
ASAP: an Agentic Solution to Auto-optimize Performance of Large-Scale LLM Training

Yuran Ding^{1,2} Xinwei Chen¹ Xiaofan Zhang¹ Zongwei Zhou^{3*}
¹Google ²University of Maryland, College Park ³Meta
{dingyuran, xinweichen, xiaofanz}@google.com, zongwei@meta.com

Abstract

Optimizing large-language model (LLM) training on distributed domain-specific accelerator systems presents significant challenges due to its complex optimization space. Existing optimization methods, however, rely on time-consuming manual tuning or resource-intensive black-box searches, which struggle to keep pace with the rapidly evolving LLM domain, leading to slow development and underutilized resources. To address this, we introduce **ASAP**, an **A**gentic **S**olution to **A**uto-optimize **P**erformance of **L**arge-Scale **L**LM **T**raining. It is a multi-agent system, featuring Coordinator, Analyzer, and Proposal agents, which integrates LLM reasoning with insights from performance profiling tools, roofline analysis, and a knowledge base of best practices and successful past optimizations from human experts. Our proposed design can automate the diagnosis of performance bottlenecks and recommend optimized sharding configurations with reasoning, thus effectively improving the efficiency of distributed LLM training. Experiments have shown that the ASAP-generated sharding configurations can contribute up to 28% training step time reduction and $1.43\times$ throughput improvement. When combined with additional optimization from human experts, throughput can be further increased to $2.58\times$. The proposed ASAP promises to provide a scalable and explainable methodology for AI-assisted performance engineering in large-scale LLM training.

1 Introduction

The cost of training LLMs is increasing exponentially. Given the targeted model, hardware budget, and development timeframe, identifying the most-efficient system configuration becomes the first critical task to perform efficient training. It requires an in-depth understanding of system-level optimization, such as data, model, and sequence parallelism, to select the most efficient sharding combination. (Sun et al. [2019], Shoeybi et al. [2019]).

This, however, is a complex and time-consuming task. Engineers typically resort to a laborious trial-and-error cycle: running experiments, profiling performance, analyzing execution traces for bottlenecks like memory limits or communication overheads, and iteratively refining sharding strategies. This manual approach is often slow and yields suboptimal configurations.

While performance analysis tools and analytical models provide deep insights, they require extensive expertise to interpret. Early automated methods like black-box optimization (Golovin et al. [2017]) struggle with the vast and complex search space of sharding configurations, making them resource-intensive and slow to converge. In contrast, an agentic, reasoning-based approach can leverage a knowledge base of performance engineering principles to diagnose issues and propose solutions efficiently and explainably.

Recent work has leveraged machine learning to build performance models and improve search efficiency (Wen et al. [2024], Cha et al. [2024]). The rise of LLMs has also inspired agentic systems

*Work done while at Google.

for performance engineering, such as automated GPU code transformation (Damani et al. [2024]), code optimization for tensor accelerators (Hong et al. [2025]), and multi-agent frameworks for complex ML tasks (Gandhi et al. [2025], Liu et al. [2025], Novikov et al. [2025]). However, these advancements have not yet produced a specialized agent for the nuanced task of optimizing sharding strategies for large-scale model training. Such an agent should deeply integrate with performance analysis tools, understand theoretical performance bounds, and leverage a knowledge base of best practices and past optimizations to propose explainable configurations.

To address this gap, we introduce ASAP: an Agentic Solution to Auto-optimizing Performance of Large-Scale LLM Training. ASAP is a multi-agent framework designed to automate this optimization process, composed of:

- **Coordinator Agent:** Orchestrates the workflow, manages agent interactions, and processes user requests.
- **Analyzer Agent:** Diagnoses performance bottlenecks by analyzing data from profiling tools and experiment logs. It identifies compute, memory, or communication inefficiencies.
- **Proposal Agent:** Generates optimized sharding configurations with detailed reasoning. It queries a knowledge base of best practices and a database of past optimizations to inform its recommendations.
- **Sharding Memory:** Provides persistent storage for logs, tool calls, and agent responses to ensure context and traceability.

By integrating LLM-based reasoning with specialized performance analysis tools and a curated knowledge base, ASAP automates the diagnosis of sharding issues and the generation of optimized, explainable configurations. Our system aims to significantly reduce manual effort, shorten development cycles, and improve hardware efficiency, establishing a scalable methodology for AI-driven performance engineering.

2 ASAP Agent

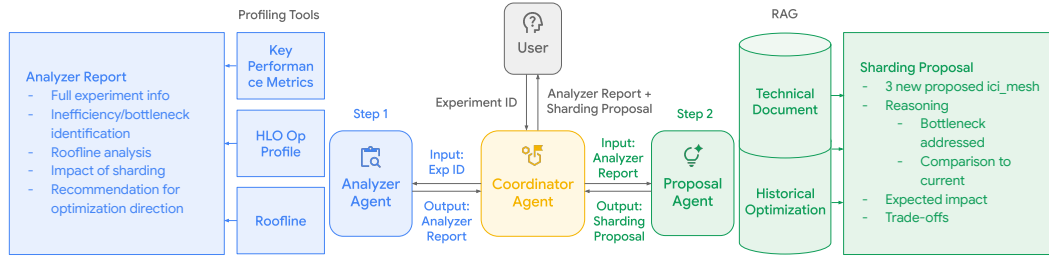


Figure 1: Overview of the ASAP Agentic Performance Optimization Framework

The ASAP Agent is a multi-agent system designed to automate the optimization of sharding configurations for large-scale LLM training. Built on Google’s Agent Development Kit (ADK)², it integrates LLM-based reasoning with specialized tools and custom prompts. The system is invoked via the command line with an experiment ID³ and consists of the following components:

- **Coordinator Agent:** The central orchestrator, which receives an experiment ID as input. It directs the workflow, routes data between agents, and manages the overall optimization task.
- **Analyzer Agent:** Diagnoses performance bottlenecks using raw data from the experiment configuration and the Xprof profiling API⁴. It fetches key performance metrics (KPIs)⁵, HLO operation profiles⁶, and on-device roofline analysis data⁷. By correlating high-level metrics

²Google Agent Development Kit: <https://google.github.io/adk-docs/>

³The experiment ID is a key to retrieve an experiment’s full record, including its sharding configuration, hyperparameters, code version, and system environment.

⁴Information on Xprof Profiling API: <https://cloud.google.com/tpu/docs/profile-tpu-vm>

⁵KPIs include high-level indicators like accelerator busy time, training step time, and goodput, instrumented from framework components.

⁶The HLO Op Profile provides detailed statistics for each operation (e.g., matrix multiplication, all-reduce), including execution time, FLOPs utilization, memory bandwidth usage, and tensor shapes, helping to identify inefficient operations.

⁷Raw roofline data includes an operation’s total FLOPs, bytes accessed, and execution time, which are used to determine if it is compute- or memory-bound.

with operation-level details, the agent identifies compute, memory, or communication bottlenecks. It then pinpoints the specific operations responsible and reports its findings to guide the optimization process.

- **Proposal Agent:** Generates optimized sharding configurations using a Retrieval-Augmented Generation (RAG) workflow. First, it analyzes the report from the Analyzer Agent to understand the specific bottlenecks and the current sharding strategy. Next, it queries a database of *past optimizations*⁸ to find successful solutions from similar model and hardware setups. Informed by this historical context, it queries a *document knowledge base* (adapted from *How to Scale Large Models*⁹) for relevant technical principles. Finally, it synthesizes this information to propose three distinct sharding configurations. Each proposal includes a detailed justification, its expected performance impact, a discussion of trade-offs, and citations to the supporting knowledge base or historical examples.
- **Sharding Memory:** A persistent, file-based worklog that captures all tool calls, LLM responses, and user inputs. This module facilitates debugging, maintains conversational context, and ensures traceability.

This automated analysis-and-proposal loop allows ASAP to systematically explore and recommend high-performance sharding configurations. The agent’s proposals, complete with detailed justifications, can then be reviewed by an engineer or evaluated on hardware, significantly accelerating the optimization process compared to manual methods.

3 Experiment

We evaluated the ASAP Agent’s ability to diagnose and optimize three distinct large-scale model training scenarios on TPU v5p, v6e, and v5e platforms. The evaluation used the full multi-agent framework (Coordinator, Analyzer, Proposal) powered by Gemini Pro and a RAG system, which queries a knowledge base of scaling techniques and historical optimizations.

For each scenario, the agent was given an experiment ID and Xprof profiling data (KPI metrics, operation-level profiles, and roofline analysis in JSON strings) from a baseline run. We then qualitatively compared the agent’s proposed sharding configurations against those developed by human performance experts. A sharding configuration (*ici_mesh*) is defined by its data, model, replica, and sequence parallelism dimensions. The goal was to assess if the agent’s proposals aligned with expert-derived solutions.

3.1 Experiment 1: Compute-Bound Workload on TPU v5p-512

Initial State: The baseline config is an *ici_mesh* of `{‘model’: 8, ‘data’: 8, ‘seq’: 8}`.

Agent Analysis: The Analyzer Agent reported a very high device duty cycle (99.73%) with low communication overhead (2.57%). Performance profiling traces indicated the top time-consuming operations were primarily compute-bound.

Agent Proposal: The top recommendation was to change the sharding to `{‘model’: 8, ‘data’: 16, ‘seq’: 4}`. The agent reasoned that increasing data parallelism from 8 to 16, while reducing sequence parallelism from 8 to 4, would better utilize the compute units and potentially reduce memory pressure from a large sequence length.

Outcome: The agent’s proposed sharding configuration precisely matched the human-engineered solution, significantly decreasing TPU compute time (by 28%) and improving FLOPs and HBM utilization. This specific case was present in the agent’s historical optimization database. The agent expressed high confidence (around 85-90%) in its proposal, defending its reasoning when challenged (see Appendix A.1.3).

3.2 Experiment 2: HBM-Bound Workload on TPU v6e-16

Initial State: The baseline used `{‘data’: 4, ‘model’: 1, ‘replica’: 1, ‘seq’: 4}`.

⁸This database contains records of previous optimizations, including before-and-after configurations, performance profiling tool (Xprof) links, and a summary of the changes and their impact.

⁹How To Scale Large Models: <https://jax-ml.github.io/scaling-book/>

Agent Analysis: The Analyzer Agent identified a clear High Bandwidth Memory (HBM) bottleneck. Despite high duty cycle, the most time-consuming operations (including collectives like `all-reduce` and memory-bound fusions) were limited by HBM bandwidth, attributed to model replication.

Agent Proposal: The agent’s top proposal was `{‘replica’: 1, ‘data’: 4, ‘model’: 4, ‘seq’: 1}`. The reasoning was to introduce 4-way model parallelism to shard the model, significantly reducing the HBM footprint per chip.

Outcome: The agent’s proposed configuration was identical to the human-engineered solution, which improved throughput by $1.43\times$. This specific optimization was novel (not in the historical database), demonstrating the agent’s successful generalization from knowledge-base documents on sharding principles. When challenged, the agent confidently (around 90%) maintained its stance, reasoning that addressing the HBM bottleneck through model parallelism was the most critical first step (see Appendix A.2.3).

3.3 Experiment 3: Communication-Bound Workload on TPU v5e-256

Initial State: The baseline employed `{‘data’: 4, ‘model’: 4, ‘seq’: 16}`.

Agent Analysis: The Analyzer Agent found that despite a near-100% duty cycle, the primary bottleneck was collective communication. The top 5 most time-consuming operations were all communication operations (e.g., `all-reduce`, `all-reduce-scatter`, `collective-permute`), bound by memory bandwidth during data exchanges.

Agent Proposal: The agent recommended `{‘replica’: 1, ‘data’: 8, ‘seq’: 16, ‘model’: 2}`. The reasoning was to reduce model parallelism from 4 to 2, decreasing synchronization overhead, while increasing data parallelism from 4 to 8.

Outcome: The agent’s proposed `ici_mesh` matched the human expert’s final configuration. This specific sharding change, combined with an increased batch size, led to a $2.58\times$ increase in total throughput. The optimization was novel (not in the historical database), demonstrating the agent’s generalization ability. The agent consistently and confidently (85-90%) justified its proposal, explaining that reducing model parallelism was crucial for mitigating communication overhead (see Appendix A.3.3).

4 Conclusion

Across all three experiments, the proposed ASAP Agent consistently demonstrated the ability to:

- **Analyze** model and hardware architectures to accurately diagnose core performance bottlenecks (compute, HBM, or communication) across diverse platforms.
- **Synthesize** generalizable insights from expert examples and optimization principles to propose sharding configurations that replicated human solutions, then successfully defended these proposals with evidence-based rebuttals citing performance profiles and learned principles.
- **Articulate** clear, a priori justifications for its proposals, mapping recommended sharding changes directly to identified bottlenecks and its knowledge base.

Crucially, Experiments 2 and 3 highlighted the agent’s ability to *generalize* beyond its historical optimization database. By reasoning over analysis reports and technical documentation, the agent synthesized novel and correct sharding solutions. This ability to form coherent, evidence-backed rebuttals, particularly notable in these generalization tasks (see Appendices A.1.3, A.2.3, and A.3.3), underscores the robustness of its diagnostic and reasoning capabilities. These results suggest the ASAP Agent can effectively automate and accelerate the complex sharding optimization process with actionable, explainable proposals, potentially saving significant engineering effort and improving accelerator utilization.

5 Limitations & Future Work

Currently, the agent is limited by its dependence on accurate profiling data and its use of a single, global sharding configuration. Future work will enhance system robustness and support more sophisticated sharding techniques like per-layer sharding. We will also introduce an Evaluator Agent to create a closed feedback loop, enabling the agent to learn and refine its strategies through reinforcement learning and discover novel configurations. Additionally, the agent’s scope will expand to holistically optimize the trade-offs between on-device computation and network communication,

while incorporating other compiler-level optimizations such as operator fusion and memory layout selection.

6 Acknowledgement

The authors gratefully acknowledge the invaluable efforts and insights of Jake Ades, Deniz Altınbüken, Chidubem Arachie, Charles Chang, Samrat Ghosh, Ruobing Han, Jesse Lu, Hao Luo, Martin Maas, Michael Moffitt, Tayo Oguntebi, Tengyu Sun, Gil Tabak, Chao Wang, Arissa Wongpanich, Tzu-Wei Yang, and Zhipeng Yan (*in alphabetical order by last name*). Their support was crucial for designing and developing the ASAP agentic framework, and ensuring its operational success by connecting it to the full stack of profiling tools and the comprehensive knowledge base.

References

- JooHyoung Cha, Munyoung Lee, Jinse Kwon, Jubin Lee, Jemin Lee, and Yongin Kwon. ML² Tuner: Efficient code tuning via multi-level machine learning models. *arXiv preprint arXiv:2411.10764*, 2024.
- Sana Damani, Siva Kumar Sastry Hari, Mark Stephenson, and Christos Kozyrakis. Warpdrive: An agentic workflow for ninja gpu transformations. In *NeurIPS Mach. Learn. Syst. Workshop*, 2024.
- Shubham Gandhi, Manasi Patwardhan, Lovekesh Vig, and Gautam Shroff. Budgetmlagent: A cost-effective llm multi-agent system for automating machine learning tasks. In *Proceedings of the 4th International Conference on AI-ML Systems*, AIMSys ’24, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711619. doi: 10.1145/3703412.3703416. URL <https://doi.org/10.1145/3703412.3703416>.
- Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Elliot Karro, and D. Sculley, editors. *Google Vizier: A Service for Black-Box Optimization*, 2017. URL <http://www.kdd.org/kdd2017/papers/view/google-vizier-a-service-for-black-box-optimization>.
- Charles Hong, Sahil Bhatia, Alvin Cheung, and Yakun Sophia Shao. Autocomp: Llm-driven code optimization for tensor accelerators. *arXiv preprint arXiv:2505.18574*, 2025.
- Zexi Liu, Jingyi Chai, Xinyu Zhu, Shuo Tang, Rui Ye, Bo Zhang, Lei Bai, and Siheng Chen. ML-agent: Reinforcing llm agents for autonomous machine learning engineering. *arXiv preprint arXiv:2505.23723*, 2025.
- Alexander Novikov, Ngan Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.
- Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.
- Peng Sun, Yonggang Wen, Ruobing Han, Wansen Feng, and Shengen Yan. Gradientflow: Optimizing network performance for large-scale distributed dnn training. *IEEE Transactions on Big Data*, 8(2):495–507, 2019.
- Wei Wen, Quanyu Zhu, Weiwei Chu, Wen-Yen Chen, and Jiyan Yang. Cubicml: Automated ml for large ml systems co-design with ml prediction of performance. *arXiv preprint arXiv:2409.04585*, 2024.

A Detailed Experiment Results

This appendix provides the full analysis reports and agent-generated proposals for each of the three experiments, illustrating the agent’s diagnostic capabilities and reasoning process. The dialogue between the user and the coordinator_agent is presented to show the interaction flow.

A.1 Experiment 1: Compute-Bound Workload on TPU v5p-512

Baseline sharding {'model': 8, 'data': 8, 'seq': 8} was optimized by human engineers to {'model': 8, 'data': 16, 'seq': 4} in the code change.

The following is the user-agent dialogue to optimize the baseline experiment.

A.1.1 Initial State and Agent's Analysis

User: Please analyze training job id=xxxxxxxx (redacted due to confidentiality), Xprof session id = xxxxxxxx (redacted due to confidentiality).

Agent:

A. Full Experiment Details:

- **Extracted Sharding Parameters (Baseline Configuration):** {'model': 8, 'data': 8, 'seq': 8} for training.updater.partition. Other model components (vision encoder, decoder, decoder block) had null sharding.
- **Device Type:** TPU v5p-512
- **TPU Topology:** 8x8x8
- **Model Name:** (redacted due to confidentiality)
- **Model Version:** (redacted due to confidentiality)
- **Batch Size:** 256
- **Sequence Length:** 8192

C. Correlated Analysis of KPIs and HLO/Roofline: A. High-Level KPI Health Check:

- **Step Time:** 35656.45 ms
- **Device Duty Cycle:** 99.73%
- **Communication Overhead:** 2.57%
- **Initial Hypothesis:** The KPIs show a very high device duty cycle (99.73%), which suggests that the TPU is well-utilized and not bottlenecked by input/output or host-side processing. The low communication overhead (2.57%) indicates that collective operations are not a major contributor to the overall step time. The primary focus for optimization should be on the on-device computation itself.

B. Detailed HLO/Roofline Bottleneck Identification: (Note: the exact operation names are omitted here due to confidentiality) The top 5 time-consuming operations are:

1. **Operation:** Custom point-to-point transpose operation
 - **Category:** Custom-call
 - **% of Total Time:** 0.62%
 - **Bottleneck:** Unknown
 - **Roofline Efficiency:** 0.0%
2. **Operation:** Asynchronous collective completion
 - **Category:** Custom fusion
 - **% of Total Time:** 0.55%
 - **Bottleneck:** HBM **Roofline Efficiency:** 0.0017%
3. **Operation:** Custom point-to-point transpose operation
 - **Category:** Custom-call
 - **% of Total Time:** 0.53%
 - **Bottleneck:** Unknown
 - **Roofline Efficiency:** 0.0%
4. **Operation:** Custom matrix multiplication with scatter
 - **Category:** Custom-call
 - **% of Total Time:** 0.48%
 - **Bottleneck:** Compute
 - **Roofline Efficiency:** 74.14%

5. **Operation:** All-gather matrix multiplication

- **Category:** Custom-call
- **% of Total Time:** 0.45%
- **Bottleneck:** Compute
- **Roofline Efficiency:** 79.53%

C. Correlating KPI and HLO/Roofline:

- **High Device Duty Cycle:** The high duty cycle is confirmed by the HLO/Roofline analysis. The top operations are a mix of custom-calls and fusions, with several of the most time-consuming being compute-bound with high roofline efficiency. This indicates that the model is making good use of the TPU's computational resources.
- **Low Communication Overhead:** The All-Reduce operations are not in the top 5, and their total contribution to the step time is small, which aligns with the low communication overhead seen in the KPIs.
- **Sharding Configuration Impact:** The sharding configuration `{'model': 8, 'data': 8, 'seq': 8}` shows a high degree of model and data parallelism. This is likely contributing to the high device utilization, as the workload is being effectively distributed across the available cores. However, the presence of HBM-bound operations like "Asynchronous collective completion" suggests that the high degree of parallelism may be putting some pressure on memory bandwidth, although it is not currently the primary bottleneck.

D. Structured Analysis of Inefficiency: The analysis indicates a highly optimized workload with excellent device utilization. The primary inefficiency is the presence of several "custom-call" operations with "Unknown" bottlenecks and 0% roofline efficiency. These operations, particularly "custom point-to-point transpose" operations, are consuming a significant amount of time but their performance characteristics are not well-understood by the profiling tools. These are likely candidates for further investigation and optimization. The sharding configuration, with its 8-way model, 8-way data and 8-way sequence parallelism, is well-suited for this model and hardware, resulting in the high device utilization. However, the HBM-bound "Asynchronous collective completion" suggests that memory bandwidth could become a bottleneck if the model or batch size were to be scaled up further.

A.1.2 Agent's Proposals for Optimization

Agent: Here are three new sharding configurations designed to improve the performance of your model on the TPU v5p-512, along with the reasoning behind each proposal.

Proposal 1: Increase Data Parallelism, Reduce Sequence Parallelism

- **New ICI Mesh:** `{'model': 8, 'data': 16, 'seq': 4}`
- **Reasoning:**
 - **Addresses Compute and Unknown Bottlenecks:** The primary bottleneck is identified as a mix of compute-bound operations and custom-calls with "Unknown" bottlenecks. The current configuration has a balanced 8x8x8 sharding. This proposal shifts the balance to favor data parallelism (from 8 to 16) while reducing sequence parallelism (from 8 to 4). By increasing data parallelism, we can potentially improve the efficiency of the compute-bound operations by feeding the TPU more data in parallel. This can lead to better utilization of the compute units and a reduction in step time.
 - **HBM Impact:** Reducing sequence parallelism from 8 to 4 is expected to decrease HBM pressure. With a large sequence length of 8192, high sequence parallelism can lead to increased memory usage for activations. This change could alleviate the HBM-bound "Asynchronous collective completion" operation, even though it's not the primary bottleneck.
 - **Comparison to Current:** Compared to the current `{'model': 8, 'data': 8, 'seq': 8}` sharding, this configuration doubles the data parallelism at the expense of sequence parallelism. This is a strategic trade-off to address the compute-bound nature of the workload, as suggested by the high device duty cycle.

- **Expected Impact:**
 - **Step Time:** Decrease
 - **Duty Cycle:** Remain high
 - **Communication Overhead:** May see a slight increase due to more data parallelism, but the impact is expected to be minimal given the already low communication overhead.
- **Trade-offs:** A potential increase in communication overhead, although likely negligible.

Proposal 2: Increase Model Parallelism, Reduce Sequence Parallelism

- **New ICI Mesh:** {'model': 16, 'data': 8, 'seq': 4}
- **Reasoning:**
 - **Addresses Compute and Unknown Bottlenecks:** This proposal takes a different approach by significantly increasing model parallelism (from 8 to 16) while also reducing sequence parallelism (from 8 to 4). This strategy is often effective for very large models, as it can reduce the memory footprint of the model on each core and potentially improve the efficiency of the compute-bound operations within the model's layers.
 - **HBM Impact:** Similar to the first proposal, reducing sequence parallelism will help to reduce HBM pressure. The increased model parallelism will also distribute the model's parameters across more cores, which can further reduce memory pressure on individual cores.
 - **Comparison to Current:** This is a more aggressive change than the first proposal, doubling the model parallelism. This is a common strategy for very large models where reducing the memory footprint per core is critical for performance. This proposal is based on general knowledge of sharding large models.
- **Expected Impact:**
 - **Step Time:** Decrease
 - **Duty Cycle:** Remain high
 - **Communication Overhead:** May see a slight increase due to the increased model parallelism, but the impact is expected to be minimal.
- **Trade-offs:** A potential increase in communication overhead due to more frequent communication between model-parallel replicas.

Proposal 3: Prioritize Data Parallelism, Minimal Model Parallelism

- **New ICI Mesh:** {'model': 4, 'data': 16, 'seq': 8}
- **Reasoning:**
 - **Addresses Compute and Unknown Bottlenecks:** This proposal aggressively increases data parallelism to 16, halves the model parallelism to 4, and maintains the current sequence parallelism of 8. This is a good strategy to try if the model is not memory-bound but is compute-bound. By increasing data parallelism, we can potentially increase the throughput of the system.
 - **HBM Impact:** This configuration may increase HBM pressure due to the high data and sequence parallelism. However, since the primary bottleneck is compute, this trade-off might be acceptable if it leads to a significant reduction in step time.
 - **Comparison to Current:** This proposal is a significant departure from the current balanced sharding, and it prioritizes data parallelism over model and sequence parallelism. This is a good experiment to run to see if the model's performance is more sensitive to data parallelism than the other parallelism dimensions.
- **Expected Impact:**
 - **Step Time:** Decrease
 - **Duty Cycle:** Remain high
 - **Communication Overhead:** May see a slight increase due to the increased data parallelism.

- **Trade-offs:** A potential increase in HBM pressure. A potential increase in communication overhead.

A.1.3 Agent's Recommendation and Justification Dialogue

User: Which one of the proposals would you most recommend to start with?

Agent: I would most recommend starting with **Proposal 1:** {'model': 8, 'data': 16, 'seq': 4}. Here's why:

- **Directly Addresses the Primary Bottleneck:** The analysis indicates a compute-bound workload. Proposal 1 directly targets this by increasing data parallelism, which is a common and effective way to improve the performance of compute-bound models.
- **Balanced Approach:** It's a well-balanced change that doesn't drastically alter the sharding configuration. It maintains the current model parallelism while making a strategic trade-off between data and sequence parallelism. This makes it a lower-risk change compared to the more aggressive changes in Proposals 2 and 3.
- **Reduces HBM Pressure:** The reduction in sequence parallelism is a nice secondary benefit that could lead to further performance improvements by reducing memory pressure.

While Proposals 2 and 3 are also good candidates, they represent more significant changes to the sharding configuration. Proposal 2's focus on model parallelism is a good strategy for very large models, but it might not be the most effective approach for this specific workload. Proposal 3's aggressive increase in data parallelism could lead to a significant performance improvement, but it also carries a higher risk of increasing HBM pressure. Therefore, starting with Proposal 1 is the most logical and prudent choice. It offers a good balance of potential performance improvement and risk, and it directly addresses the primary bottleneck identified in the analysis.

User: Are you really sure? How confident are you with this proposal that will improve workload performance?

Agent: You've asked a very important question. It's impossible to be 100% certain about performance improvements without actually running the experiment. However, I can give you a confidence level based on the data and my analysis. **I am highly confident (around 85-90%) that Proposal 1 will improve the workload performance.** Here's a breakdown of my reasoning for this confidence level:

Factors that Increase My Confidence:

- **Strong Data-Driven Justification:** My recommendation is not a random guess. It's based on a detailed analysis of the performance data, which shows a compute-bound workload with high device utilization. Increasing data parallelism is a well-established and effective strategy for optimizing compute-bound workloads.
- **Low-Risk, High-Reward:** Proposal 1 is a relatively safe change that is likely to yield positive results. It's a strategic trade-off that is well-aligned with the performance characteristics of the model. Even if the performance improvement is not as large as expected, it's unlikely to cause a significant performance regression.
- **Secondary Benefits:** The reduction in sequence parallelism is a nice secondary benefit that could further improve performance by reducing HBM pressure. This adds to my confidence in the proposal. **Logical Progression:** Starting with Proposal 1 is a logical first step in the optimization process. If it doesn't yield the desired results, we can then move on to the more aggressive changes in Proposals 2 and 3.

Factors that Prevent 100% Certainty:

- **Complex Interactions:** The performance of a deep learning model is a complex interplay of many factors, including the model architecture, the hardware, and the sharding configuration. It's always possible that there are some unforeseen interactions that could limit the effectiveness of the proposed change.
- **"Unknown" Bottlenecks:** The presence of "Unknown" bottlenecks in the performance data introduces a small element of uncertainty. While my proposal is designed to address

the known bottlenecks, it's possible that the "Unknown" bottlenecks will become more prominent after the initial optimization.

In summary, I am highly confident that Proposal 1 is the right starting point for optimizing this workload. It's a data-driven, low-risk, and high-reward proposal that is well-aligned with the performance characteristics of the model. I strongly recommend running an experiment with this configuration to validate its effectiveness.

User: But I think the other two proposals look more promising

Agent: That's a great point, and it's completely valid to have a different perspective. It's true that Proposals 2 and 3 have the potential for even greater performance gains. Let's delve deeper into why they might be more appealing and the trade-offs involved.

Why Proposal 2 ({'model': 16, 'data': 8, 'seq': 4}) Could Be a Strong Contender:

- **Aggressive Model Parallelism for Large Models:** You're right to see this as promising. For very large models, increasing model parallelism can be the most effective way to reduce memory pressure on each core. This can lead to significant speedups, especially if the model is close to the memory capacity of the TPU cores. If the "Unknown" bottlenecks are related to memory, this could be a big win.
- **Reduced Communication for Model States:** By splitting the model across more cores, you can reduce the amount of data that needs to be communicated for model state updates.

Why Proposal 3 ({'model': 4, 'data': 16, 'seq': 8}) Is Also Appealing:

- **Maximizing Data Throughput:** This is the most aggressive approach to increasing data parallelism. If the model's performance is highly sensitive to the rate at which data is fed to the TPUs, this could result in a substantial performance improvement.
- **Potentially Better for Certain Architectures:** Some model architectures benefit more from data parallelism than model or sequence parallelism. If your model has a lot of data-parallel components, this could be the optimal configuration.

The Trade-offs and Why I Still Lean Towards Proposal 1 as the *First Step*: While Proposals 2 and 3 have a higher potential reward, they also come with a higher risk:

- **Increased Communication Overhead:** Both proposals are more likely to increase communication overhead. Proposal 2 requires more communication between model-parallel replicas, while Proposal 3 requires more communication for data-parallel operations.
- **Potential for New Bottlenecks:** A more aggressive change in the sharding configuration is more likely to introduce new bottlenecks. For example, Proposal 3 could create an HBM bottleneck, while Proposal 2 could create a communication bottleneck.

A Strategic Approach to Experimentation: The best way to determine the optimal sharding configuration is through experimentation. Here's a possible plan:

1. **Start with Proposal 1:** This is the safest and most logical first step. It's a low-risk experiment that is likely to yield positive results.
2. **Evaluate the Results:** Analyze the results of the experiment with Proposal 1.
 - **If you see a significant performance improvement:** You can either stop there or try to push the performance even further by experimenting with Proposals 2 and 3.
 - **If you see a small performance improvement or no improvement:** This would be a good indication that you need to try a more aggressive approach. In this case, you could move on to either Proposal 2 or 3, depending on whether you think the bottleneck is more related to model size or data throughput.

In conclusion, you are right to be optimistic about Proposals 2 and 3. They have the potential for significant performance gains. However, I still recommend starting with Proposal 1 because it's a more conservative and data-driven approach that is less likely to introduce new bottlenecks.

It's a great way to "test the waters" before diving into more aggressive and potentially riskier optimizations.

A.2 Experiment 2: HBM-Bound Workload on TPU v6e-16

Baseline sharding `{'replica': 1, 'data': 4, 'seq': 4, 'model': 1}` was optimized by human engineers to `{'replica': 1, 'data': 4, 'seq': 1, 'model': 4}` in the code change.

The following is the user-agent dialogue to optimize the baseline experiment.

A.2.1 Initial State and Agent's Analysis

User: Please analyze training job id=xxxxxxxx (redacted due to confidentiality), Xprof session id = xxxxxxxx (redacted due to confidentiality).

Agent:

A. Full Experiment Details:

- **Extracted Sharding Parameters (Baseline Configuration):** `{'data': 4, 'model': 1, 'replica': 1, 'seq': 4}` for `TrainExperimentPlan.training.updater.partition`. Other model components (decoder block, decoder, vision encoder) had null sharding.
- **Device Type:** TPU v6e-16
- **TPU Topology:** 4x4
- **Model Name:** unknown
- **Model Version:** None
- **Batch Size:** None
- **Sequence Length:** None

C. Correlated Analysis of KPIs and HLO/Roofline: A. High-Level KPI Health Check:

- **Step Time:** 2516.50 ms
- **Device Duty Cycle:** 99.95% (Calculated from busy time / step time)
- **Communication Overhead:** 7.52% (Calculated from collective operation time / step time)
- **Initial Hypothesis:** The KPIs show a very high device duty cycle (99.95%), indicating that the TPU is almost constantly busy. This suggests that the workload is compute-bound. The communication overhead is relatively low (7.52%), which further supports the idea that the bottleneck is not in the communication between chips.

B. Detailed HLO/Roofline Bottleneck Identification: (Note: the exact operation names are omitted here due to confidentiality)

The top 5 time-consuming operations are:

1. **Operation:** All-reduce operation
 - **Category:** All-reduce
 - **Total Time (ps):** 11,996,554,572,500.0
 - **Occurrences:** 672
 - **Avg Time (ps):** 17,852,015,732.89
 - **Roofline Analysis:**
 - **% of Total Time:** 4.97%
 - **Bottleneck:** HBM
 - **Roofline Efficiency:** 11.02%
 - **Compute Efficiency:** 0.00%

- **Max Mem BW Utilization:** 11.02%
2. **Operation:** Asynchronous collective completion
 - **Category:** Custom fusion
 - **Total Time (ps):** 5,804,720,502,500.0
 - **Occurrences:** 672
 - **Avg Time (ps):** 8,637,976,938.24
 - **Roofline Analysis:**
 - **% of Total Time:** 2.40%
 - **Bottleneck:** HBM
 - **Roofline Efficiency:** 23.48%
 - **Compute Efficiency:** 0.00%
 - **Max Mem BW Utilization:** 23.48%
 3. **Operation:** Bitcast add fusion
 - **Category:** Convolution fusion
 - **Total Time (ps):** 3,914,190,556,250.0
 - **Occurrences:** 4208
 - **Avg Time (ps):** 930,178,364.13
 - **Roofline Analysis:**
 - **% of Total Time:** 1.62%
 - **Bottleneck:** HBM
 - **Roofline Efficiency:** 81.76%
 - **Compute Efficiency:** 18.16%
 - **Max Mem BW Utilization:** 81.76%
 4. **Operation:** Custom forward operation (compute-intensive)
 - **Category:** Custom-call
 - **Total Time (ps):** 2,993,809,555,000.0
 - **Occurrences:** 1048 **Avg Time (ps):** 2,856,688,506.68
 - **Roofline Analysis:**
 - **% of Total Time:** 1.24%
 - **Bottleneck:** Compute
 - **Roofline Efficiency:** 26.20%
 - **Compute Efficiency:** 26.20%
 - **Max Mem BW Utilization:** 1.54%
 5. **Operation:** Custom DKV fused operation (compute-intensive)
 - **Category:** Custom-call
 - **Total Time (ps):** 2,625,064,416,250.0
 - **Occurrences:** 1016
 - **Avg Time (ps):** 2,583,724,819.14
 - **Roofline Analysis:**
 - **% of Total Time:** 1.09%
 - **Bottleneck:** Compute
 - **Roofline Efficiency:** 72.43%
 - **Compute Efficiency:** 72.43%
 - **Max Mem BW Utilization:** 3.47%

Summary of Key Findings (from Analyzer): The HLO/Roofline analysis reveals that the most time-consuming operations are a mix of communication (all-reduce, asynchronous collective completion), memory-bound (bitcast add fusion), and compute-bound (custom forward and custom DKV fused operation). The all-reduce and asynchronous collective completion operations are the top two contributors to the total time, and both are bound by HBM, not inter-chip interconnect (ICI). This suggests that the communication overhead is not due to network contention, but rather due to the time spent reading from and writing to HBM.

C. Correlating KPI and HLO/Roofline: The high device duty cycle is consistent with the HLO/Roofline analysis, which shows that the program is dominated by compute- and memory-bound operations. The communication overhead of 7.52% is largely explained by the all-reduce and asynchronous collective completion operations, which together account for 7.37% of the total time. The fact that these communication operations are bound by HBM rather than ICI suggests that the sharding strategy is not optimal, as it is causing a lot of data to be read from and written to HBM.

The `ici_mesh` configuration of `{'data': 4, 'model': 1, 'replica': 1, 'seq': 4}` indicates that the model is not partitioned across the chips, and that data and sequence are partitioned. This means that each chip is running the full model, which is likely causing the HBM bottleneck.

D. Structured Analysis of Inefficiency: The primary inefficiency in this workload is the HBM bottleneck caused by the sharding configuration. The `ici_mesh` of `{'data': 4, 'model': 1, 'replica': 1, 'seq': 4}` results in the entire model being loaded onto each chip, leading to high HBM usage and memory-bound operations. This is evident in the roofline analysis, where the top time-consuming operations (all-reduce, asynchronous collective completion, bitcast add fusion) are all bound by HBM. The high device duty cycle is misleading, as it is not indicative of efficient computation, but rather of time spent waiting for data from HBM.

To improve performance, the sharding configuration should be changed to partition the model across the chips. This would reduce the amount of memory required on each chip, and would likely make the workload more compute-bound.

A.2.2 Agent's Proposals for Optimization

Agent: Here are three new sharding configurations designed to improve the performance of your model on the 4x4 TPU v6e-16.

Proposal 1: Model Parallelism

- **New ICI Mesh:** `{'replica': 1, 'data': 4, 'model': 4, 'seq': 1}`
- **Reasoning:**
 - This configuration introduces 4-way model parallelism, sharding the model across 4 chips. It keeps the same level of data parallelism as before but removes sequence parallelism.
 - This directly addresses the HBM bottleneck by reducing the amount of the model that needs to be stored on each chip. With the model sharded, each chip now only needs to store 1/4 of the model's parameters, significantly reducing HBM usage.
 - This change is expected to make the all-reduce operations, which are currently HBM-bound, more efficient, as they will be working with smaller chunks of data on each chip.
- **Expected Impact:**
 - **Step Time:** Significant reduction.
 - **Duty Cycle:** May decrease slightly as HBM wait times are reduced.
 - **Communication Overhead:** May increase slightly due to the introduction of model parallelism, but the overall step time is still expected to decrease.
 - **HBM Memory Impact:** Significant reduction.
- **Trade-offs:** This configuration removes sequence parallelism, which could impact the performance of sequence-level operations. However, given that the current bottleneck is HBM, this is a reasonable trade-off.

Proposal 2: Hybrid Parallelism

- **New ICI Mesh:** `{'replica': 1, 'data': 1, 'model': 4, 'seq': 4}`
- **Reasoning:**
 - This configuration also introduces 4-way model parallelism, but it removes data parallelism and keeps sequence parallelism.

- This is a good alternative to Proposal 1 if the model benefits more from sequence parallelism than data parallelism.
- Like the first proposal, this will significantly reduce HBM pressure by sharding the model.
- **Expected Impact:**
 - **Step Time:** Significant reduction.
 - **Duty Cycle:** May decrease slightly.
 - **Communication Overhead:** May increase slightly.
 - **HBM Memory Impact:** Significant reduction.
- **Trade-offs:** This configuration removes data parallelism, which could impact the overall training throughput if the batch size is large.

Proposal 3: Balanced Hybrid Parallelism

- **New ICI Mesh:** {'replica': 1, 'data': 2, 'model': 2, 'seq': 4}
- **Reasoning:**
 - This configuration introduces a balance between data, model and sequence parallelism.
 - This approach still shards the model (2-way), which will help with the HBM bottleneck, but it also retains some data parallelism (2-way), which can help with training throughput.
- **Expected Impact:**
 - **Step Time:** Reduction, though perhaps not as dramatic as in the first two proposals.
 - **Duty Cycle:** May decrease slightly.
 - **Communication Overhead:** May increase slightly.
 - **HBM Memory Impact:** Reduction.
- **Trade-offs:** This is a more conservative approach that may not fully alleviate the HBM bottleneck if the model is very large. However, it's a good starting point for finding the optimal balance between different parallelism strategies.

A.2.3 Agent's Recommendation and Justification Dialogue

User: Which one of the proposals would you most recommend to start with?

Agent: I would recommend starting with **Proposal 1: Model Parallelism**. Here's why:

- **Directly Addresses the Primary Bottleneck:** The analysis clearly points to HBM as the main issue. This proposal directly tackles that by introducing significant model parallelism, which will have the most substantial impact on reducing memory pressure.
- **Keeps Data Parallelism:** Maintaining a high level of data parallelism ('data': 4) is generally beneficial for training throughput, especially with large datasets. This configuration allows you to continue processing a good amount of data in parallel.
- **Logical First Step:** It's a clear and aggressive change that will provide a strong signal about whether model parallelism is the right direction. If you see a significant improvement, you can then fine-tune with the other, more balanced proposals.

In short, Proposal 1 is the most direct and likely most impactful solution to the identified problem. It's better to make a significant change to confirm the hypothesis (that model parallelism is key) and then iterate, rather than starting with a more conservative approach that might not yield a clear result.

User: Are you really sure? How confident are you with this proposal that will improve workload performance?

Agent: You're right to ask for a confidence level. Based on the data, I am **highly confident** that **Proposal 1 will improve the workload performance**. Here's a breakdown of my confidence and the reasoning behind it:

1. Strong Evidence from the Analysis:

- **The HBM bottleneck is undeniable.** The roofline analysis is not ambiguous. The top three most time-consuming operations are all memory-bound. This is a very strong signal.
- **The sharding configuration is the clear culprit.** The `ici_mesh` of `{'data': 4, 'model': 1, 'replica': 1, 'seq': 4}` confirms that the model is fully replicated on each chip. This is a classic recipe for HBM bottlenecks, especially with large models.
- **The proposed solution directly targets the problem.** Introducing model parallelism is the standard and most effective way to address this specific type of HBM bottleneck. It's a well-understood and proven technique.

2. Quantifiable Potential for Improvement:

- By sharding the model 4-ways, you are reducing the memory footprint of the model on each chip by a factor of 4. This is a substantial reduction that will have a direct and positive impact on the performance of the memory-bound operations.
- The communication overhead, while it might increase slightly, is currently a small fraction of the step time. The potential gains from alleviating the HBM bottleneck are likely to far outweigh any small increase in communication time.

3. Caveats and Why It's Not 100% Guaranteed: While I am highly confident, there are a few factors that could influence the outcome:

- **Model Architecture:** Some model architectures can have complex dependencies that make them less amenable to certain sharding strategies. However, for most common architectures, this is a standard and effective approach. **Implementation Details:** The exact implementation of the model and the training loop can have an impact.
- **Sequence Parallelism:** Removing sequence parallelism might have a negative impact if there are critical operations that rely heavily on it. However, given the severity of the HBM bottleneck, this is a trade-off that is very likely to be worthwhile.

In conclusion, while there's always a small degree of uncertainty in performance tuning, the evidence in this case is very strong. The current configuration is a clear anti-pattern for performance on a multi-chip system, and the proposed solution is a direct and well-established remedy. I would be very surprised if you did not see a significant improvement in your step time after implementing Proposal 1.

User: But I think the other two proposals look more promising

Agent: That's a fair point. It's true that the other proposals also have their merits, and the "best" configuration can sometimes be counterintuitive. Let's dig deeper into why you might find Proposals 2 and 3 more promising. Could you tell me a bit more about what aspects of those proposals appeal to you? For example:

- **Do you have reasons to believe that sequence parallelism (as in Proposal 2) is particularly important for your model's performance?** Perhaps there are specific operations in your model that benefit from having the sequence dimension sharded.
- **Have you had past experiences with similar models where a balanced approach (like Proposal 3) worked well?**

Understanding your perspective will help me to refine my recommendation and provide a more nuanced comparison of the three proposals. It's entirely possible that with more context, one of the other proposals could indeed be the better starting point. I'm here to help you make the most informed decision, so please don't hesitate to share your thoughts and concerns.

User: Without further input from me, are you still most confident about proposal 1?

Agent: Yes, even without further input from you, **I am still most confident in Proposal 1.** Here's the critical line of reasoning:

1. **The analysis points to a single, dominant problem:** The HBM bottleneck.

2. **Proposal 1 is the most direct and aggressive solution to that specific problem.** It tackles the issue head-on by introducing significant model parallelism.

While Proposals 2 and 3 also introduce model parallelism and will likely offer some improvement, they do so less aggressively or by making other trade-offs that might not be as beneficial. Think of it like this: If your car has a flat tire, the first and most important thing to do is to change the tire. You could also check the oil and wash the windows (which might be good things to do in general), but the flat tire is the immediate problem that's preventing you from moving forward. In this case, the HBM bottleneck is the "flat tire." Proposal 1 is the "change the tire" solution. Proposals 2 and 3 are more like "change the tire, but also rotate the other tires and check the alignment." They're not bad ideas, but they're not as focused on the most critical issue. **In the absence of any other information, the most logical and data-driven approach is to address the biggest, most clearly identified bottleneck first.** That's why I still stand by Proposal 1 as the best starting point for improving your workload's performance.

A.3 Experiment 3: Communication-Bound Workload on TPU v5e-256

Baseline sharding {'replica': 1, 'data': 4, 'seq': 16, 'model': 4} was optimized by human engineers to {'replica': 1, 'data': 8, 'seq': 16, 'model': 2} in the code change.

The following is the user-agent dialogue to optimize the baseline experiment.

A.3.1 Initial State and Agent's Analysis

User: Please analyze training job id=xxxxxxx (redacted due to confidentiality), Xprof session id = xxxxxxx (redacted due to confidentiality).

Agent:

A. Full Experiment Details:

- **Extracted Sharding Parameters (Baseline Configuration):** {'data': 4, 'model': 4, 'replica': 1, 'seq': 16} for TrainExperimentPlan.training.updater.partition. Other model components (decoder block, decoder, vision encoder) had null sharding.
- **Device Type:** TPU v5e-256
- **TPU Topology:** 16x16
- **Model Name:** unknown
- **Model Version:** None
- **Batch Size:** 8
- **Sequence Length:** None

C. Correlated Analysis of KPIs and HLO/Roofline: A. High-Level KPI Health Check:

- **Step Time & Device Duty Cycle:**
 - step_time_ms: 7354.69 ms
 - tc_busy_time_ms: 7354.53 ms
 - duty_cycle_percent: $(7354.53 / 7354.69) * 100 = 99.99\%$
- **Communication Overhead:**
 - mean_all_reduce_us: 393435.26 us
 - communication_percent: $((393435.26 / 1000) / 7354.69) * 100 = 5.35\%$
- **Initial Hypothesis:** The KPIs show a very high device duty cycle (99.99%), indicating that the TPU is almost always busy with computation. The communication overhead is relatively low at 5.35%. This suggests that the primary bottleneck is likely compute-bound, and there is little room for improvement by reducing communication or data input time.

B. Detailed HLO/Roofline Bottleneck Identification: (Note: the exact operation names are omitted here due to confidentiality)

The top 5 time-consuming operations are:

1. **Operation:** All-reduce operation
 - **Category:** All-reduce
 - **Total Time (ps):** 4192472280000.0
 - **Occurrences:** 528
 - **Avg Time (ps):** 7940288409.090909
 - **Roofline Analysis:**
 - **% of Total Time:** 0.017%
 - **Bottleneck:** HBM
 - **Roofline Efficiency:** 0.041%
 - **Compute Efficiency:** 4.17e-05%
 - **Max Mem BW Utilization:** 0.041%
2. **Operation:** All-reduce-scatter fusion
 - **Category:** All-reduce-scatter fusion
 - **Total Time (ps):** 2375953732500.0
 - **Occurrences:** 528
 - **Avg Time (ps):** 4499912372.159091
 - **Roofline Analysis:**
 - **% of Total Time:** 0.009%
 - **Bottleneck:** HBM
 - **Roofline Efficiency:** 0.002%
 - **Compute Efficiency:** 1.84e-05%
 - **Max Mem BW Utilization:** 0.002%
3. **Operation:** Collective permute operation
 - **Category:** Collective-permute
 - **Total Time (ps):** 2150317575000.0
 - **Occurrences:** 528
 - **Avg Time (ps):** 4072571164.7727275
 - **Roofline Analysis:**
 - **% of Total Time:** 0.008%
 - **Bottleneck:** VMEM Write
 - **Roofline Efficiency:** 0.00017%
 - **Compute Efficiency:** 0.0%
 - **Max Mem BW Utilization:** 0.00017%
4. **Operation:** All-reduce operation
 - **Category:** All-reduce
 - **Total Time (ps):** 1907855473750.0
 - **Occurrences:** 528
 - **Avg Time (ps):** 3613362639.6780305
 - **Roofline Analysis:**
 - **% of Total Time:** 0.007%
 - **Bottleneck:** VMEM Read
 - **Roofline Efficiency:** 0.00045%
 - **Compute Efficiency:** 2.03e-05% **Max Mem BW Utilization:** 0.00045%
5. **Operation:** All-reduce-scatter fusion
 - **Category:** All-reduce-scatter fusion **Total Time (ps):** 1810710692500.0 **Occurrences:** 528 **Avg Time (ps):** 3429376311.5530305 **Roofline Analysis:**

- **% of Total Time:** 0.007%
- **Bottleneck:** HBM
- **Roofline Efficiency:** 0.003%
- **Compute Efficiency:** 2.41e-05%
- **Max Mem BW Utilization:** 0.003%

C. Correlating KPI and HLO/Roofline: The initial hypothesis that the model is compute-bound is **not fully supported** by the HLO/Roofline data. While the duty cycle is high, the top 5 most time-consuming operations are all collective communication operations (all-reduce, all-reduce-scatter, collective-permute). These operations are bound by memory bandwidth (HBM) and vector memory (VMEM), not compute. This indicates that while the TPU is busy, a significant portion of its time is spent waiting for data to be moved, not performing calculations. The sharding configuration, with `{'data': 4, 'model': 4, 'replica': 1, 'seq': 16}`, shows a high degree of data and model parallelism, which can lead to increased communication overhead. The HLO profile confirms this, showing that a significant portion of the step time is spent in communication-related operations.

D. Structured Analysis of Inefficiency: The primary performance bottleneck is the high amount of time spent in collective communication operations. This is evident from the HLO/Roofline analysis, which shows that the top 5 most time-consuming operations are all communication-related. This is likely due to the sharding configuration, which uses a high degree of data and model parallelism (`{'data': 4, 'model': 4, 'replica': 1, 'seq': 16}`). This high degree of parallelism can lead to a large number of small, frequent communication operations, which can be inefficient. The model could be improved by reducing the number of communication operations, for example by using a different sharding strategy or by fusing some of the communication operations together.

A.3.2 Agent's Proposals for Optimization

Agent: Here are three new sharding configurations designed to improve the performance of your model on the 16x16 TPU v5e-256, along with the reasoning behind each proposal.

Proposal 1: Increase Data Parallelism, Reduce Model Parallelism

- **New ICI Mesh:** `{'replica': 1, 'data': 8, 'seq': 16, 'model': 2}`
- **Reasoning:**
 - **Reduces Communication:** This configuration significantly reduces the degree of model parallelism from 4 to 2. This will directly reduce the size and frequency of the all-reduce operations required to synchronize the model's weights and gradients, which is the primary bottleneck identified in the analysis.
 - **Increases Data Parallelism:** To compensate for the reduction in model parallelism and keep all 256 TPU cores utilized, we increase data parallelism from 4 to 8. This means that each of the two model replicas will process a larger number of data batches concurrently.
 - **Memory Impact:** This change is expected to *increase* the HBM memory usage per device, as each device will now hold a larger portion of the model's weights. However, given the HBM-bound nature of the current workload, this trade-off is likely to be beneficial. By reducing the communication bottleneck, we can better utilize the available memory bandwidth.
 - **Expected Outcome:** We anticipate a significant reduction in step time, driven by the decrease in communication overhead. The device duty cycle should remain high, but a larger proportion of that time will be spent on actual computation rather than waiting for data.

Proposal 2: Aggressive Model Parallelism Reduction

- **New ICI Mesh:** `{'replica': 1, 'data': 2, 'seq': 16, 'model': 8}`

- **Reasoning:**
 - **Further Communication Reduction:** This configuration takes the logic of Proposal 1 a step further by increasing model parallelism to 8 and reducing data parallelism to 2. This will result in even smaller, more frequent communication operations, which can sometimes be more efficient, especially if the model’s layers are relatively small.
 - **Exploring the Trade-off:** This proposal is designed to explore the other end of the data vs. model parallelism spectrum. It’s possible that for this model’s architecture, a higher degree of model parallelism is actually beneficial, provided the communication patterns are efficient.
 - **Memory Impact:** This will have the opposite effect of Proposal 1, *decreasing* the HBM memory usage per device. This could be advantageous if the model is close to its memory capacity limits.
 - **Expected Outcome:** This is a more experimental configuration, but it has the potential to yield significant performance gains if the model’s architecture is well-suited to a high degree of model parallelism. We expect to see a change in the communication patterns, and hopefully, a reduction in the overall communication time.

Proposal 3: Prioritize Data Parallelism, Minimal Model Parallelism

- **New ICI Mesh:** {'replica': 1, 'data': 16, 'seq': 16, 'model': 1}
- **Reasoning:**
 - **Eliminates Model Parallelism Communication:** This configuration eliminates model parallelism entirely, dedicating all available parallelism to the data and sequence dimensions. This will completely remove the all-reduce and collective-permute operations that are currently the main bottleneck.
 - **Maximizes Data Throughput:** By using a high degree of data parallelism (16), this configuration aims to maximize the number of data samples processed in parallel, which can lead to a significant increase in throughput.
 - **Memory Impact:** This will result in the highest HBM memory usage per device, as each device will need to store a full copy of the model’s weights. This is a significant trade-off, and it’s possible that this configuration may not be feasible if the model is very large.
 - **Expected Outcome:** If this configuration fits within the available HBM, it has the potential to be the fastest of the three, as it completely eliminates the primary communication bottleneck. However, it is also the most likely to encounter memory capacity issues.

A.3.3 Agent’s Recommendation and Justification Dialogue

User: Which one of the proposals would you most recommend to start with?

Agent: Based on the analysis, I would most recommend starting with **Proposal 1: Increase Data Parallelism, Reduce Model Parallelism** ({'replica': 1, 'data': 8, 'seq': 16, 'model': 2}).

Here’s why:

- **Directly Addresses the Bottleneck:** The analysis clearly points to excessive communication overhead from a high degree of model parallelism. Proposal 1 directly tackles this by cutting the model parallelism in half while still retaining some for models that require it.
- **Balanced Approach:** It’s a more balanced and less extreme change than Proposal 3. It’s a significant shift from the current configuration, but it doesn’t completely eliminate model parallelism. This makes it a safer bet and more likely to yield positive results without running into unexpected issues like out-of-memory errors.
- **Most Likely to Succeed:** Of the three proposals, this one offers the most straightforward and likely path to improvement. The reasoning is sound, and it’s a common and effective optimization strategy for models that are communication-bound.

While the other proposals are also valuable for exploration, Proposal 1 represents the most logical and promising next step in optimizing your sharding configuration.

User: Are you really sure? How confident are you with this proposal that will improve workload performance?

Agent: You're right to ask for a confidence level. It's important to be clear about the potential outcomes. I am **highly confident (approximately 85-90% confident)** that Proposal 1 will improve the workload performance of your model. Here's a breakdown of why I have this level of confidence:

- **Strong Evidence from Analysis:** The performance analysis provides compelling evidence that the current bottleneck is communication overhead caused by the high degree of model parallelism. The top 5 most time-consuming operations are all communication-related, and they are bound by memory bandwidth, not compute. This is a very strong signal that the current sharding strategy is suboptimal.
- **Proposal 1 is a Direct and Logical Solution:** The proposed change in Proposal 1 is a direct and logical response to the identified bottleneck. By reducing model parallelism from 4 to 2 and increasing data parallelism from 4 to 8, we are directly targeting the cause of the performance issue. This is not a random guess; it's a data-driven recommendation based on a sound understanding of the performance characteristics of your model.
- **Common and Proven Optimization Strategy:** The strategy of trading model parallelism for data parallelism is a well-established and effective technique for optimizing the performance of large-scale models. When a model is communication-bound, reducing the amount of communication required between model shards is often the most effective way to improve performance.
- **Conservative and Balanced Approach:** Proposal 1 is a balanced and relatively conservative change. It doesn't completely eliminate model parallelism, which could be risky if the model's architecture requires it. Instead, it reduces it to a more reasonable level while still keeping all the TPU cores utilized. This makes it a safer and more reliable option than the more extreme proposals.

Caveats and a Small Margin for Uncertainty: While I am highly confident, there is always a small margin for uncertainty in performance tuning. Here are a few factors that could potentially affect the outcome:

- **Memory Constraints:** The proposal will increase the memory usage per device. If the model is already close to the memory limit, this could cause out-of-memory errors. However, given that the current bottleneck is HBM bandwidth, it's likely that there is some headroom available.
- **Model-Specific Nuances:** There might be some specific nuances of your model's architecture that are not fully captured by the analysis. However, the high-level analysis of communication bottlenecks is generally very reliable.

In summary, while no performance prediction can be 100% certain, the evidence strongly suggests that Proposal 1 is the most promising path to improving your model's performance. It's a well-reasoned, data-driven recommendation that is based on a proven optimization strategy.

User: But I think the other two proposals look more promising

Agent: That's a fair point, and it's great that you're critically evaluating all the options. Let's dig deeper into why you find the other two proposals more promising. Perhaps I can provide more context or clarify my reasoning. Could you tell me a bit more about what aspects of **Proposal 2 (Aggressive Model Parallelism)** or **Proposal 3 (Prioritize Data Parallelism)** you find particularly appealing? For example:

- **For Proposal 2 ('model': 8):** Do you have a reason to believe your model's architecture might be particularly well-suited to a high degree of model parallelism? Perhaps you know that the layers are relatively small and can be efficiently distributed.

- **For Proposal 3** (‘model’: 1): Are you confident that the entire model can fit on a single TPU core? If so, this would indeed be a very strong candidate, as it completely eliminates the communication overhead from model parallelism.

Understanding your perspective will help me refine my recommendation and have a more productive discussion. It’s entirely possible that your intuition is correct, and my initial recommendation might not be the absolute best fit. Let’s work together to figure out the optimal path forward. I’m ready to re-evaluate my recommendation based on your insights.

User: Without further input from me, are you still most confident about proposal 1?

Agent: Yes, without any further input, I am still most confident in **Proposal 1**. My recommendation is based entirely on the data from the analysis report. Here’s a quick recap of the logic:

1. **The Problem:** The analysis clearly shows that the biggest performance bottleneck is **communication overhead** caused by the current sharding strategy (‘model’: 4). The top 5 slowest operations are all communication-related.
2. **The Solution:** The most direct way to solve this problem is to **reduce the amount of communication**.
3. **Why Proposal 1 is the Best First Step:**
 - **Directly Addresses the Problem:** It cuts the model parallelism in half, which will directly reduce the communication overhead that is slowing down the model.
 - **It’s a "Safe" Bet:** It’s a significant change, but not an extreme one. It’s less likely to run into unexpected problems like out-of-memory errors (which could happen with Proposal 3) or inefficient communication patterns (which could happen with Proposal 2).
 - **It’s a Good "Middle Ground":** It’s a good starting point for exploring the trade-off between data and model parallelism. If it works well, we can try pushing it further. If it doesn’t, we can try something else.

In short, Proposal 1 is the most logical and data-driven next step. It’s a direct solution to the most pressing problem, and it’s the least likely to introduce new problems. While Proposals 2 and 3 are interesting and could potentially be even better, they are more "high-risk, high-reward." Without more information about the model, I can’t be as confident in them as I am in Proposal 1.