

interested in adopting this library, please reach out to the owners first to discuss compatibility, or proceed at your own risk.

🚀 What is TPU Sync?

TPU Sync is a high-performance Key-Value (KV) cache orchestration, multi-tiered memory management, and cross-node data transfer library purpose-built to accelerate large-scale Large Language Model (LLM) inference and Reinforcement Learning (RL) training loops on Cloud TPU infrastructure.

As LLMs scale toward massive context windows and complex, multi-turn AI agents, the bottleneck in the underlying AI infrastructure has shifted from raw compute power to memory orchestration. Managing the key-value (KV) cache has become one of the most prominent bottlenecks driving up latency and leaving expensive compute cycles underutilized.

To address these bottlenecks, we are introducing TPU Sync — an open-source, native C++ framework for high-performance data transfer and KV cache management on TPUs. TPU Sync functions as the low-level, zero-copy memory and DMA foundation executing directly beneath Python-based ML orchestrators. By mapping directly onto hardware-accelerated memory layers, TPU Sync bridges HBM, Local System Host DRAM, and Remote Disaggregated TPU Nodes, unlocking bare-metal throughput and context capacity with minimal latency overhead.

Key Architectural Features

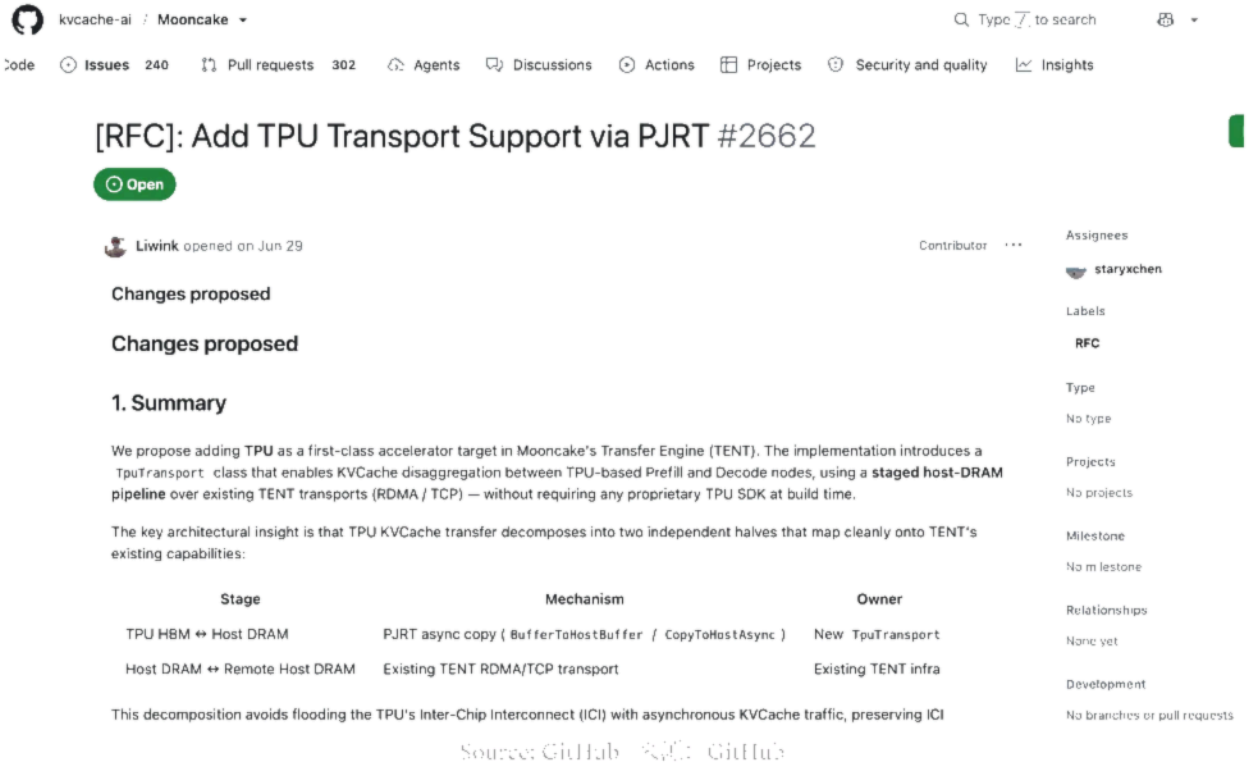
- Zero-Copy PJRT Memory Aliasing: TPU Sync accesses internal JAX arrays and PyTorch tensors by extracting native `PjRtBuffer` hardware descriptors. This bypasses intermediary CPU bounce-allocation, establishing true zero-copy H2D, D2H, and H2H (reshard) tensor transmission.
- GIL-Independent Asynchronous Pipelines: Dispatches high-volume tensor transfers, networking routines, and multi-tier memory allocations entirely onto native C++ background thread pools. By eliminating the Python

Source: GitHub

Google is externalizing its native TPU offloading stack and supporting the industry-standard Mooncake Store offloading library along with Mooncake Store's DRAM P2P

pooling support. We believe Mooncake Store support will be implemented using the primitives in tpu-sync.

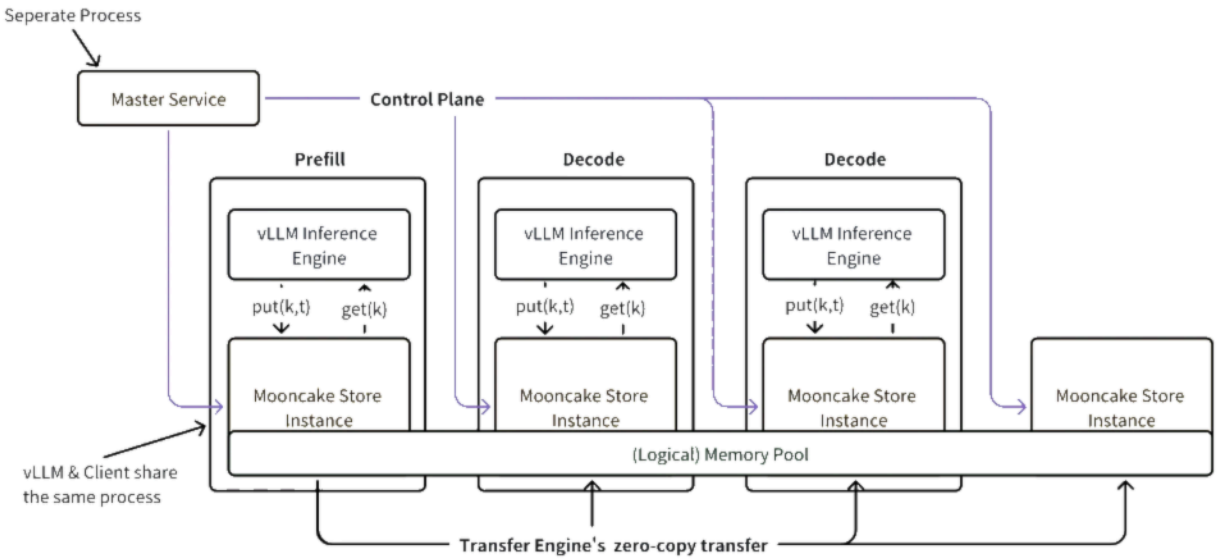
谷歌正在将其原生 TPU 卸载栈外部化，并支持行业标准的 Mooncake Store 卸载库以及 Mooncake Store 的 DRAM P2P 池化支持。我们相信 Mooncake Store 的支持将使用 tpu-sync 中的原语来实现。



Source: GitHub PR: GitHub

With P2P pooling, the KV-cache storage on each TPU host is aggregated into a single logical memory pool, so a TPU on any server can access KV cache from any other server. This unifies memory contributions from multiple nodes into one shared logical pool.

通过 P2P 池化，每个 TPU 主机上的 KV 缓存存储被聚合到一个单一的逻辑内存池中，因此任何服务器上的 TPU 都可以访问来自其他服务器的 KV 缓存。这将多个节点的内存贡献统一到一个共享的逻辑池中。



Source: Mooncake Mooncake

Furthermore, Mooncake Store pools NVMe storage from multiple servers into a single logical pool in addition to supporting traditional distributed filesystem backends like WEKA/Vast.

此外，Mooncake Store 除了支持传统的分布式文件系统后端（如 WEKA/Vast）外，还将多台服务器的 NVMe 存储整合到一个逻辑池中。

AgentX TPU

KV-cache offloading is especially important for long-context, multi-turn agentic workloads, where optimized KV-cache storage can enable high KV-cache hit rates.

KV 缓存卸载对于长上下文、多轮代理型工作负载尤为重要，优化的 KV 缓存存储可以实现高 KV 缓存命中率。

At a high level, an agentic workload is characterized by four elements:

从高层次来看，代理型工作负载具有四个特征：

1. Multi-turn: a session includes tens or hundreds of interactions between the user and assistant, compared with a handful in a chatbot scenario. These workloads combine long contexts and high prefill reuse with sub-agent bursts and numerous tool calls.

   多轮交互：一个会话包含用户与助手之间的数十次或数百次交互，而聊天机器人场景中只有少数几次。这些工作负载将长上下文和高预填充复用与子代理突发及大量工具调用相结合。

2. Long context: system prompts, tool definitions, and the large number of turns make context accumulate quickly.

长上下文：系统提示、工具定义以及大量轮次使得上下文迅速累积。

3. High prefix reuse: since the conversation progresses linearly, where output from turn n-1 is concatenated to turn n (typically), most context can be served from KV cache rather than recomputed (this depends on the amount of storage available to store KV tensors). As n grows, the ratio of cached input relative to uncached typically tends towards 1.

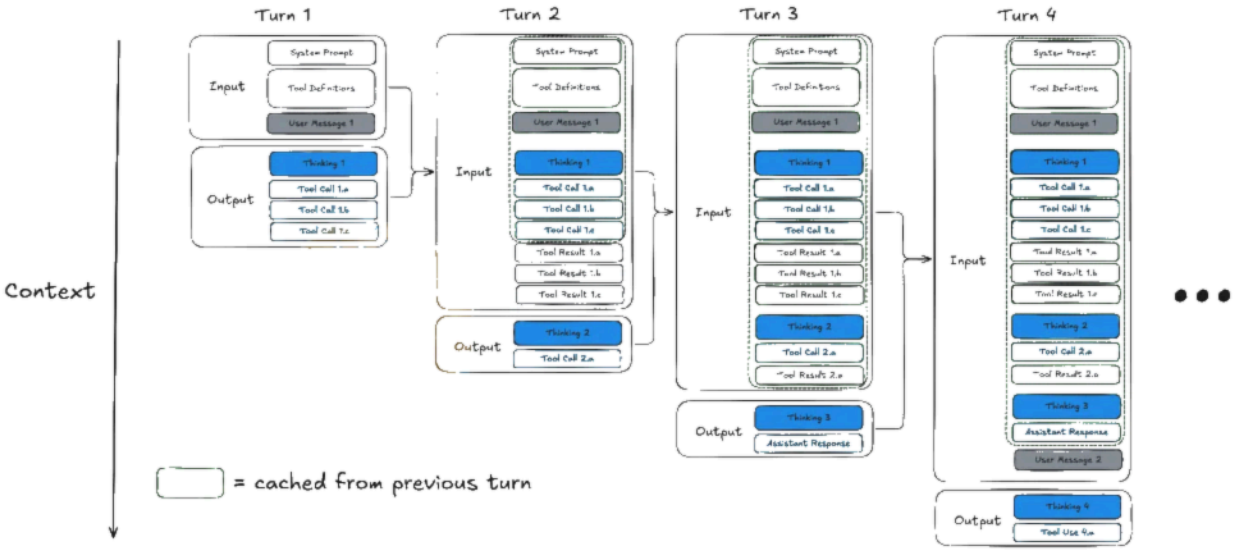
   高前缀复用：由于对话按线性方式推进，第 n-1 轮的输出通常与第 n 轮拼接，大部分上下文可以从 KV 缓存中提供，而无需重新计算（这取决于可用于存储 KV 张量的存储量）。随着 n 的增长，缓存输入与未缓存输入的比例通常趋向于 1。

4. Sub-agent bursts: a session launches multiple short-lived sub-agents with fresh context, which create bursty KV-cache patterns.

   子代理突发：一个会话会启动多个具有全新上下文的短期子代理，从而产生突发性的 KV 缓存模式。

We are hearing that many Google TPU customers are already asking for AgentX performance results, and we are excited to announce that we are working to bring TPUs to AgentX too!

我们听说许多谷歌 TPU 客户已经在询问 AgentX 的性能结果，我们很高兴地宣布，我们正在努力将 TPU 也引入 AgentX！



Source: DeepSeek, SemiAnalysis

来源：DeepSeek，SemiAnalysis

Ironwood has no native FP4, so for now the fair apples-to-apples comparison is against FP8 Blackwell: FP8 vs FP8 carries no quality loss, whereas FP4 vs FP8 introduces a quality difference on the FP4 side. Google's TPUv8i does have native FP4 acceleration, so when we bring up TPUv8i on InferenceX/AgentX, we will compare

FP4 to FP4.

Ironwood 没有原生 FP4 支持，因此目前公平的同类比较是针对 FP8 Blackwell：FP8 对 FP8 没有质量损失，而 FP4 对 FP8 会在 FP4 侧引入质量差异。谷歌的 TPUv8i 确实具有原生 FP4 加速能力，因此当我们在 InferenceX/AgentX 上启用 TPUv8i 时，我们将进行 FP4 对 FP4 的比较。

SemiAnalysis is a reader-supported publication. To receive new posts and support our work, consider becoming a subscriber.

We strongly believe that building native vLLM and SGLang support on native TorchTPU's stable foundation is the right direction for TPU externalization. TorchTPU is meant to replace the previous TorchAX stack, which will soon be deprecated. Google will next work to enable Kimi K3 and GLM5.3 on single-turn and/or agentic workloads like AgentX, along with Google's own open-weight models such as Gemma4. Once a few models are running on the TorchTPU stack, adding new ones will become far easier, and TPU support will land much closer to day 0.

我们坚信，在原生 TorchTPU 的稳定基础上构建原生 vLLM 和 SGLang 支持，是 TPU 外部化的正确方向。TorchTPU 旨在取代即将弃用的 TorchAX 技术栈。谷歌下一步将致力于在单轮和/或代理型工作负载（如 AgentX）上启用 Kimi K3 和 GLM5.3，同时支持谷歌自家的开放权重模型（如 Gemma4）。一旦少数模型在 TorchTPU 技术栈上运行，添加新模型将变得容易得多，TPU 支持也将更接近第 0 天落地。

Rome wasn't built on day 0, so we shouldn't expect the externalization of TPUs to happen instantly. But we strongly believe that it is happening at an extremely rapid pace.

罗马不是一天建成的，因此我们不应期望 TPU 的外部化能瞬间实现。但我们坚信，这一进程正在以极快的速度推进。

TPU Cost Effective Total Cost of Ownership

TPU 成本效益总拥有成本

Our full BOM and TCO estimates for TPUv7 are enumerated in the SemiAnalysis AI TCO Model. A structural overview is below.

我们针对 TPUv7 的完整物料清单（BOM）和总拥有成本（TCO）估算已详列于 SemiAnalysis AI TCO 模型中。以下为结构概览。

We present a truncated view of the TPUv7 2D Torus internal BOM and TCO below, for comparison against other SKUs on a single-node basis. On a server cost basis, we estimate TPUv7 costs less than half of a Blackwell-class rack-scale system, and it also lands lower on capex per watt than any Blackwell or GB configuration. Both feed directly into its chip-hour economics for Google's internal workloads.

我们在下方展示了 TPUv7 2D 环形拓扑内部 BOM 和 TCO 的截取视图，以便在单节点基础上与其他 SKU 进行比较。在服务器成本方面，我们估算 TPUv7 的成本不到 Blackwell 级机架规模系统的一半，并且在每瓦特资本支出上也低于任何 Blackwell 或 GB 配置。这两点直接促成了谷歌内部工作负载在芯片小时经济性上的优势。

Cluster Configuration	Unit	TPU v7 - 2D Torus - Internal
Customer Type		**Hyperscaler**
Release Date		**1Q25**
Cluster Capital Costs		
GPU Provider Compute Tray Content	USD	[redacted]
Other Compute Tray Content	USD	
Total Compute Tray Content (Ex DRAM and NAND)	USD	
DRAM Costs	USD	
NAND Costs	USD	
Switch Tray	USD	
Scale-up (excluding Switch tray)	USD	
All other Server	USD	
Standard OEM/ODM Markup	USD	
Reseller/Additional Markup Observed	USD	
Server Cost	**USD**	
Server Service	**USD**	**$64,000**
Networking Cost	**USD**	[redacted]
Storage Cost	**USD**	**$39,082**
Software Licenses and Other Costs	**USD**	**$13,630**
Other Installation	**USD**	**$64,000**
Total Upfront Cluster Capex, per Server (Pre-Margin)	**USD**	[redacted]
Total Upfront Cluster Capex, per Server	**USD**	

Source: SemiAnalysis AI TCO Model

来源：SemiAnalysis AI TCO 模型

On a TCO per chip-hour basis, TPUv7 sits well below the owning-hyperscaler cost of a B200 or B300.

在每芯片小时 TCO 方面，TPUv7 远低于拥有超大规模数据中心运营商所使用 B200 或 B300 的成本。

Cluster Configuration	Unit	TPU v7 - 2D Torus - Internal
Customer Type		**Hyperscaler**
Capital Cost per Unit, per Hour	USD/hr/GPU	$0.68
Operating Cost per Unit, per Hour	USD/hr/GPU	$0.34
Total Cost per Unit per Hour	**USD/hr/GPU**	**$1.02**
Capital Cost as % of Total Ownership Cost	%	66.5%

Source: SemiAnalysis AI TCO Model

来源：SemiAnalysis AI TCO 模型

Recommend SemiAnalysis to your readers

向您的读者推荐 SemiAnalysis

Bridging the gap between the world's most important industry, semiconductors, and business.

弥合全球最重要产业——半导体——与商业之间的鸿沟。



68 Likes 68 个赞 · 6 Restacks 6 次转帖

Previous 上一篇

Discussion about this post

关于此帖的讨论

Comments 评论 **Restacks 转发**

Write a comment...

© 2026 Dylan Patel · Privacy · Terms · Collection notice
Substack is the home for great culture