vLLM Office Hours #39

Special Topic: Intro to batch invariant in vLLM

January 8, 2026

Michael Goin
Principal Engineer, Red Hat
vLLM Committer

Saša Zelenović
Developer Marketing and Advocacy Manager, Red Hat

Wentao Ye
Machine Learning Engineer, Red Hat
vLLM Committer

Bram Wasti
Software Engineer, Meta

What's new in the past two weeks?

vLLM Project Update

- vLLM-Omni v0.12.0rc1
- vLLM Semantic Router v0.1
- vLLM v0.13.0

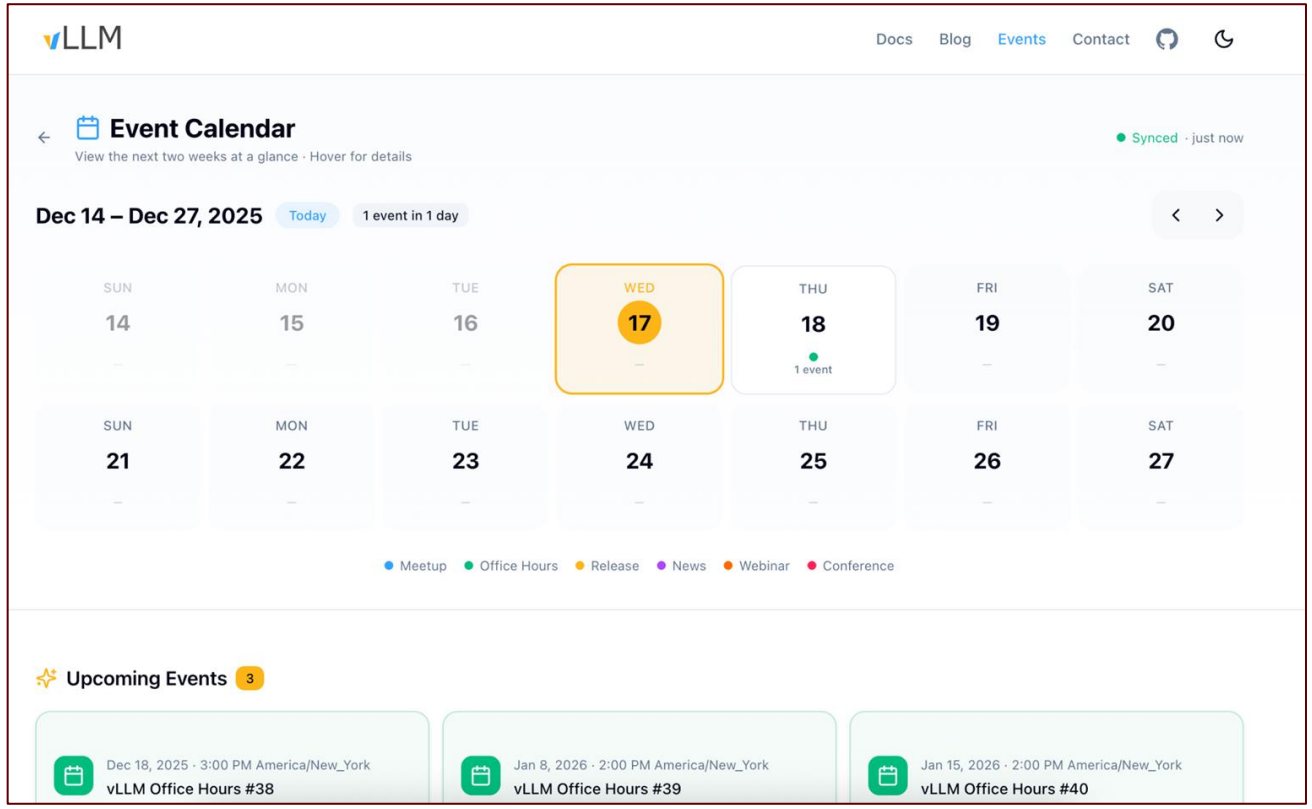
Upcoming vLLM Office Hours Sessions

- **[Dec 18]** vLLM 2025 Retrospective & 2026 Roadmap
- **[TODAY]** Intro to batch invariant in vLLM
- **[Jan 15]** Intro to Speculators, a unified library for building and storing speculative decoding algorithms for LLMs with vLLM
- **[Jan 22]** LLM Compressor update
- **[Jan 29]** Deep Dive into the vLLM CPU offloading connector

Register for all sessions here.

View previous recordings here.

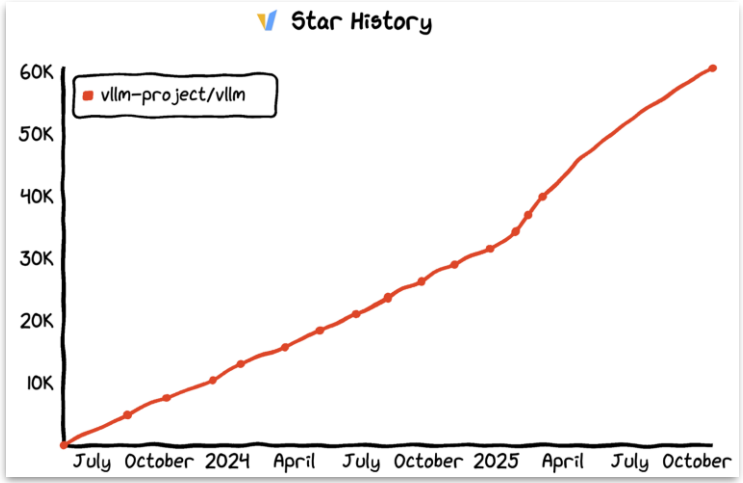
Reminder: New vLLM Website & **Events Calendar**



- Head over to vllm.ai/events to see upcoming office hours, meetups, conferences, etc.

What is vLLM?

The High-Throughput and Memory-Efficient inference and serving engine for LLMs



Most Popular LLM Serving Engine

- 65K+ GitHub stars, 800+ PRs/month
- 500K++ GPUs deployed 24/7
- 2K+ contributors, 10K+ members in slack.vllm.ai

>2000 Contributors from >50 major companies

https://github.com/vllm-project/vllm

```
$ uv pip install vllm --torch-backend=auto
$ vllm serve deepseek-ai/DeepSeek-V3.1 -tp 8
```

Broad Model Support (>100 arches)

Llama DeepSeek Qwen GPT-OSS Gemma Mistral Phi Granite Cohere

Flexible Device Parallelism

Tensor, Pipeline, Expert, Data, Context Parallel
Disagg Prefill/Decode, Disagg Encoder

Wide Hardware Support

CUDA ROCm Gaudi/XPU TPU Neuron Ascend CPU
Metal MACA RBLN Spyre MLU Kunlun

Diverse Project Ecosystem

llm-d vLLM-Omni GuideLLM vLLM Semantic Router
LLM Compressor PECULATORS Production Stack

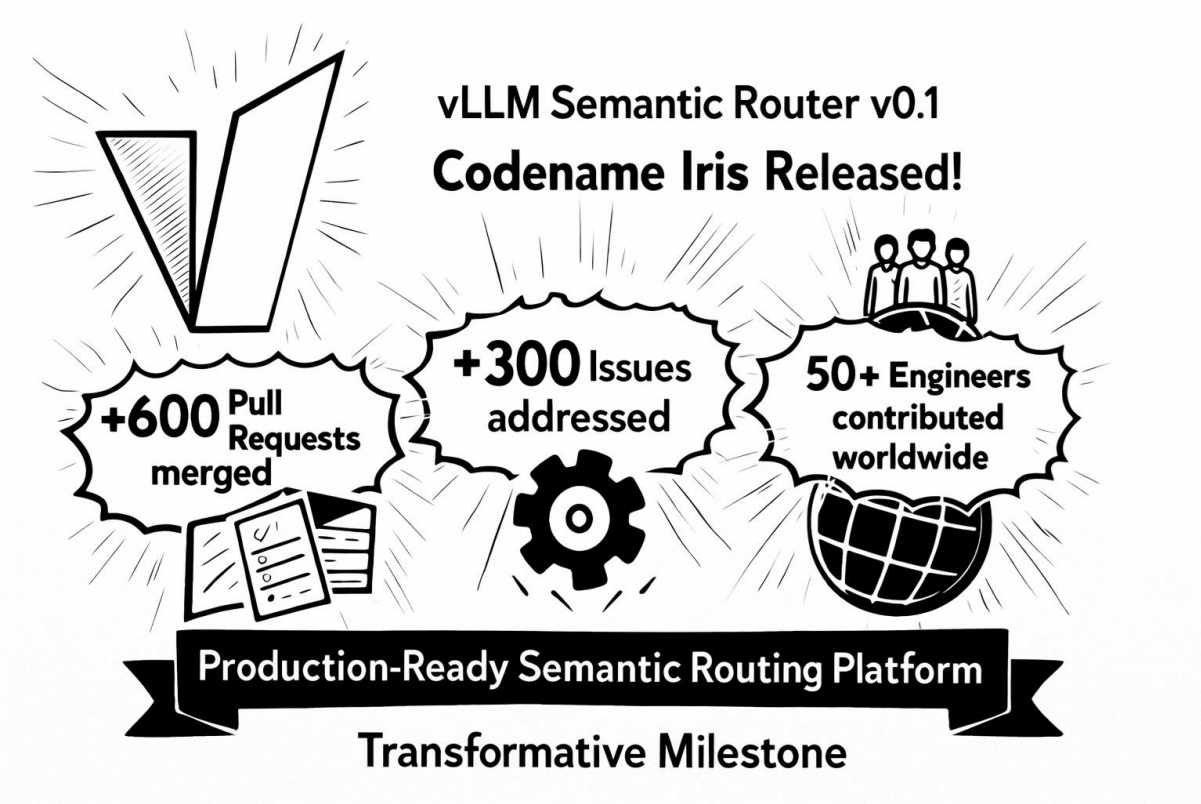
vLLM-Omni v0.12.0rc1 Pre-Release

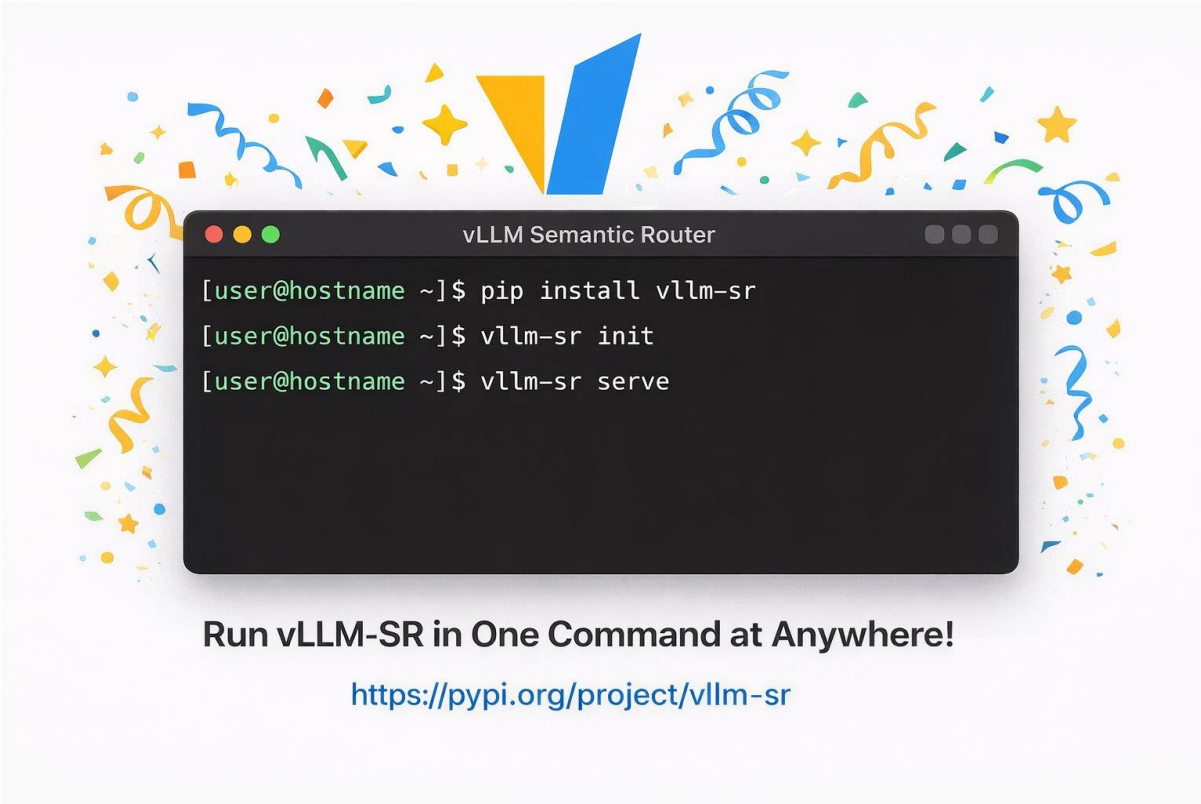
https://github.com/vllm-project/vllm-omni/releases/tag/v0.12.0rc1

https://docs.vllm.ai/projects/vllm-omni/en/latest/

vLLM Semantic Router v0.1 “Iris” Release

https://blog.vllm.ai/2026/01/05/vllm-sr-iris.html





Thanks to the many clean, from-scratch implementations of vLLM!

https://github.com/GeeeekExplorer/nano-vllm

https://github.com/Wenyueh/MinivLLM

https://github.com/skyzh/tiny-llm

What's new in vLLM v0.13.0

🎉 442 commits from 207 contributors, including 61 new contributors! 🎉

Model Support

- New models: BAGEL (AR only) (#28439), AudioFlamingo3 (#30539), JAIS 2 (#30188), Nemotron latent MoE (#30203).
- Tool parsers: DeepSeek-V3.2 (#29848), Gigachat 3 (#29905), Holo2 reasoning (#30048).
- Model enhancements: Qwen3-VL embeddings (#30037), Qwen3-VL EVS (Efficient Video Sampling) (#29752), DeepSeek V3.2 proper `drop_thinking` logic (#30490) and top-k fix (#27568).
- Task expansion: Automatic TokenClassification model conversion (#30666), Ultravox v0.7 transformer projector (#30089).
- Quantization: BitsAndBytes for Qwen3-Omni-MoE (#29896).
- Speculative decoding: Eagle/Eagle3 Transformers backend (#30340), Mamba `selective_state_update` spec decode (#29488).

Engine Core

- Compilation via `compile_ranges` for selective kernel compilation (#24252).
- Prefix caching: xxHash high-performance hash option (#29163).
- Attention: PrefixLM support for FlexAttention (#27938) and TritonAttention (#30386), CUDA graphs for 3D Triton attention (#28306).
- Batch invariance: FA2 and LoRA batch-invariant support (#30018), `TRITON_MLA` without prefix-caching (#29125).
- Pooling: Chunked prefill for ALL pooling tasks (#27145), multi-vector retrieval API (#26686).
- Model Runner V2: Min-p (#30171), NaN detection in logits (#30187).
- Speculative decoding: Medusa GPU-CPU sync avoidance (#29723), async spec-decode improvements (#29624).
- Whisper: ~3x speedup vs v0.12.0, Encoder batching (#29421), `FULL_DECODE_ONLY` CUDA graph (#30072), CPU support (#30062).
- Performance: Fused blockwise quant RMS norm (#27883), MoE LoRA loading reduction (#30243), encoder cache optimization (#30475), CPU KV offloading streams (#29013).

What's new in vLLM v0.13.0

🎉 442 commits from 207 contributors, including 61 new contributors! 🎉

Hardware & Performance

- NVIDIA Blackwell Ultra SM103 (GB300) support (#30484).
- Several DeepSeek/Kimi optimizations:
  - DeepEP High-Throughput CUDA graph enabled by default: 5.3% throughput, 4.4% TTFT improvement (#29558)
  - DeepGEMM fused layout: 10.7% TTFT improvement (#29546)
  - DeepGEMM experts init: 3.9% TTFT improvement (#30494)
  - `group_topk` kernel: 2% throughput improvement (#30159)
  - Sparse prefill kernel for DeepSeek-V3.2 FP8 KV-cache (#27532)
  - MLA FP8 Quant (#29795), broadcast k_nope/k_pe (#29710)
- CPU: Whisper support (#30062), Arm vectorized exp (#30068), x86 CPU wheel pipeline (#28848).
- AMD ROCm: Aiter quantization kernels (#25552), torch.compile layernorm/silu + FP8 quant (#25693), Triton ScaledMM fallback (#26668), MXFP4 w4a4 inference (#29775).
- Intel XPU: wNa16 compressed tensors (#29484).
- Build: CUDA 13 aarch64 wheels (#30341), Docker kernel build stage (#29452), Ascend NPU Docker (#30015).

Large Scale Serving & Disaggregated Prefill/Decode

- KV connectors: Mooncake Transfer Engine (#24718), cache reset via `/reset_prefix_cache` (#27170), KV events (#28309), failure recovery config (#26813).
- NIXL: Compatibility checking in handshake (#29503), large batch proxy support (#28782).
- EPLB: NVFP4 support (#29804), algorithm abstraction (#26471).
- Multi-node: External launcher mode (#29833).
- Hybrid allocator: Optional KV connector integration (#29805).
- Performance: silu_mul_per_token_group_quant_fp8 kernel for DP/EP (#29470).

What's new in vLLM v0.13.0

🎉 442 commits from 207 contributors, including 61 new contributors! 🎉

API & Frontend

- Responses API: MCP type infrastructure (#30054), Browser/Container MCP tools (#29989), full MCP Python loop (#29798), extra body parameters (#30532).
- Configuration: `AttentionConfig` replaces `VLLM_ATTENTION_BACKEND` env var (#26315).
- Chat templates: DeepSeek-V3.2 (#29837), DeepSeek-V3.2 developer tools (#30040).
- Anthropic API: Streaming fixes (#29971, #30266).
- Embeddings: Binary format with `encoding_format=bytes_only` (#30249), multiple image/audio per request (#29988), tokenization_kwargs override (#29794).
- Metrics: Prefill KV compute metric excluding cached tokens (#30189).
- Profiling: Layer-wise NVTX (#29990), profiling CLI config (#29912).
- UX: Better OOM errors (#28051), ModelConfig validation (#30213), distributed executor errors (#30140).

Breaking Changes & Deprecations

This release includes deprecation removals, PassConfig flag renames, and attention configuration changes from environment variables to CLI arguments. **Please review the breaking changes section carefully before upgrading.**

- PassConfig flags renamed per RFC #27995 (#29646)
- Attention env vars → CLI args: `VLLM_ATTENTION_BACKEND` replaced with `--attention-backend` (#26315)
- Removed `-O.xx` flag (#29991)
- Removed deprecated plugin/compilation fields (#30396)
- Removed deprecated task, seed, MM settings (#30397)
- Removed `embed_input_ids`/`embed_multimodal` fallbacks (#30458)
- Removed tokenizer setter (#30400)
- Deprecations: `merge_by_field_config` (#30035, #30170), `--convert reward` → `--convert embed` (#30463)

Thank you to the over 2000 contributors!

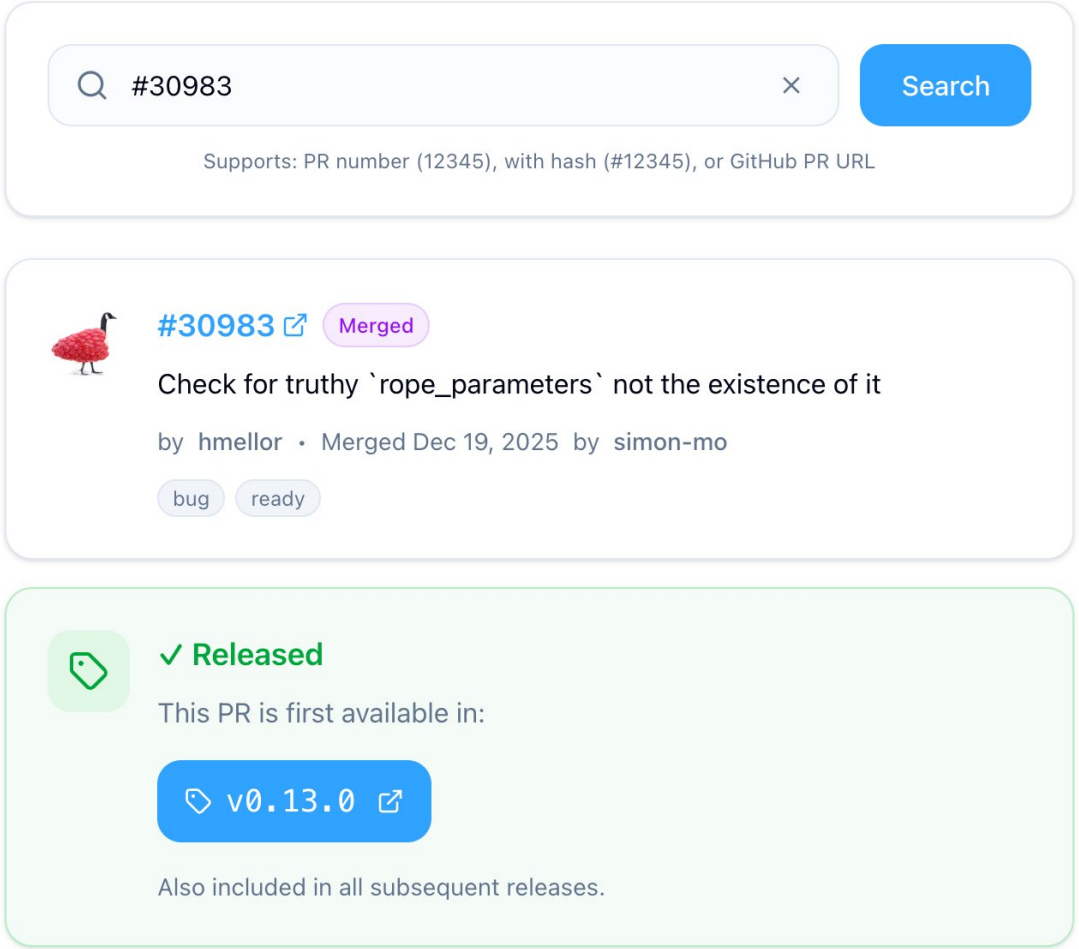
https://blog.vllm.ai/2025/12/15/vllm-epd.html

A thank you gift: PR Release Finder

Ever wondered "Which release first included my PR?"

Just enter your PR number or URL to track your code's journey into production.

vllm.ai/pr-lookup



Today's special topic:

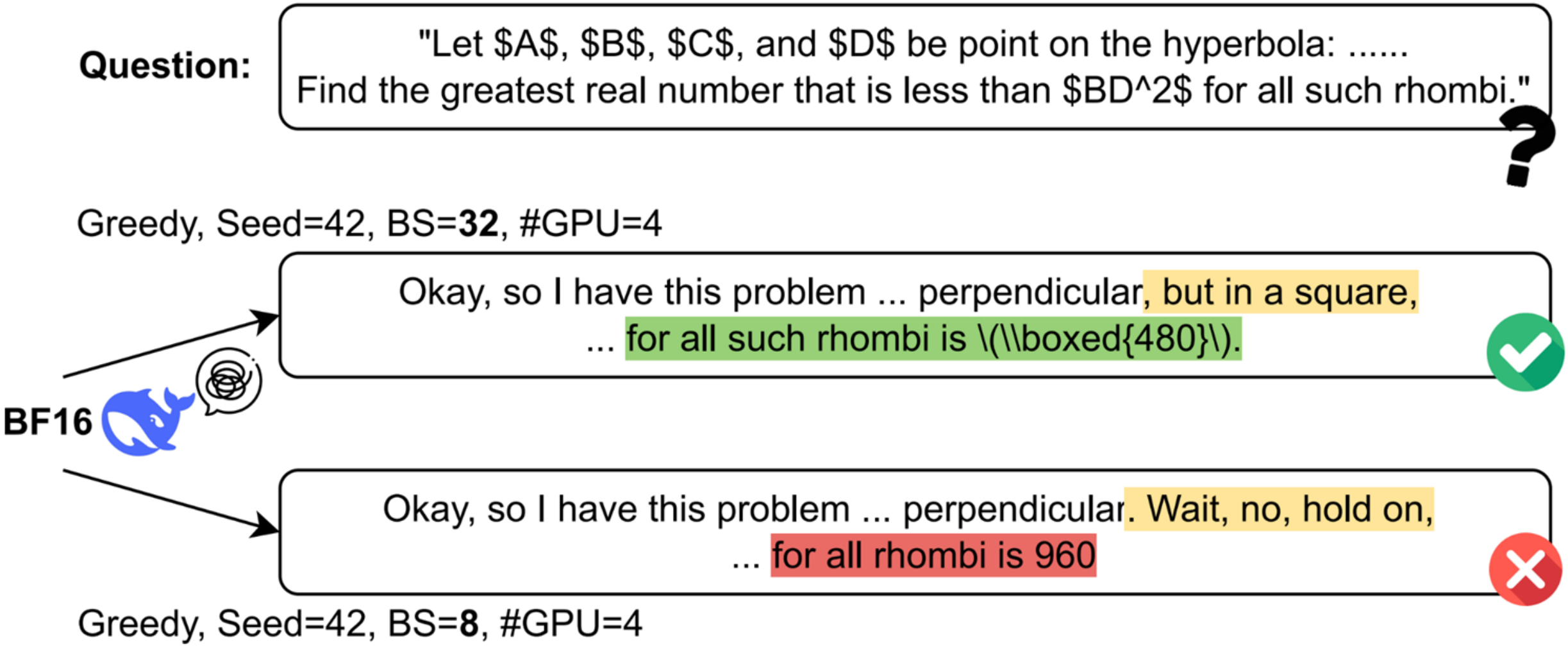
Intro to batch invariance in vLLM

Wentao Ye
Machine Learning Engineer, Red Hat
vLLM Committer

Bram Wasti
Software Engineer, Meta
vLLM contributor

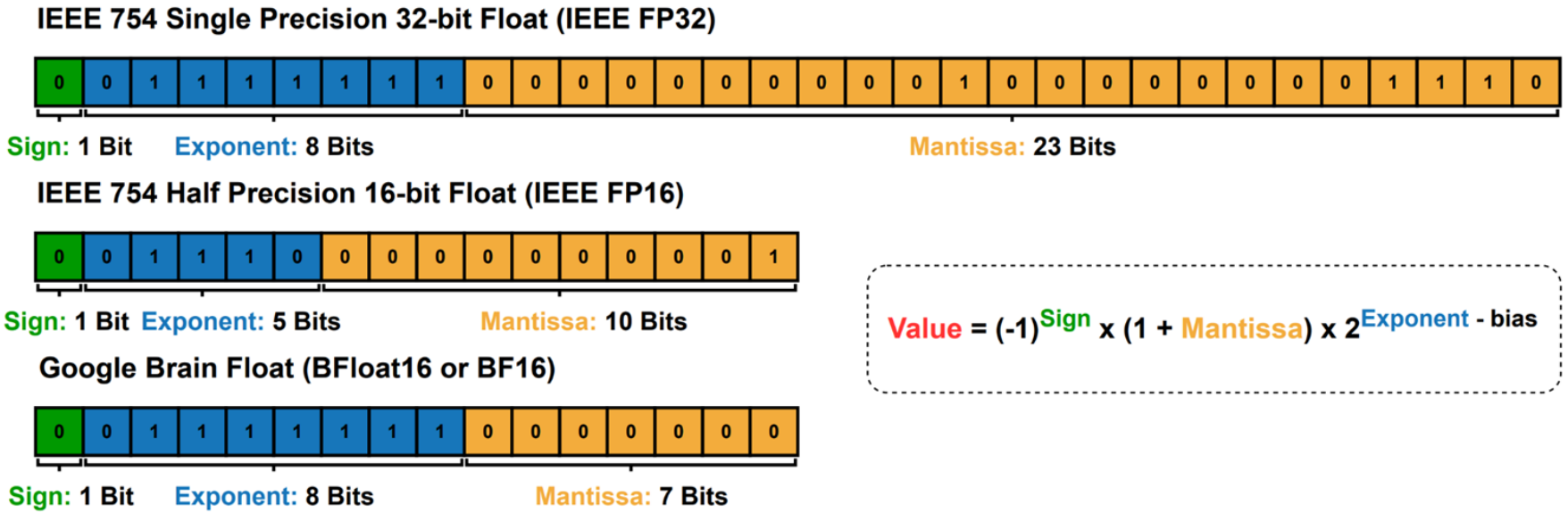
Red Hat

Why does this happen?



Even with a fixed seed and temperature=0?

Reason 1: Rounding errors of Floating Points

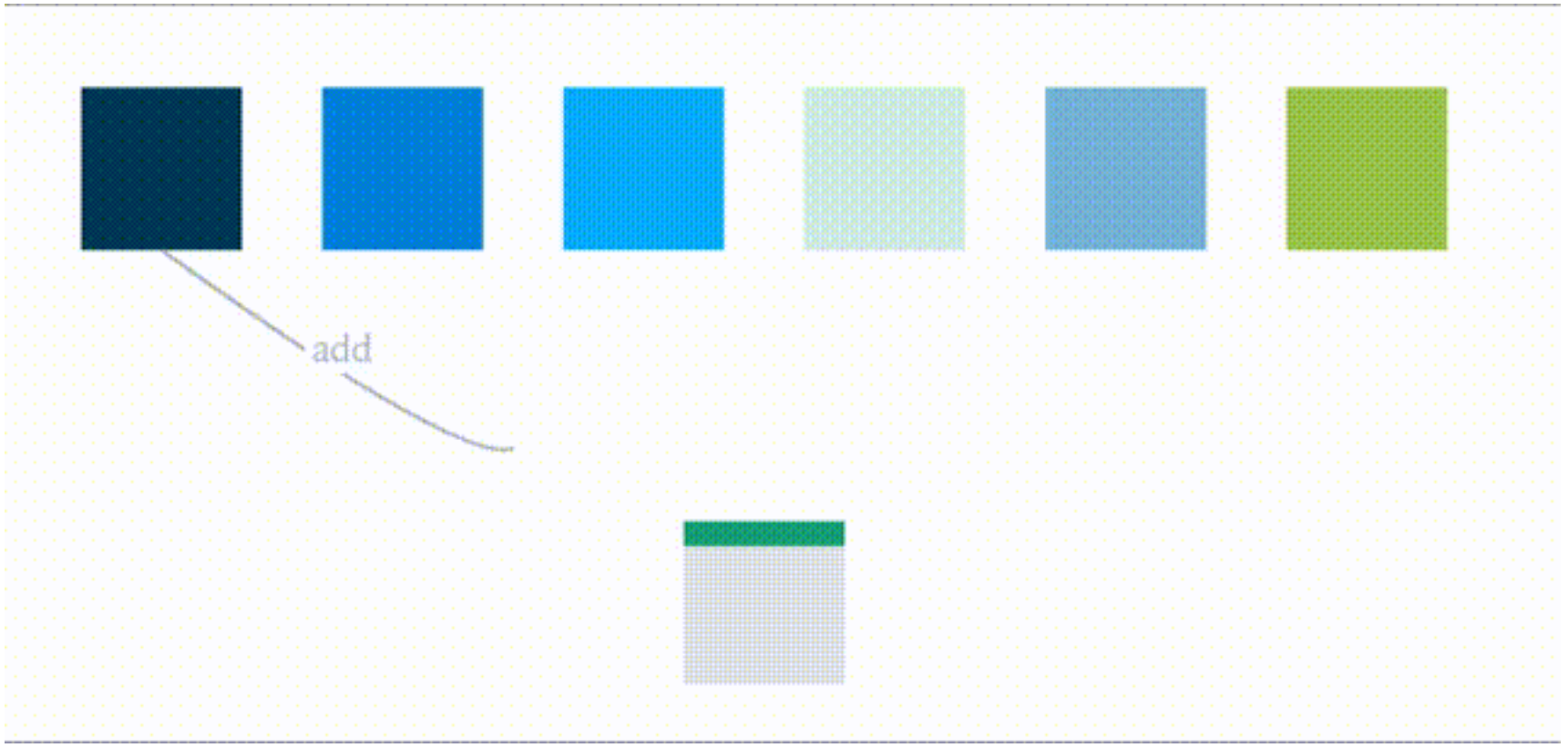


Precision	Decimal	Rounding Error
FP32	1.00012004375457761	$\approx +4.38\text{e}{-8}$
FP16	1.0	≈ -0.00012
BF16	1.0	≈ -0.00012

Reason 2: Non-Associativity

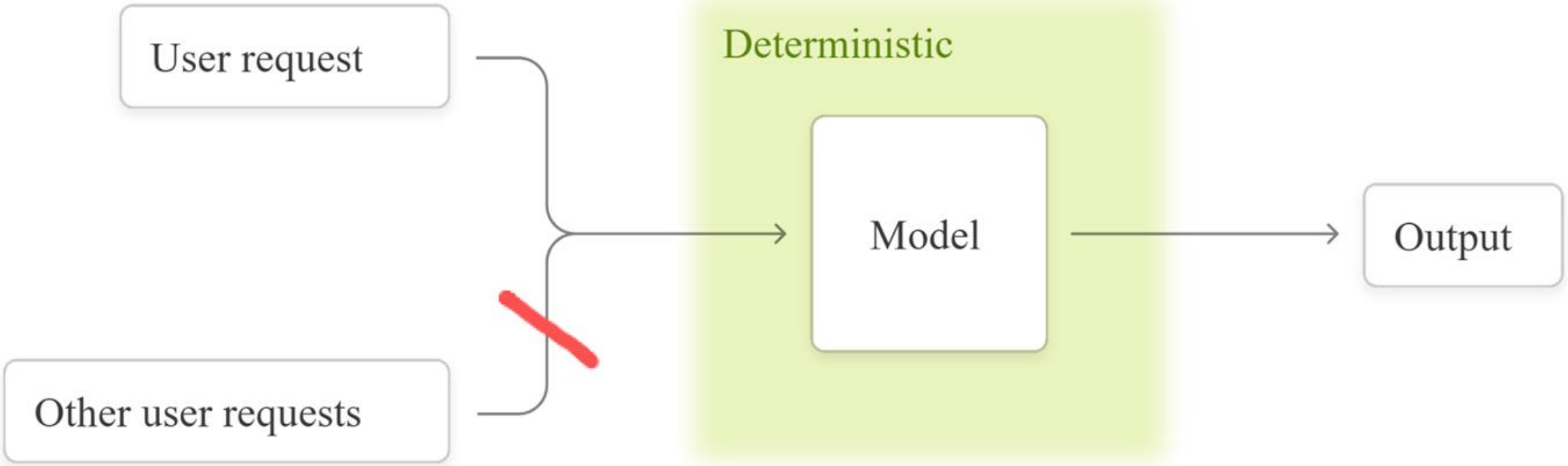
Example	Sum Order	FP32	BF16
$a, b, c = 0.1, -0.1, 0.2$	$a + b + c$	00111111001001100110011001100111**01**	0011111001001**101**
	$a + c + b$	00111111001001100110011001100111**10**	0011111001001**110**
$a, b, c = 0.0016, 0.0027, 1.0$	$a + b + c$	00111111100000001000110011100111	0011111110000001
	$a + c + b$	00111111100000001000110011100111	0011111110000000

Reason 1+2 -> Reduction order matters



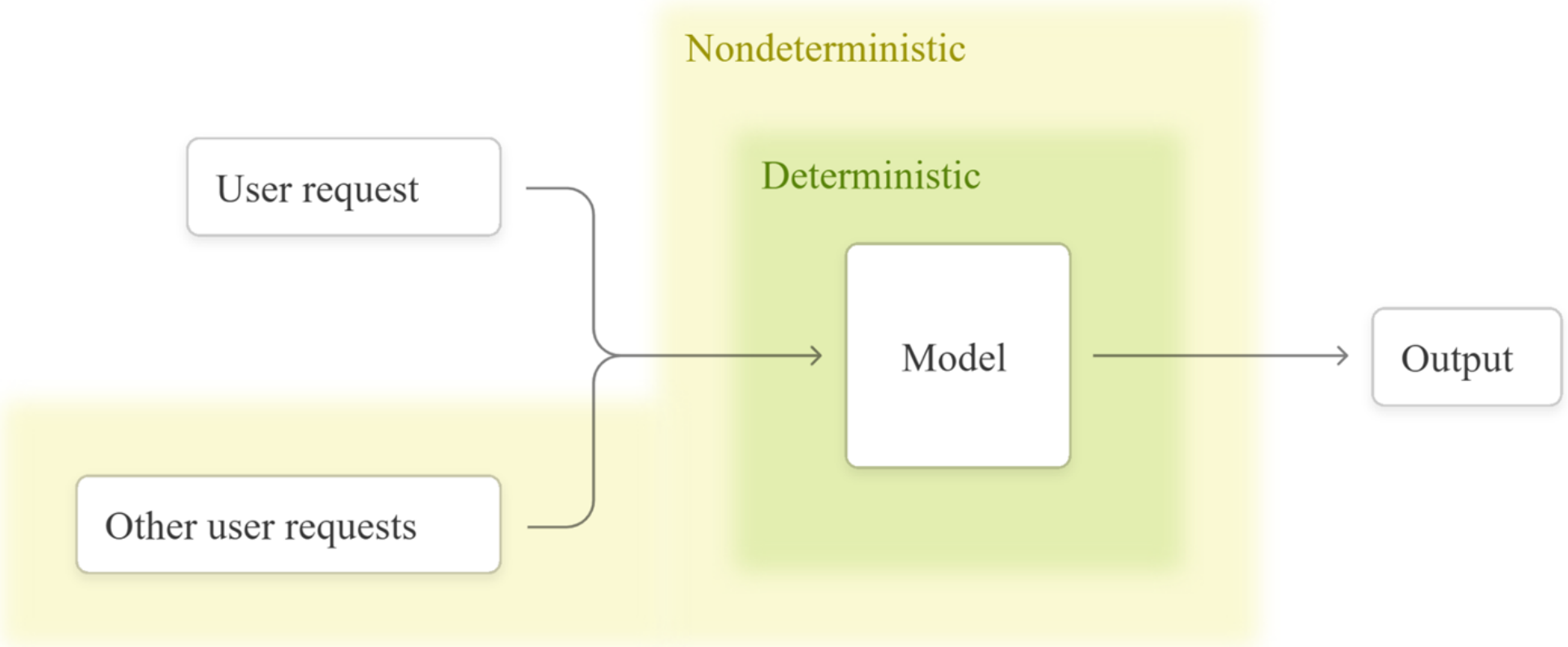
Sum with atomic add on GPU
Guaranteed for all elements, but the order might change

Reason 3: Continuous Batching



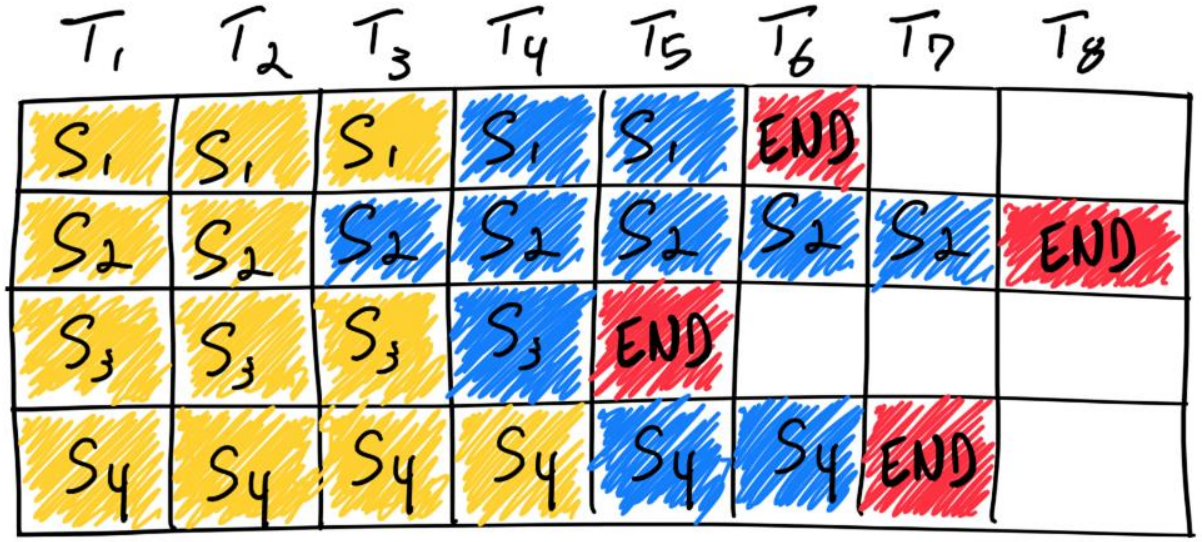
For LLM inference:
No atomic add operator during forward, good for single request

Reason 3: Continuous Batching



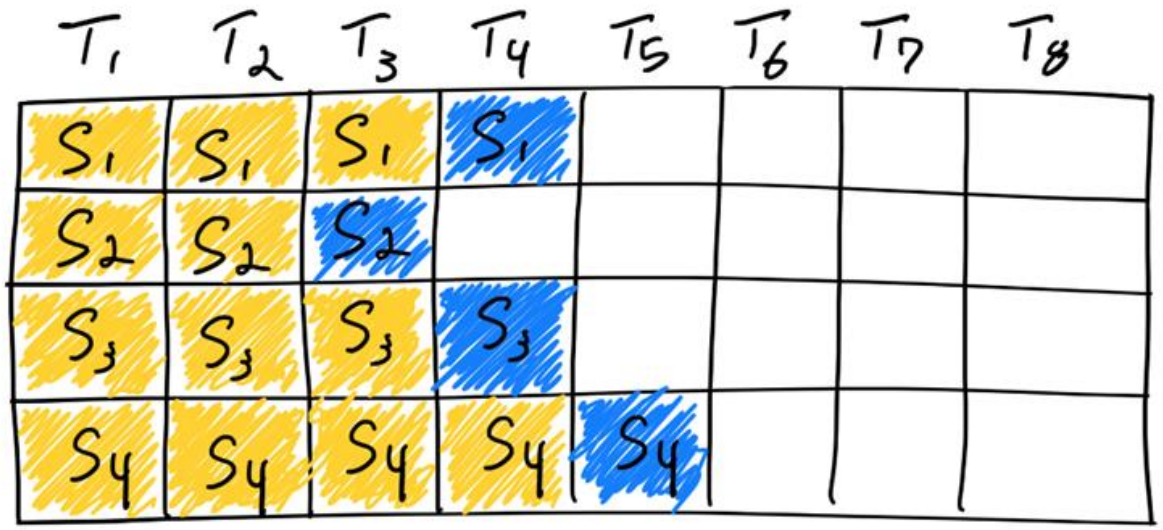
Batching multiple requests together matters

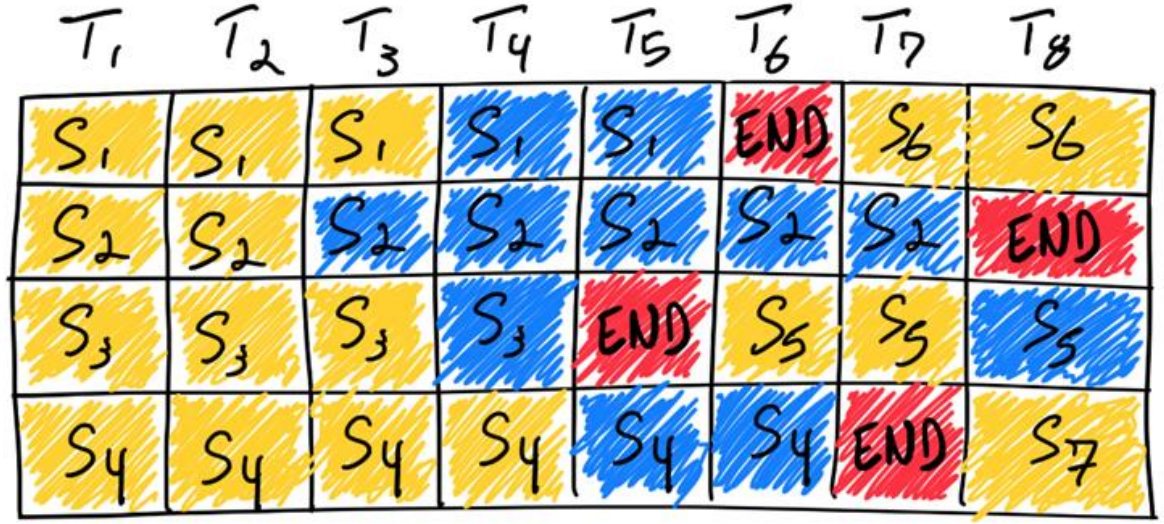
Reason 3: Continuous Batching



Naive batching, where would S5 be?

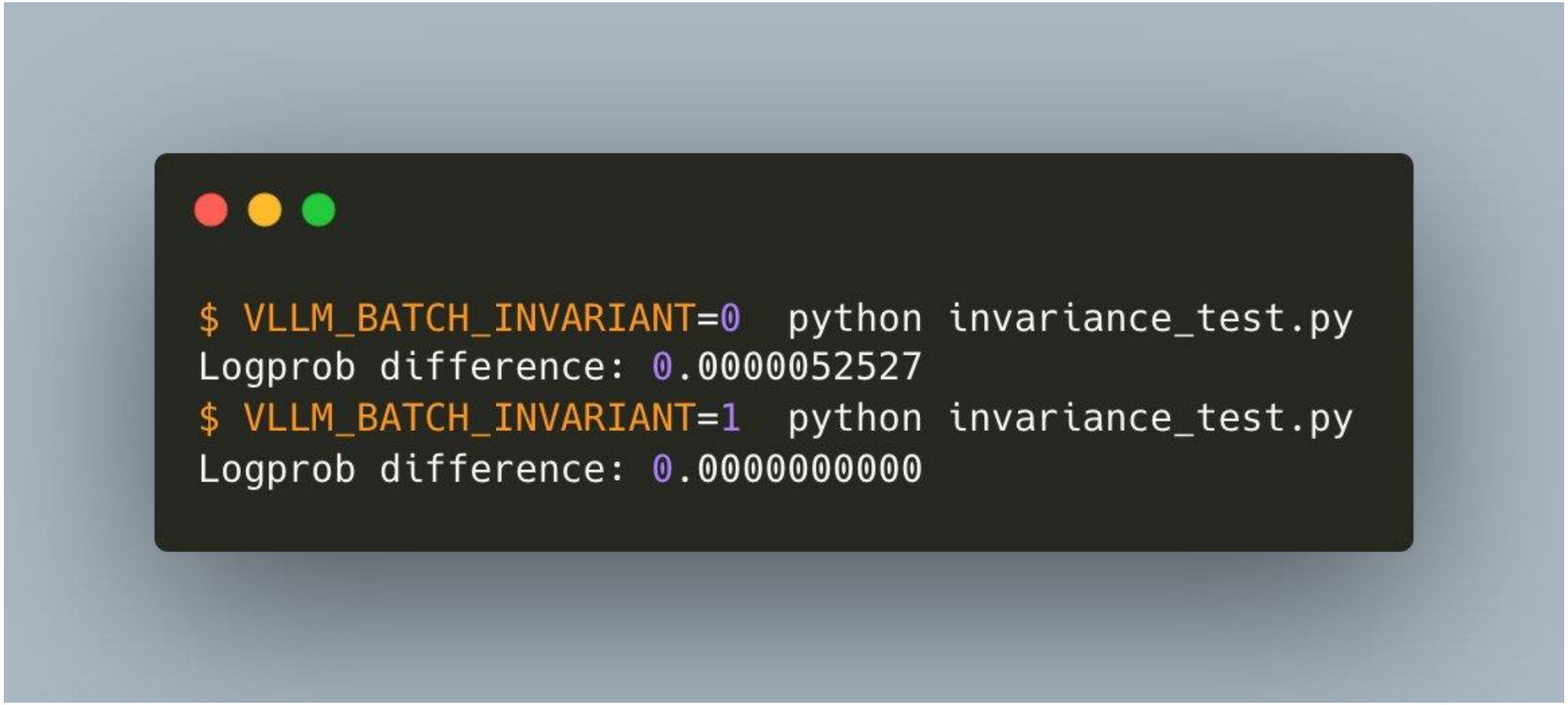
Reason 3: Continuous Batching





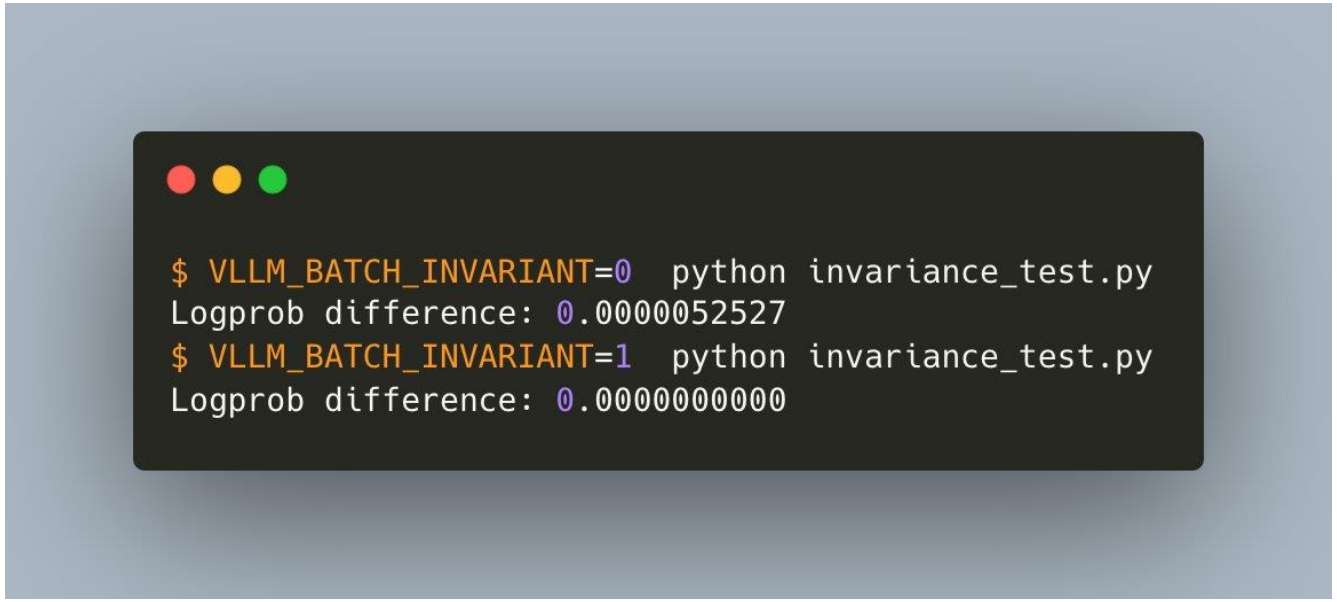
S5 now starts in a different place

How to solve this? Batch invariance



It ensures the output of a model is deterministic and independent of the batch size or the order of requests in a batch (fixed reduction order).

How to Achieve Batch Invariance?



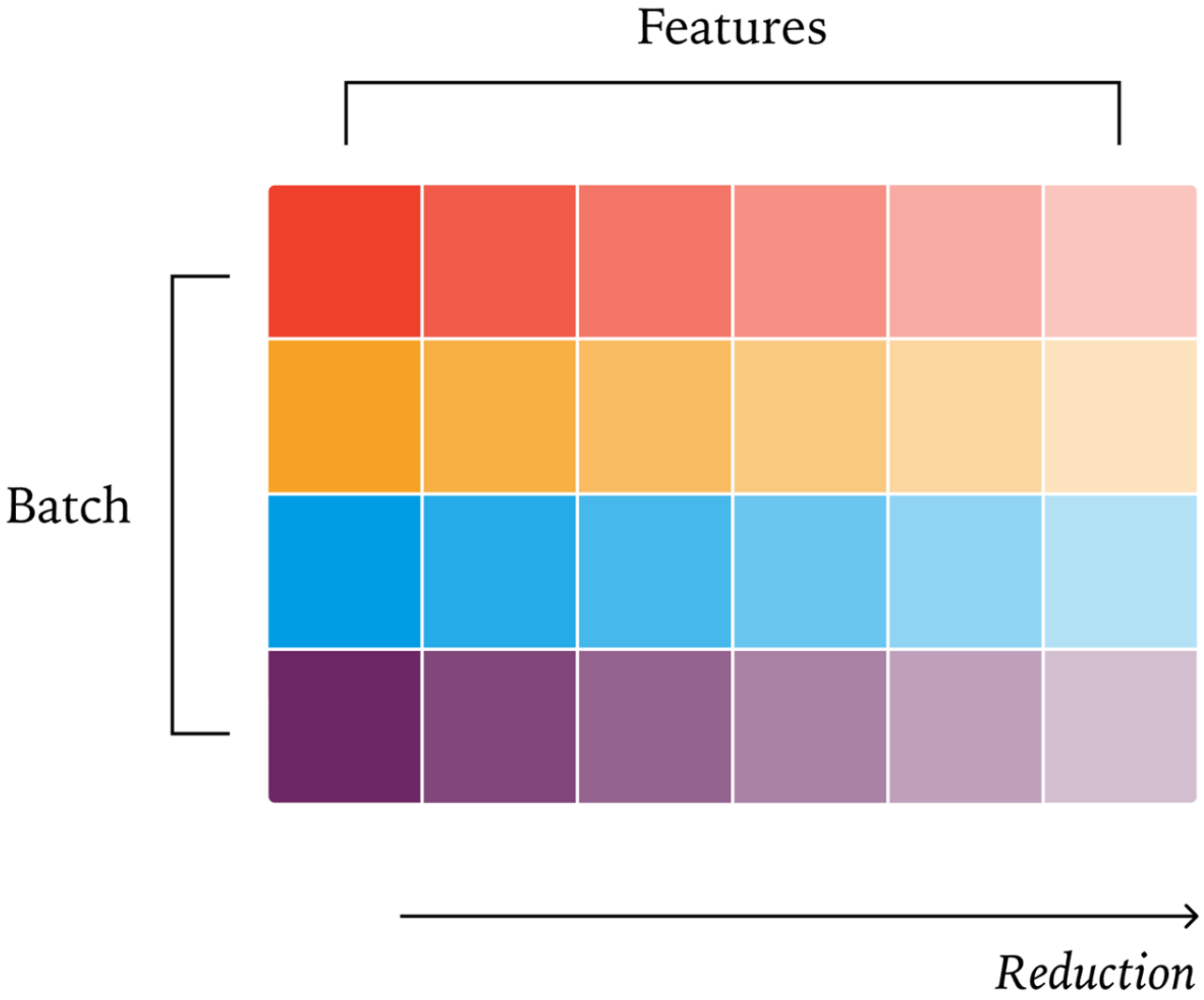
Make every kernel involving reductions batch invariant

For LLM inference, **three kernels dominate**

- RMSNorm
- Matrix Mul
- Attention

Key 1: RMSNorm

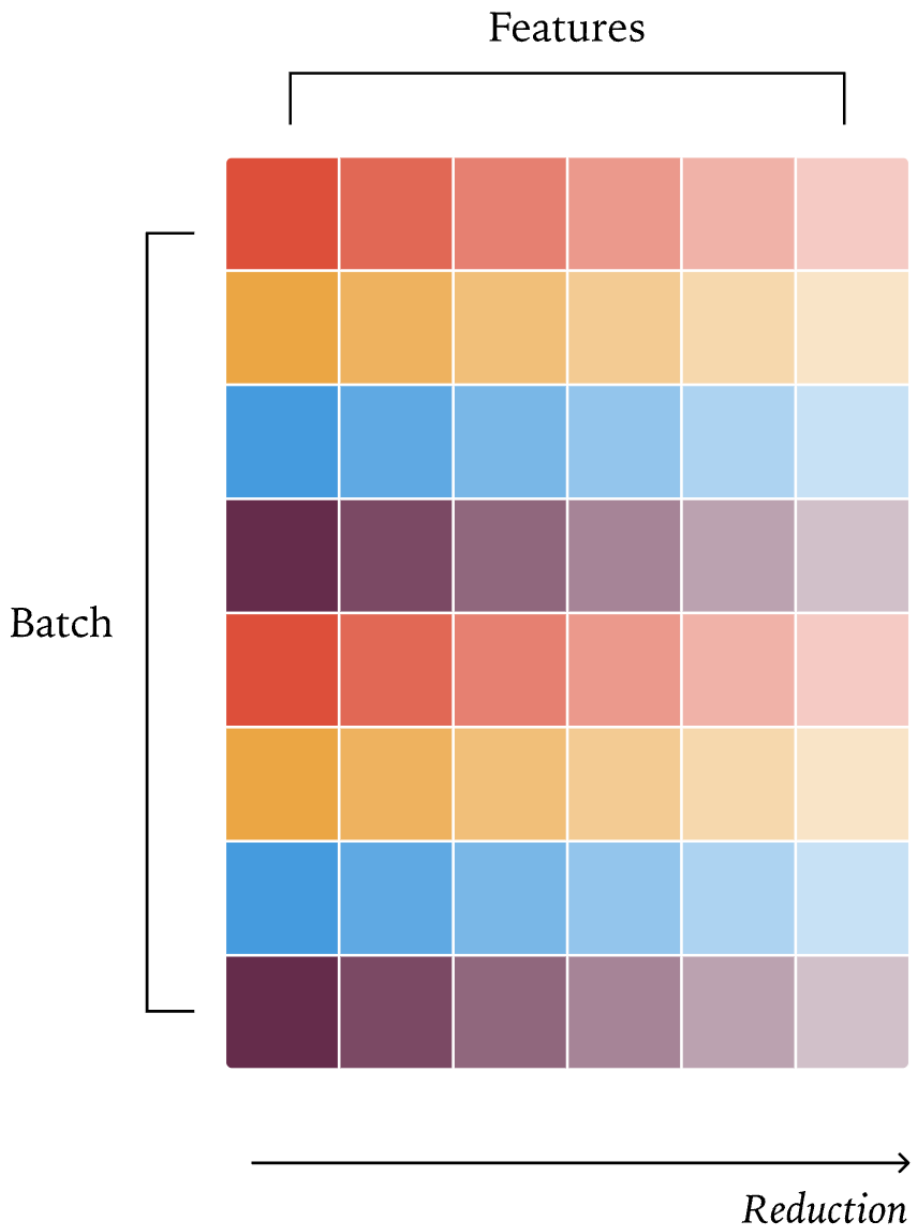
$$y = x \cdot \text{rsqrt}\left(\text{mean}(x^2) + \epsilon\right) \cdot w$$



In naive solution, parallel through rows

Key 1: RMSNorm

$$y = x \cdot \text{rsqrt}\left(\text{mean}(x^2) + \epsilon\right) \cdot w$$



In large batch, this works well

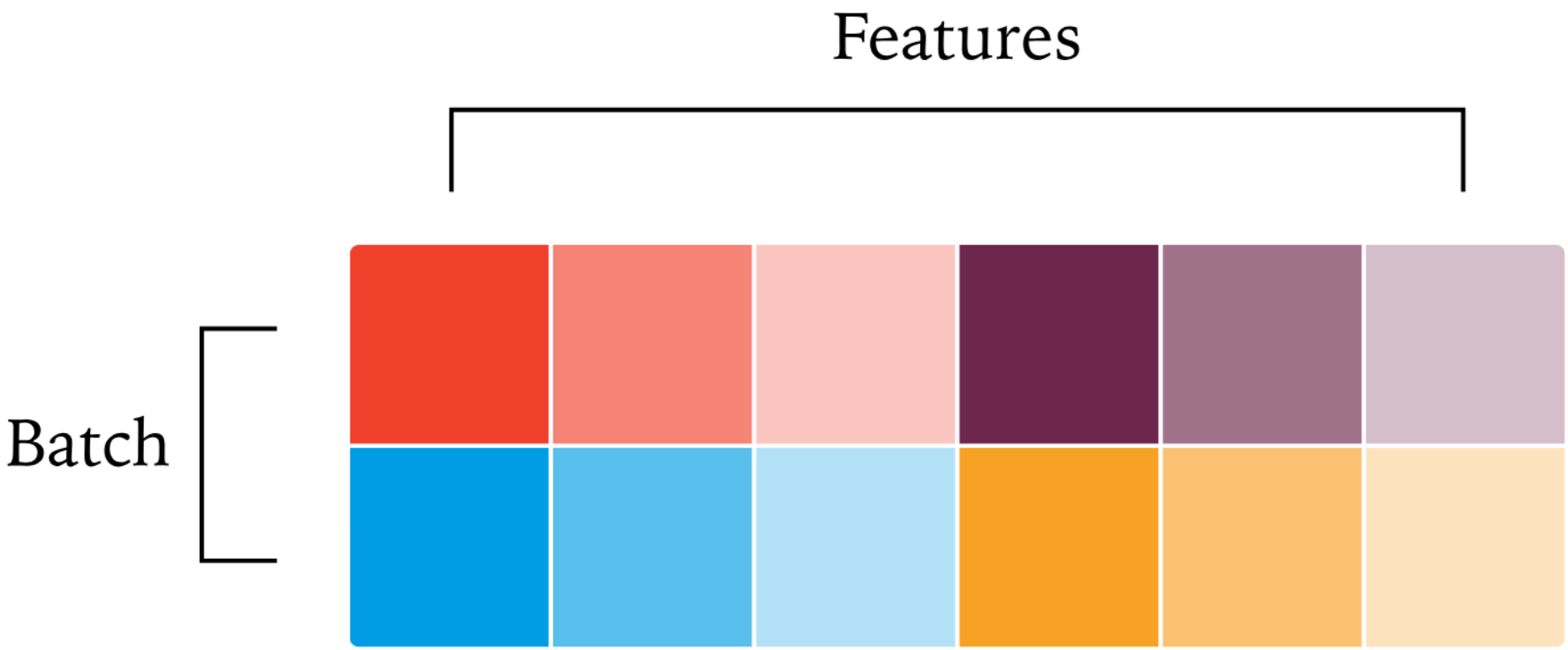
Key 1: RMSNorm

$$y = x \cdot \text{rsqrt}\left(\text{mean}(x^2) + \epsilon\right) \cdot w$$

In small batch, wasting compute resources

Key 1: RMSNorm

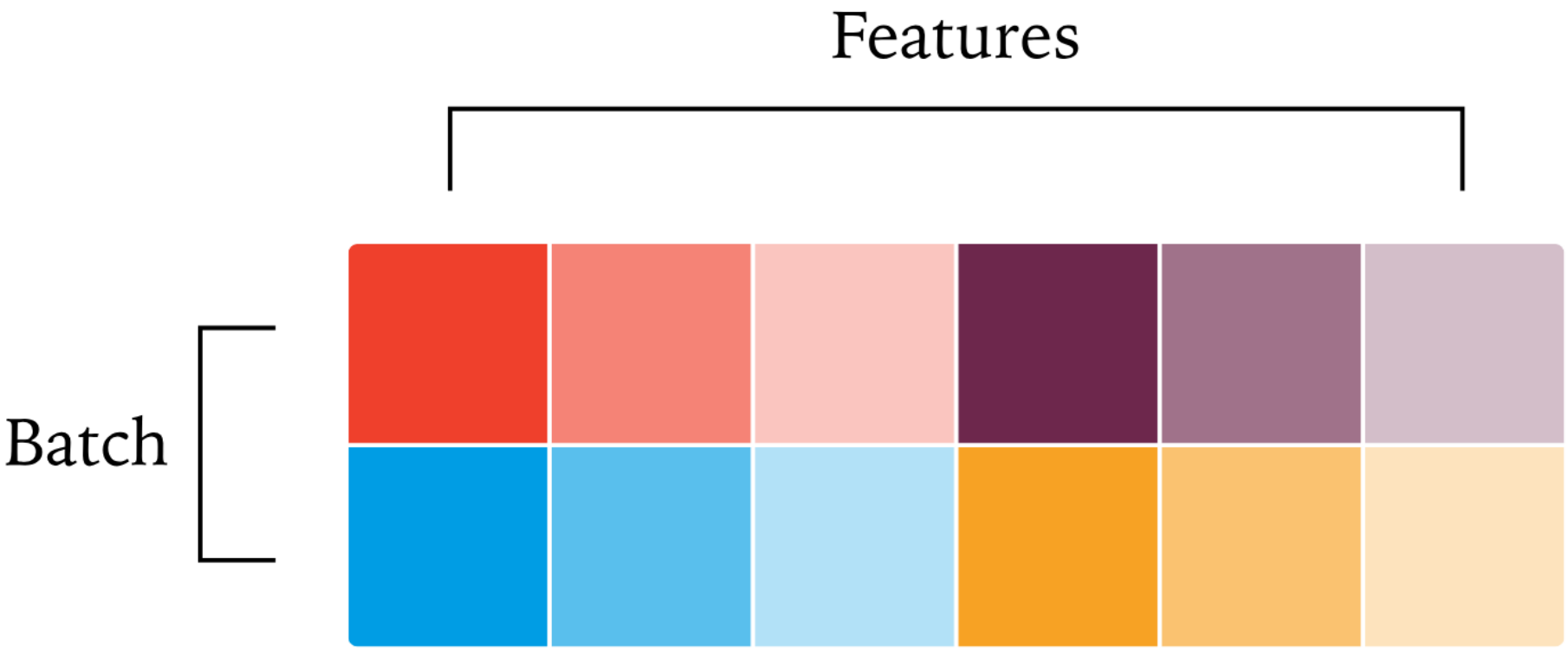
$$y = x \cdot \text{rsqrt}\left(\text{mean}(x^2) + \epsilon\right) \cdot w$$



In small batch, we usually optimize using Split-K

Key 1: RMSNorm

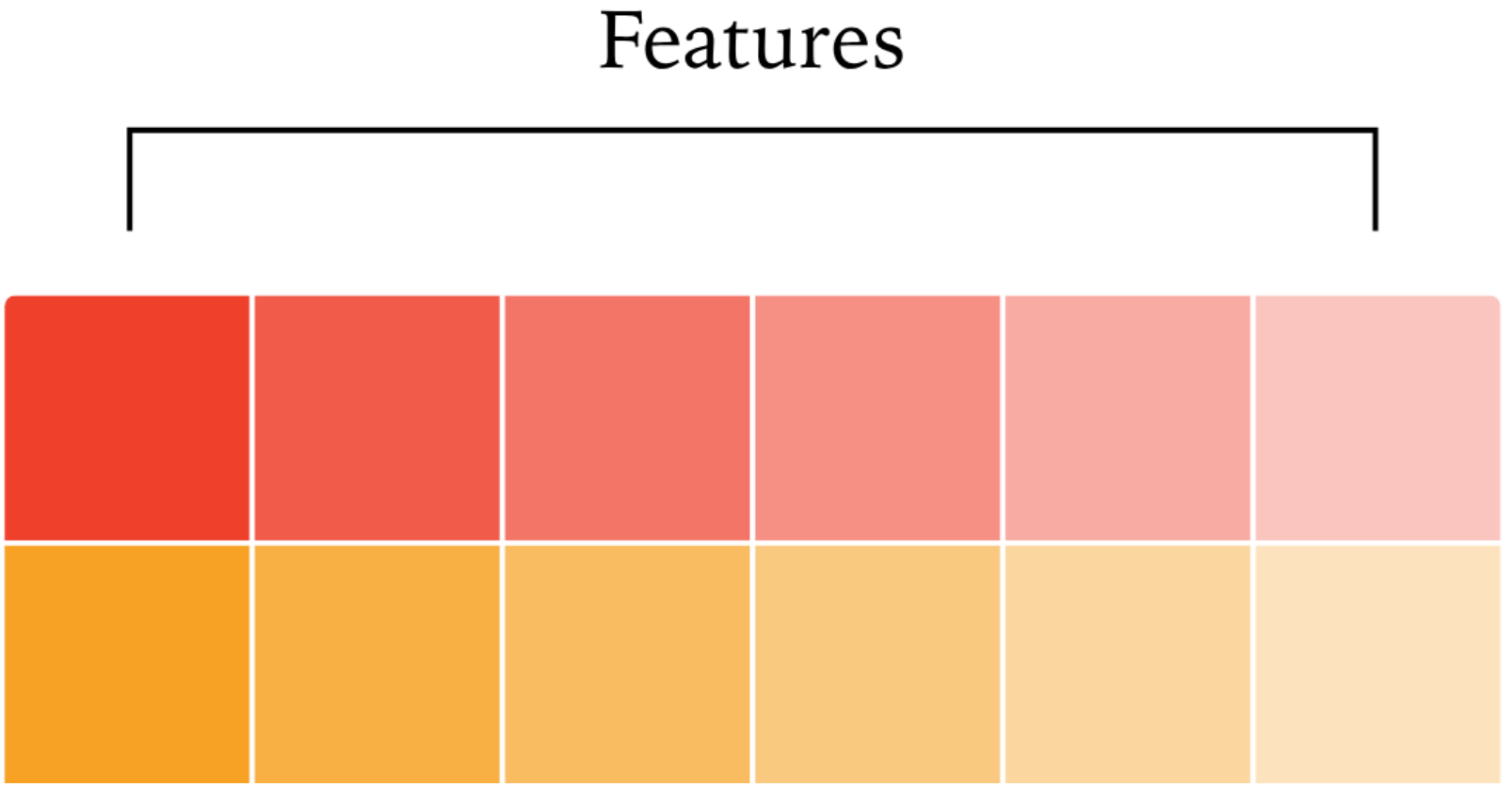
$$y = x \cdot \text{rsqrt}\left(\text{mean}(x^2) + \epsilon\right) \cdot w$$



However, as x grows, the reduction order changes

Key 1: RMSNorm

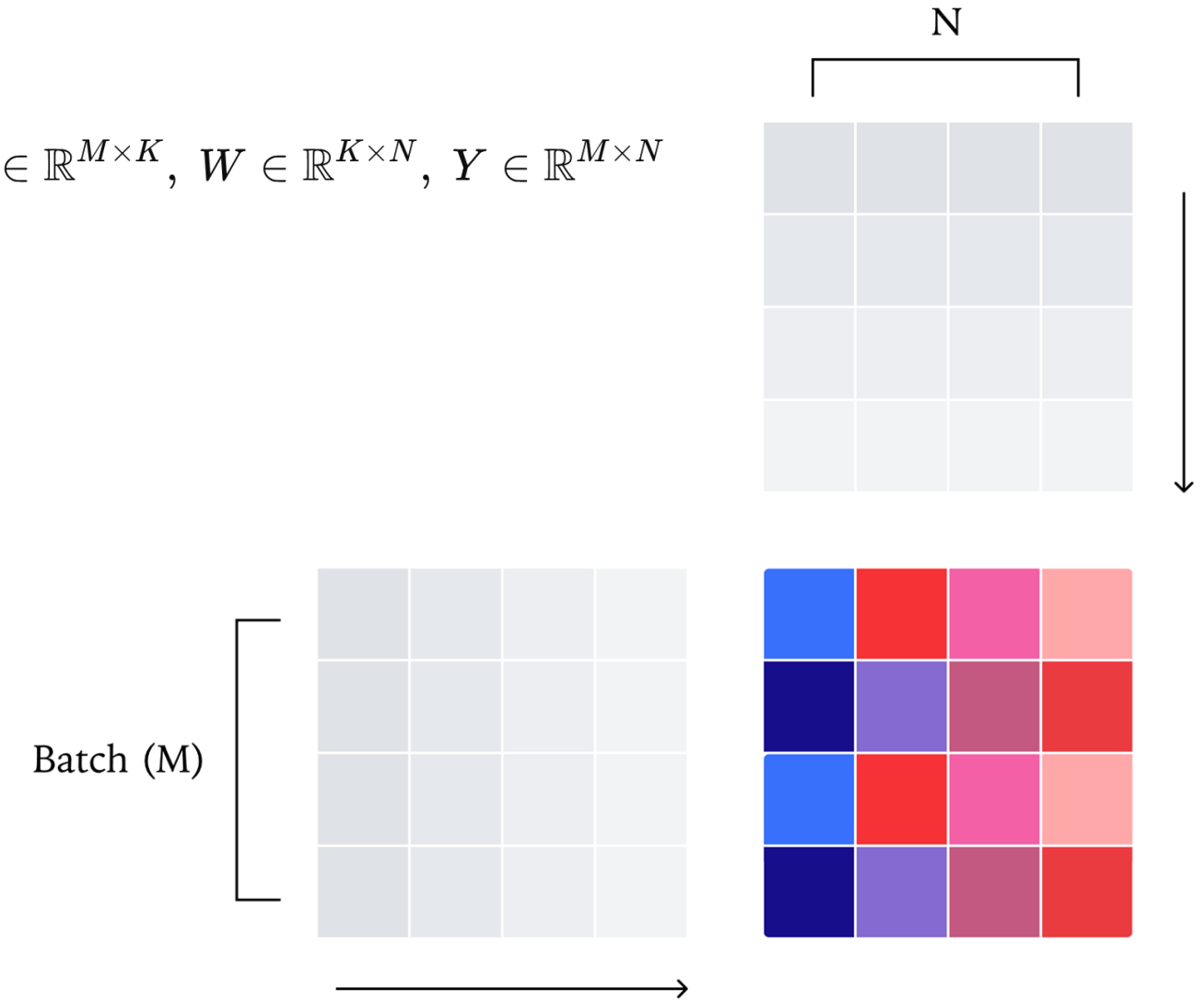
$$y = x \cdot \mathrm{rsqrt}\left(\mathrm{mean}(x^2) + \epsilon\right) \cdot w$$



The easiest way is just not to do the split and keep the order

Key 2: Matrix Mul

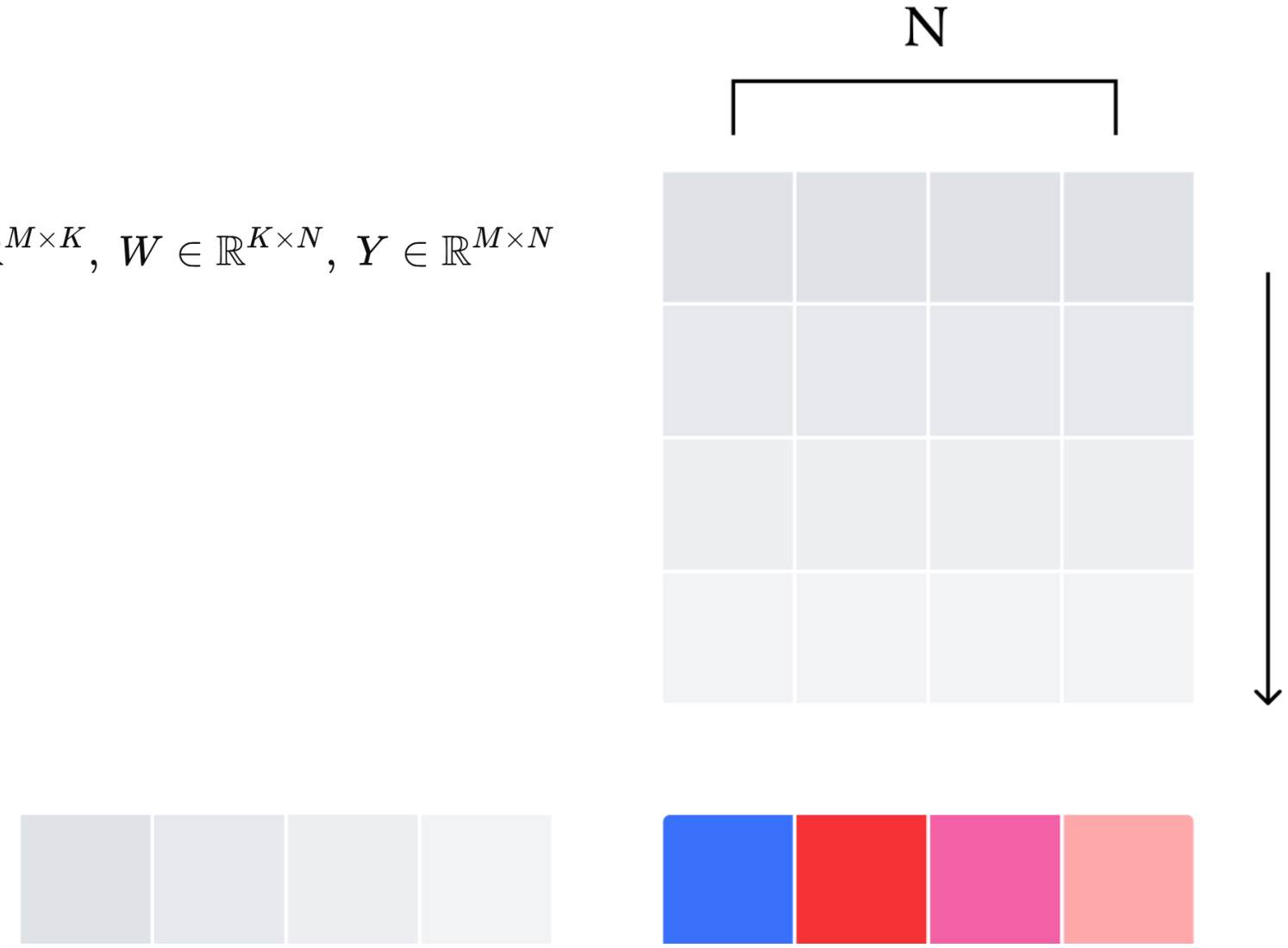
$$Y = XW, \quad X \in \mathbb{R}^{M \times K},\ W \in \mathbb{R}^{K \times N},\ Y \in \mathbb{R}^{M \times N}$$



The naive matrix mul follows the batch invariant logic, good for large batch

Key 2: Matrix Mul

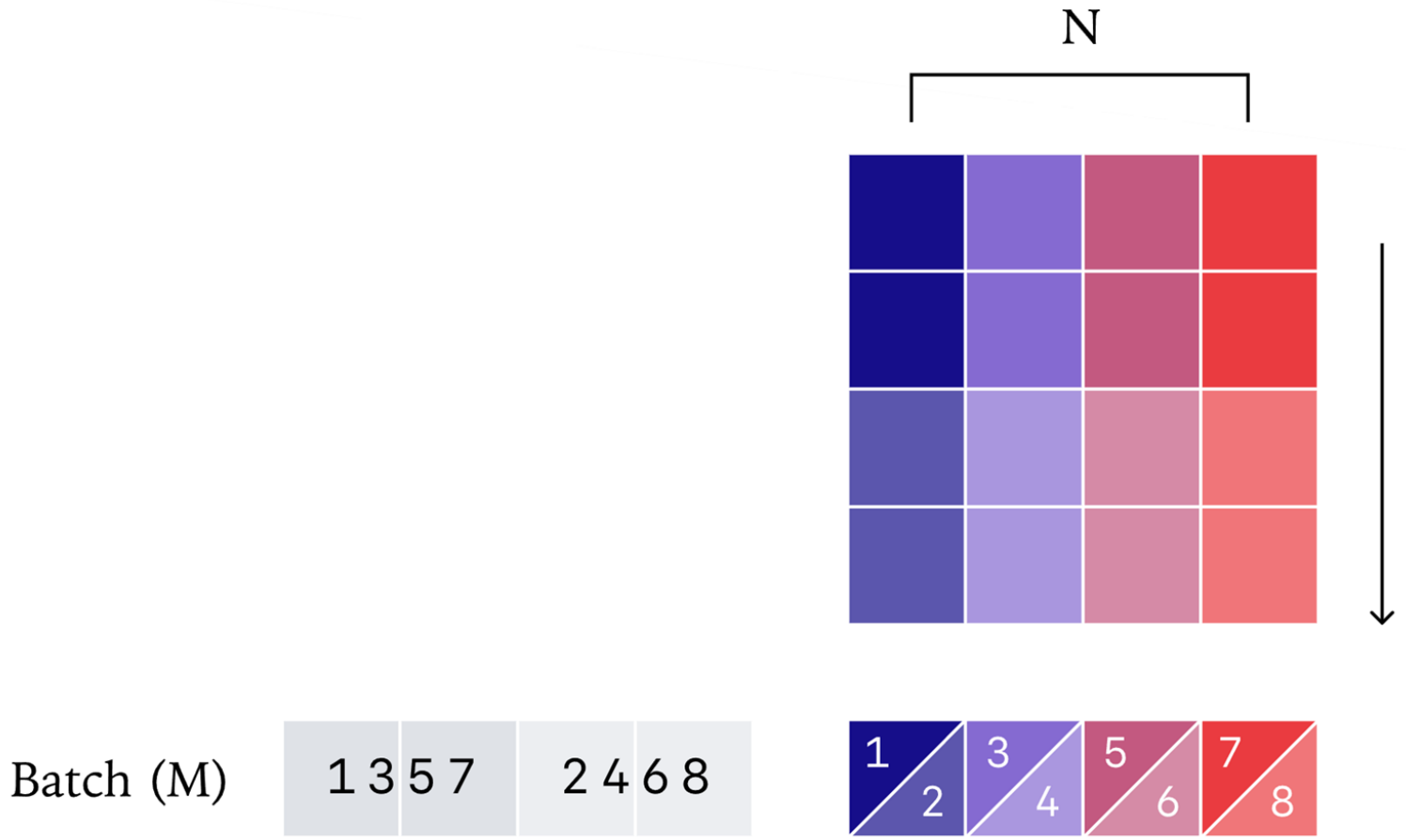
$$Y = XW, \quad X \in \mathbb{R}^{M \times K}, W \in \mathbb{R}^{K \times N}, Y \in \mathbb{R}^{M \times N}$$



Similar problem when batch size is small

Key 2: Matrix Mul

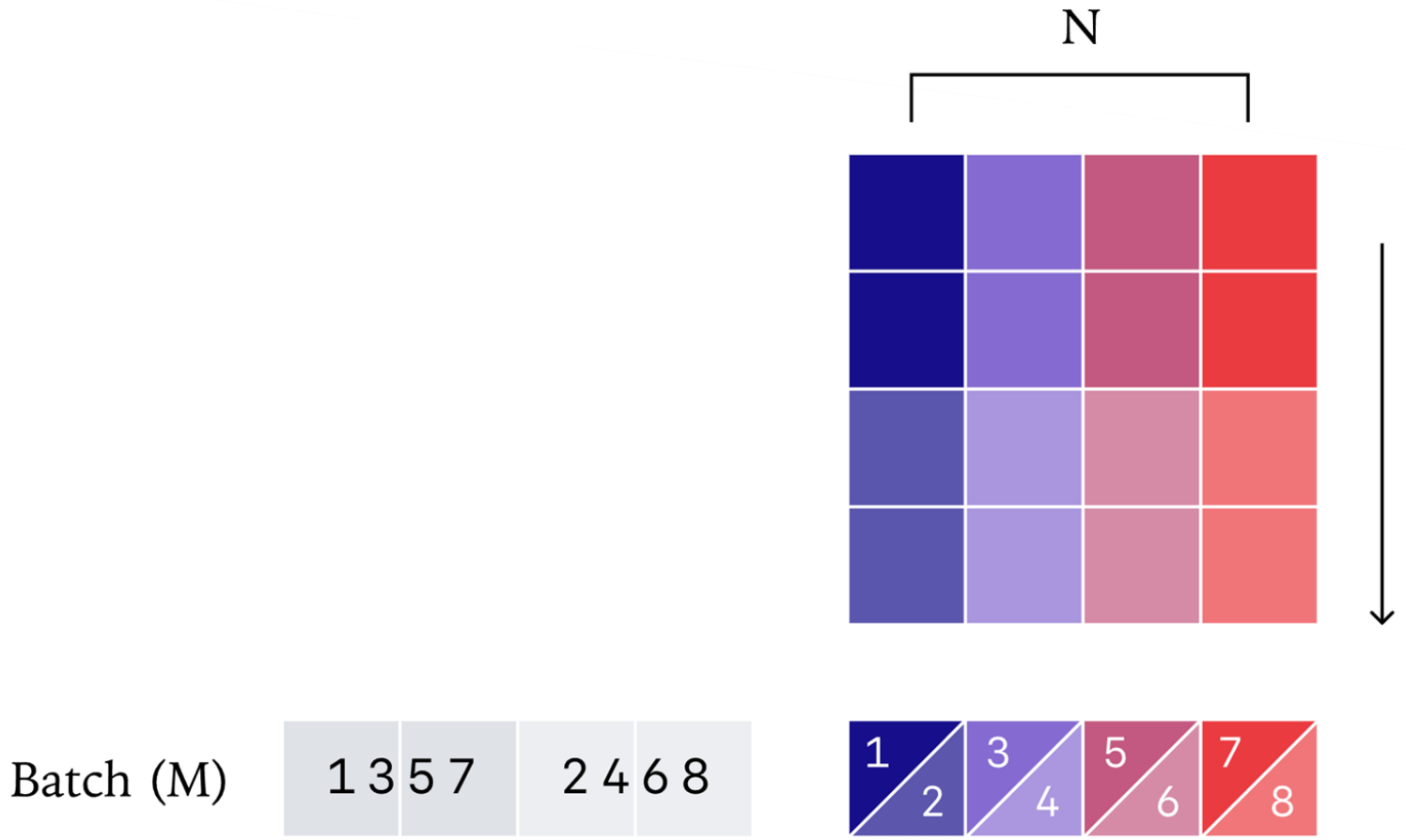
$$Y = XW, \quad X \in \mathbb{R}^{M \times K}, W \in \mathbb{R}^{K \times N}, Y \in \mathbb{R}^{M \times N}$$



In small batch, we usually optimize using Split-K

Key 2: Matrix Mul

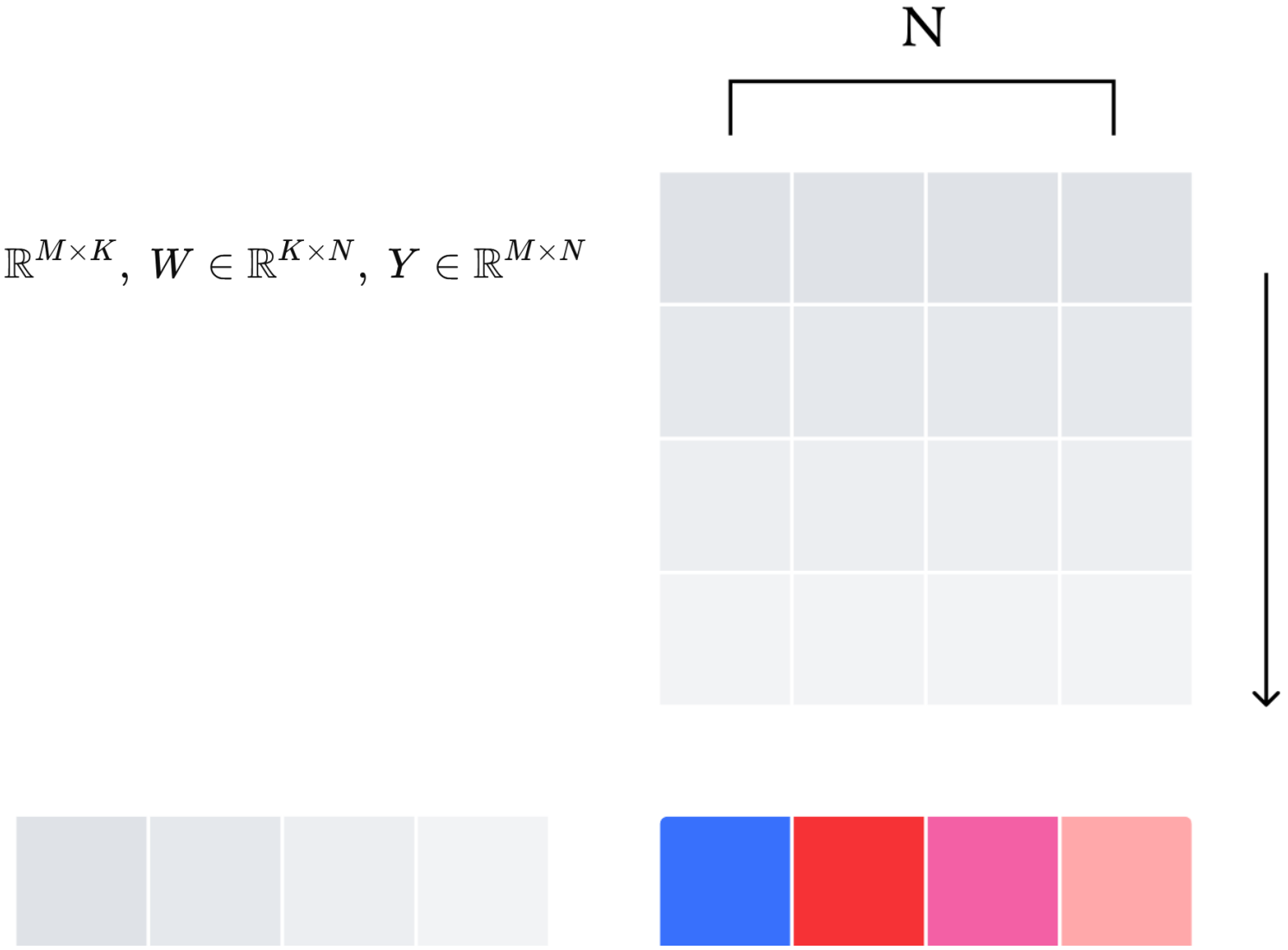
$$Y = XW, \quad X \in \mathbb{R}^{M \times K}, W \in \mathbb{R}^{K \times N}, Y \in \mathbb{R}^{M \times N}$$



This breaks the **order of reduction** as K grows

Key 2: Matrix Mul

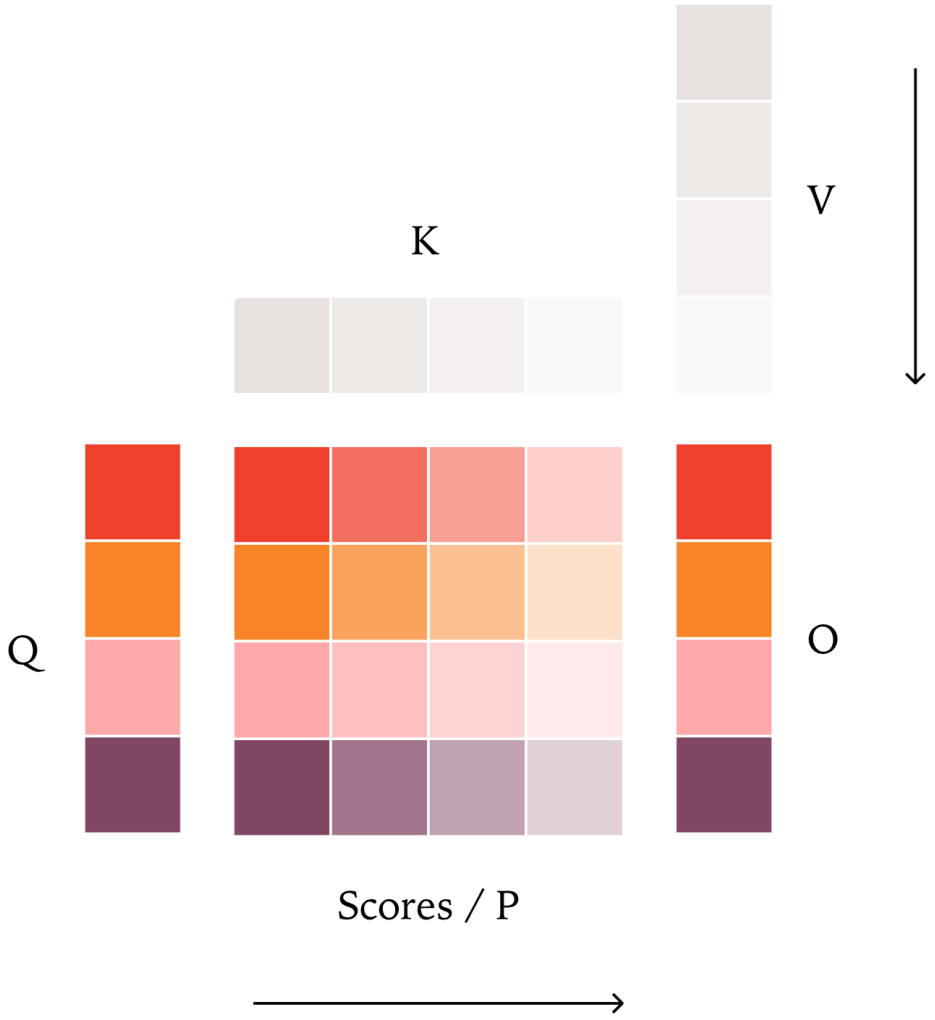
$$Y = XW, \quad X \in \mathbb{R}^{M \times K}, W \in \mathbb{R}^{K \times N}, Y \in \mathbb{R}^{M \times N}$$



Solution: compile one kernel configuration and use that for all shapes (fixed tile, fixed split-k etc)

Key 3: Attention

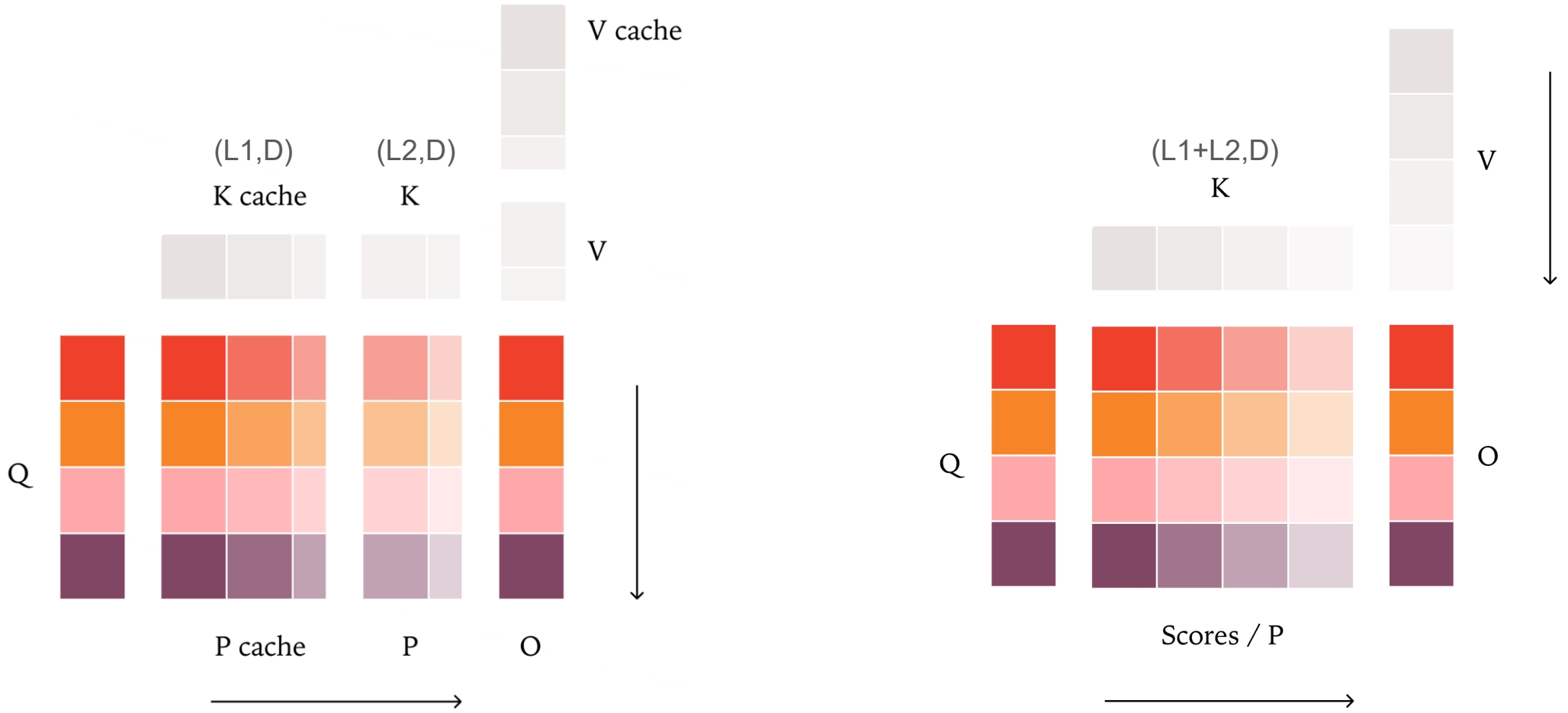
$$O = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$



Parallel along Q in naive solution, which is good for batch invariance

Key 3: Attention

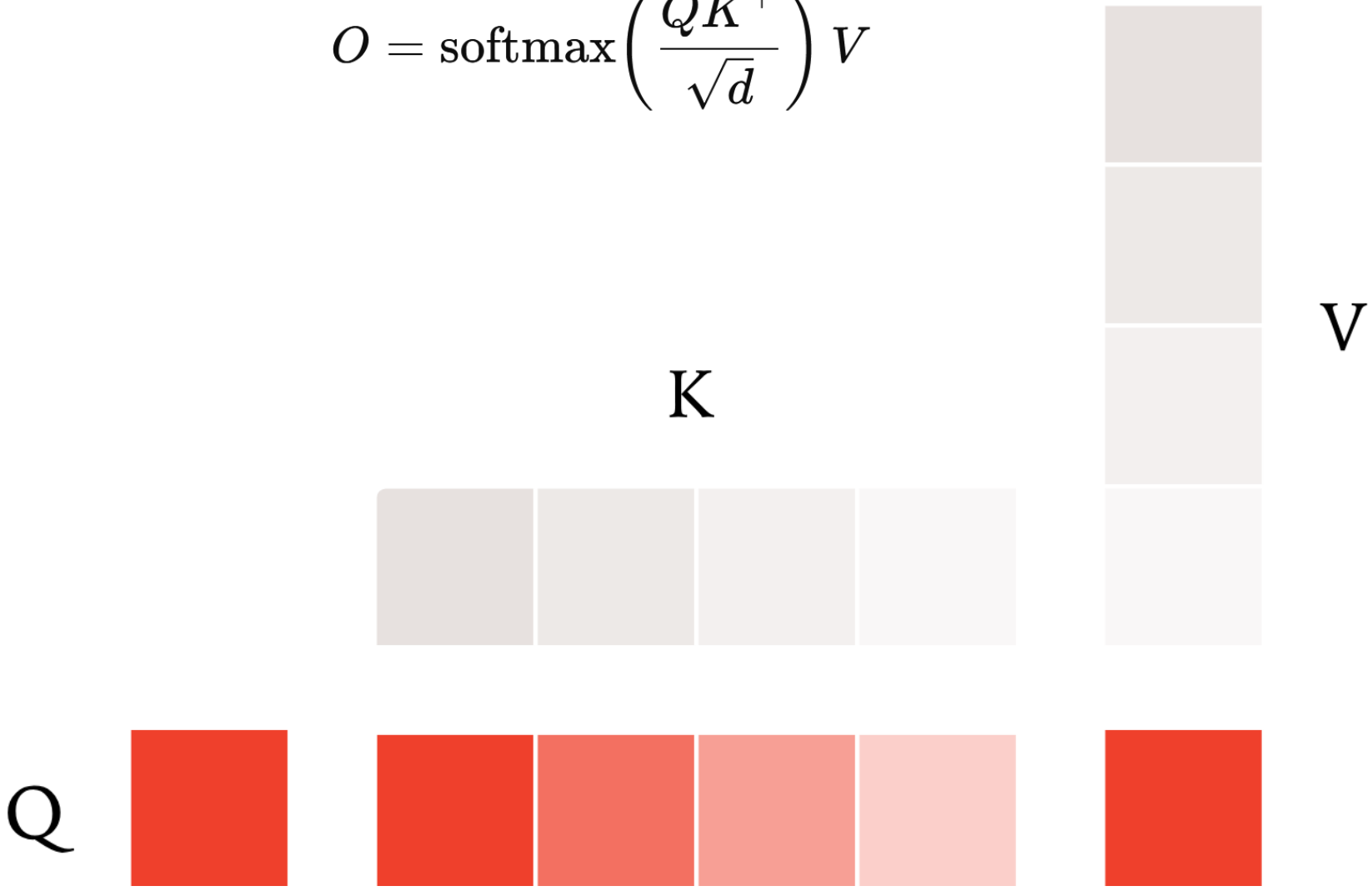
$$O = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$



Calculation for K, then added to the K-V cache

Key 3: Attention

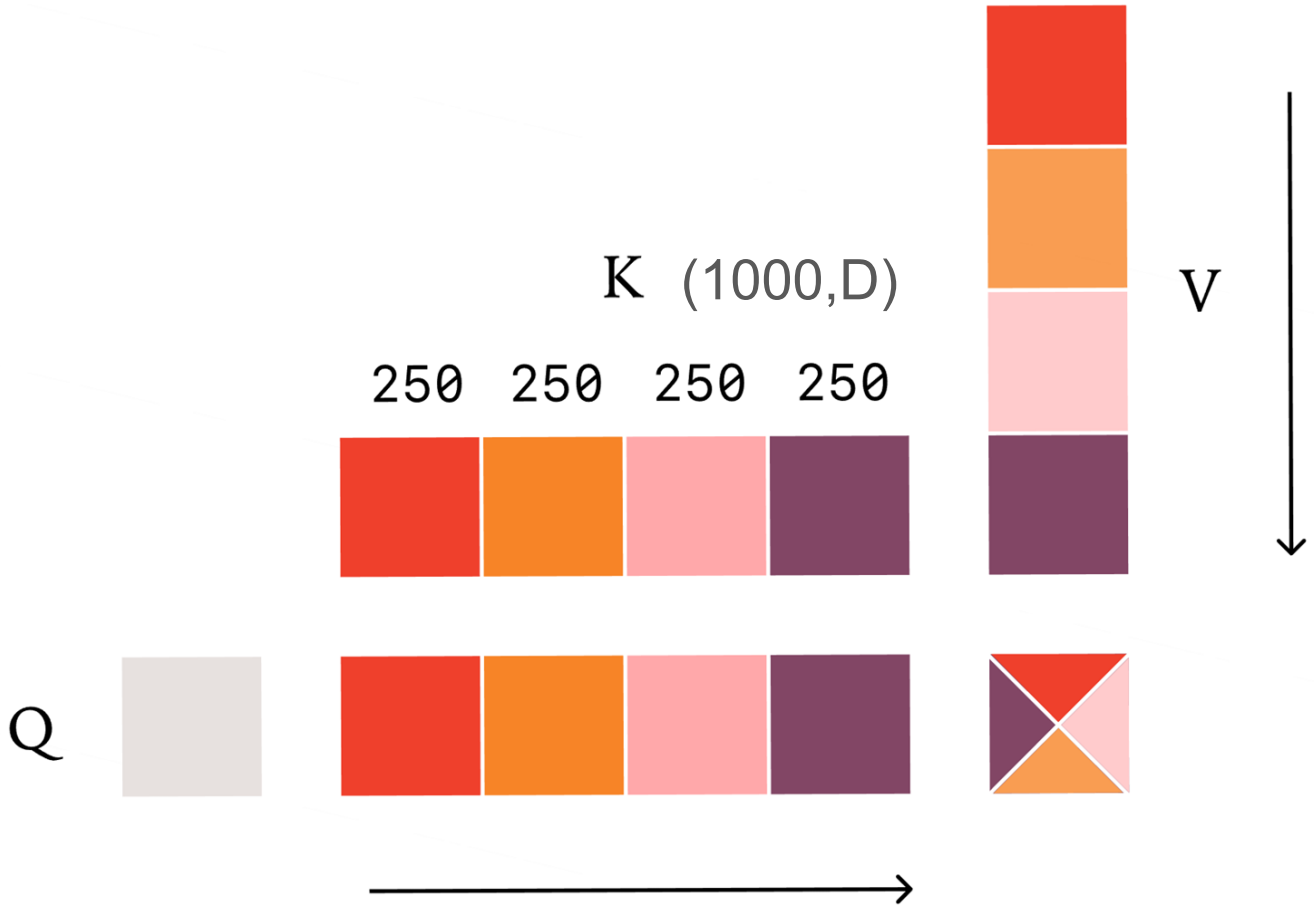
$$O = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$



Similar problem when batch is small

Key 3: Attention

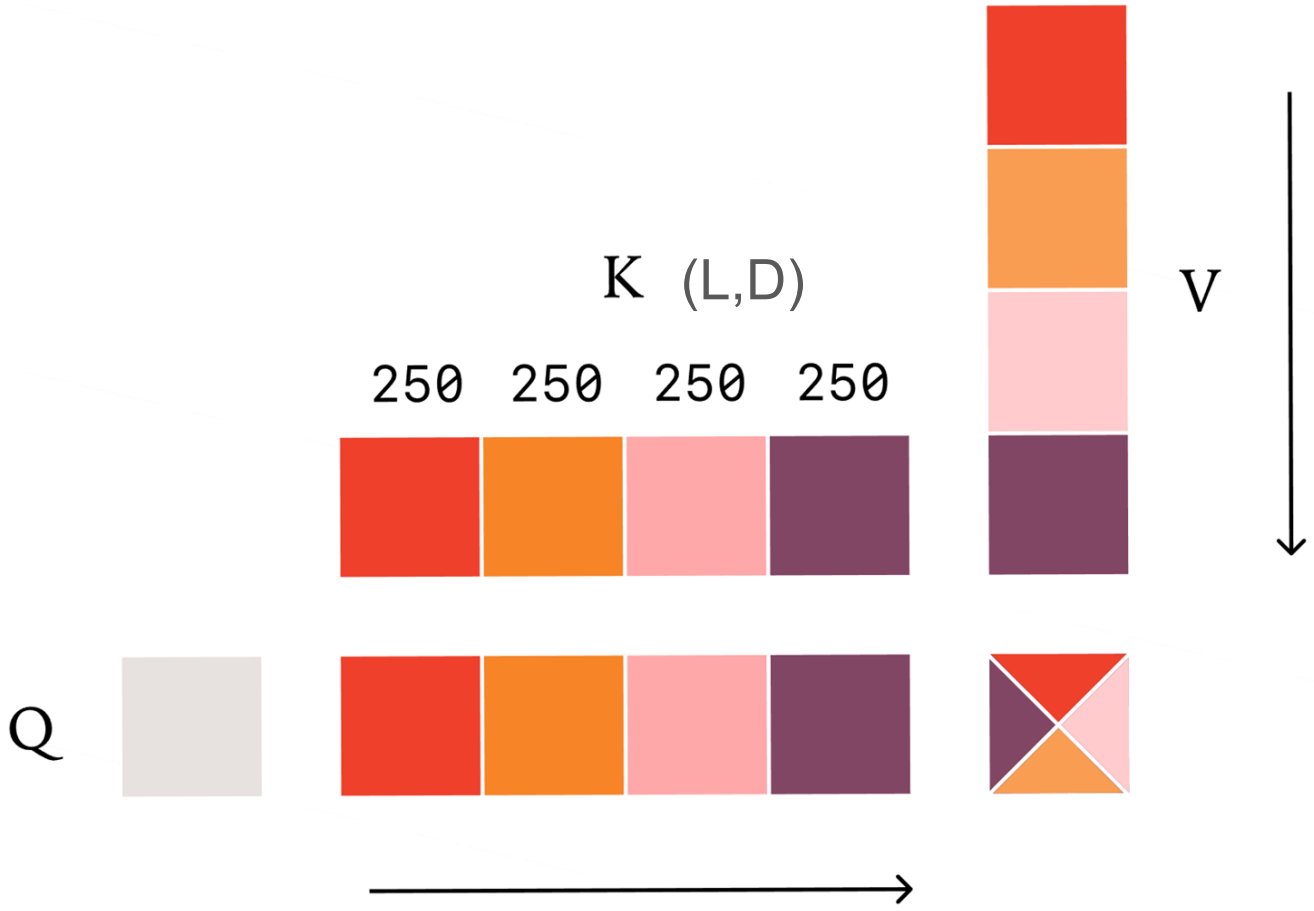
$$O = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$



We usually optimize using Split-K

Key 3: Attention

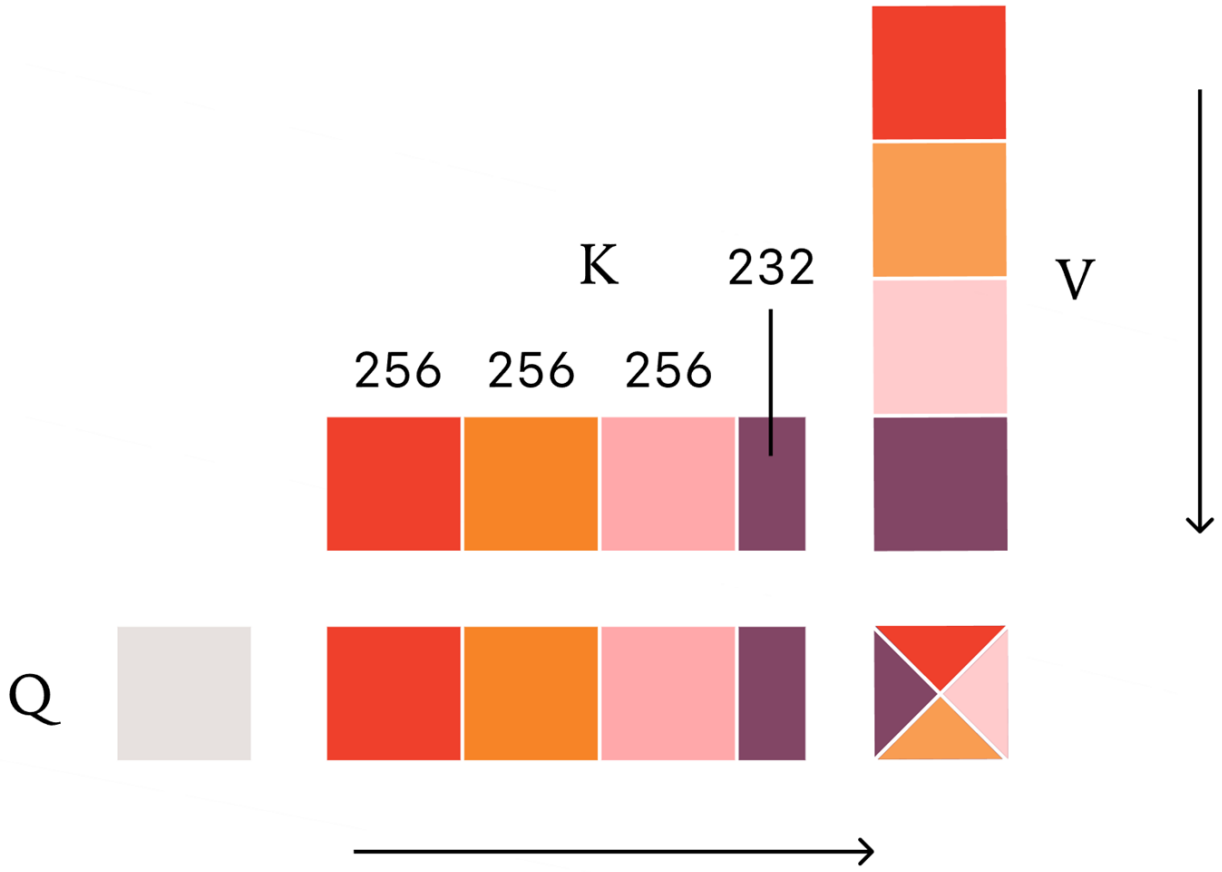
$$O = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$



Determinism broken as L increases

Key 3: Attention

$$O = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

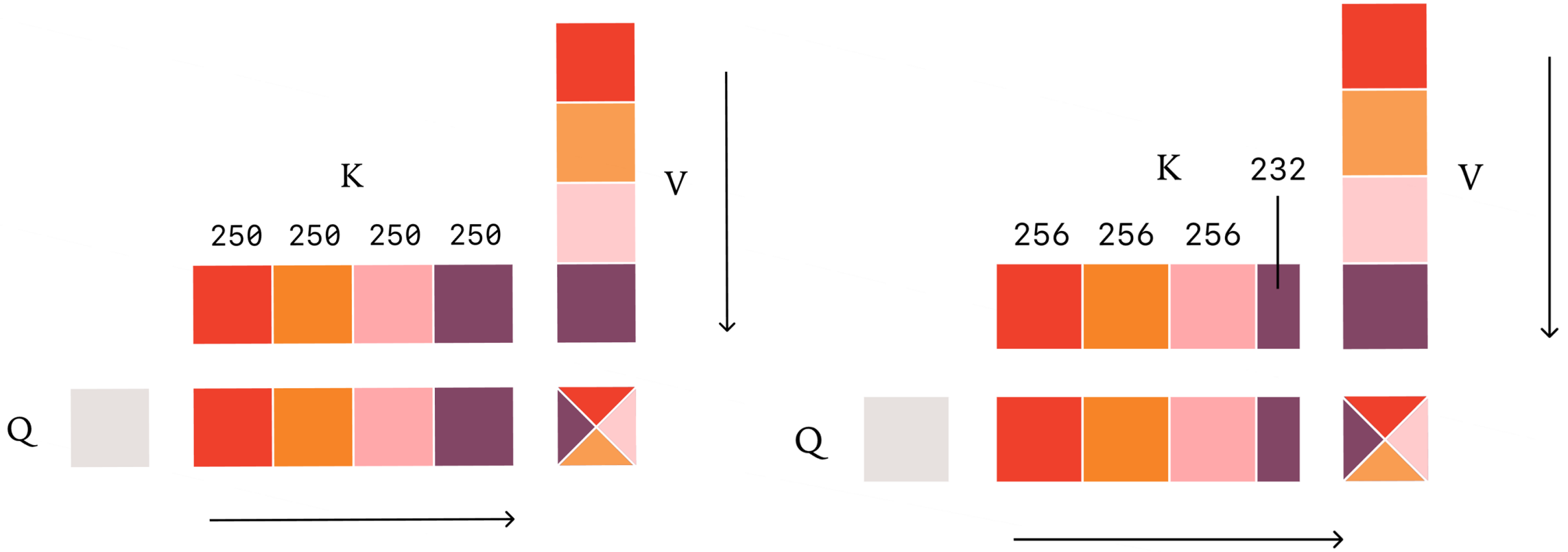


So we should use a fixed split size

e.g., 256

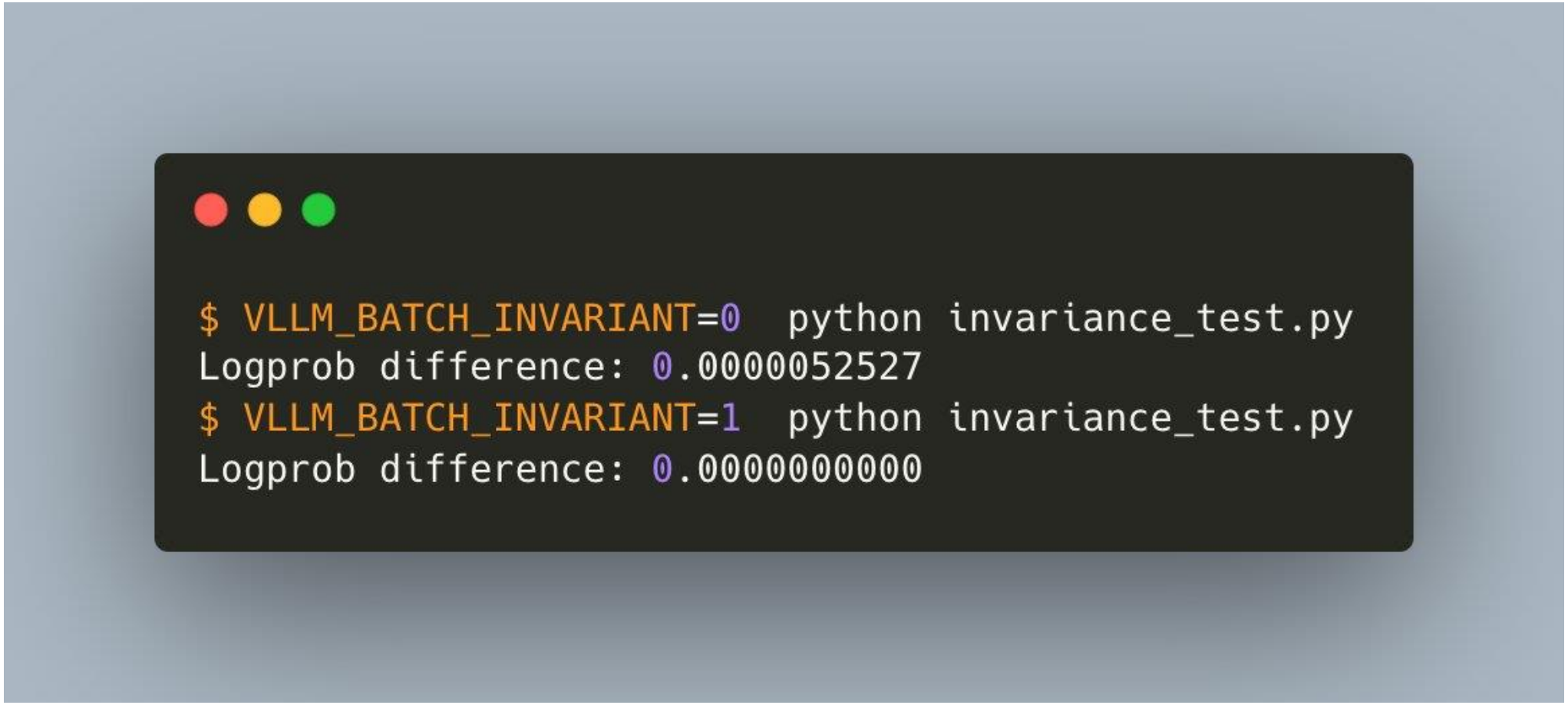
Key 3: Attention

$$O = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$



Detailed comparison

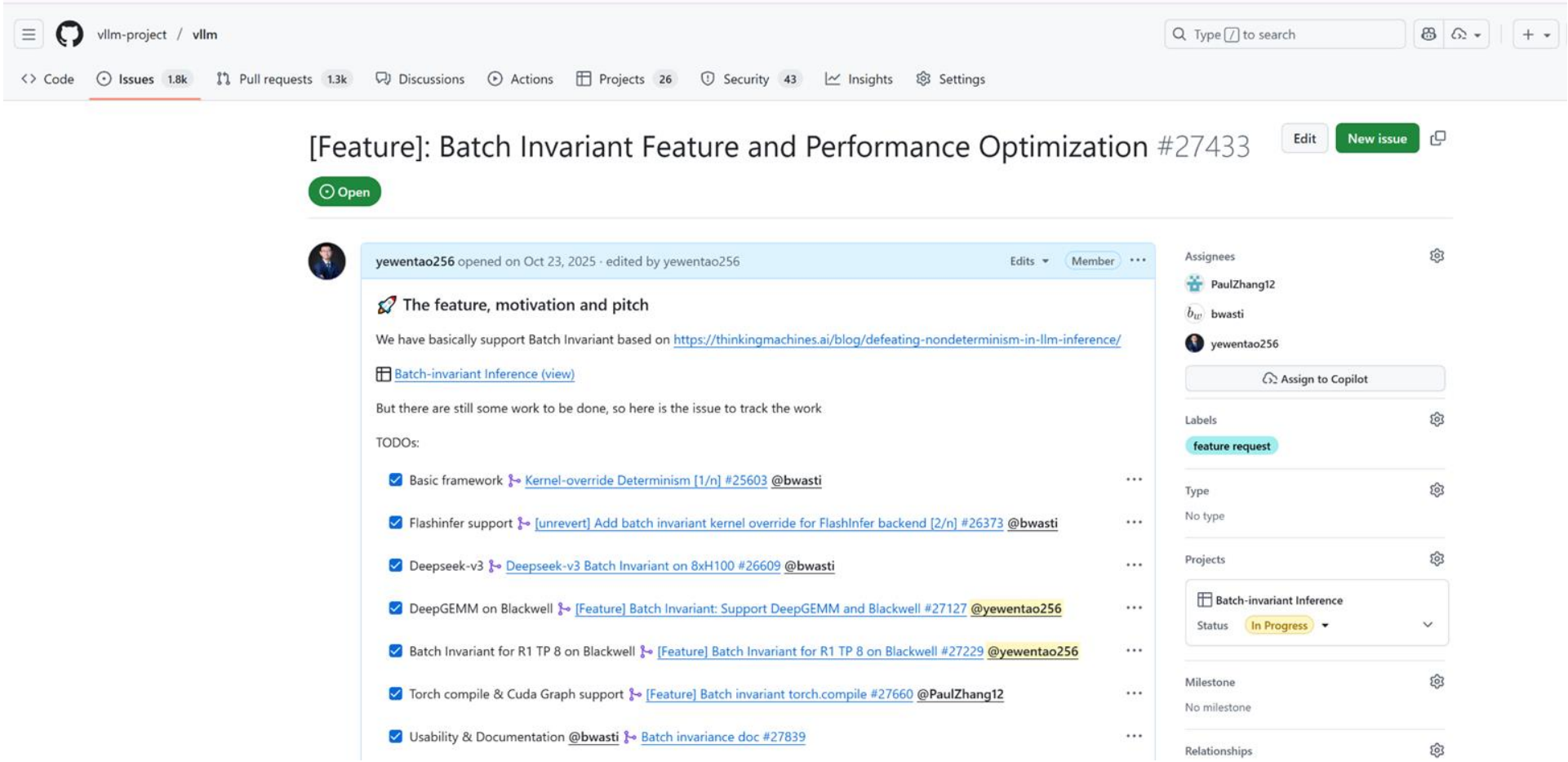
Batch invariance summary



Every kernel is batch invariant, with fixed reduction order

**Simplified due to time constraints.

Tracking the Latest Updates



https://github.com/vllm-project/vllm/issues/27433

Features, optimizations, model validations…
Any help is greatly appreciated!

Contribution to Batch Invariance in vLLM

Tested Models ¶

Batch invariance has been tested and verified on the following models:

- **DeepSeek series**: `deepseek-ai/DeepSeek-V3`, `deepseek-ai/DeepSeek-V3-0324`, `deepseek-ai/DeepSeek-R1`, `deepseek-ai/DeepSeek-V3.1`
- **Qwen3 (Dense)**: `Qwen/Qwen3-1.7B`, `Qwen/Qwen3-8B`
- **Qwen3 (MoE)**: `Qwen/Qwen3-30B-A3B`, `Qwen/Qwen3-Next-80B-A3B-Instruct`
- **Llama 3**: `meta-llama/Llama-3.1-8B-Instruct`, `meta-llama/Llama-3.2-1B-Instruct`

🙋 Help needed for validations of more models.

1. Test a model using the current script
2. Submit a PR updating the document

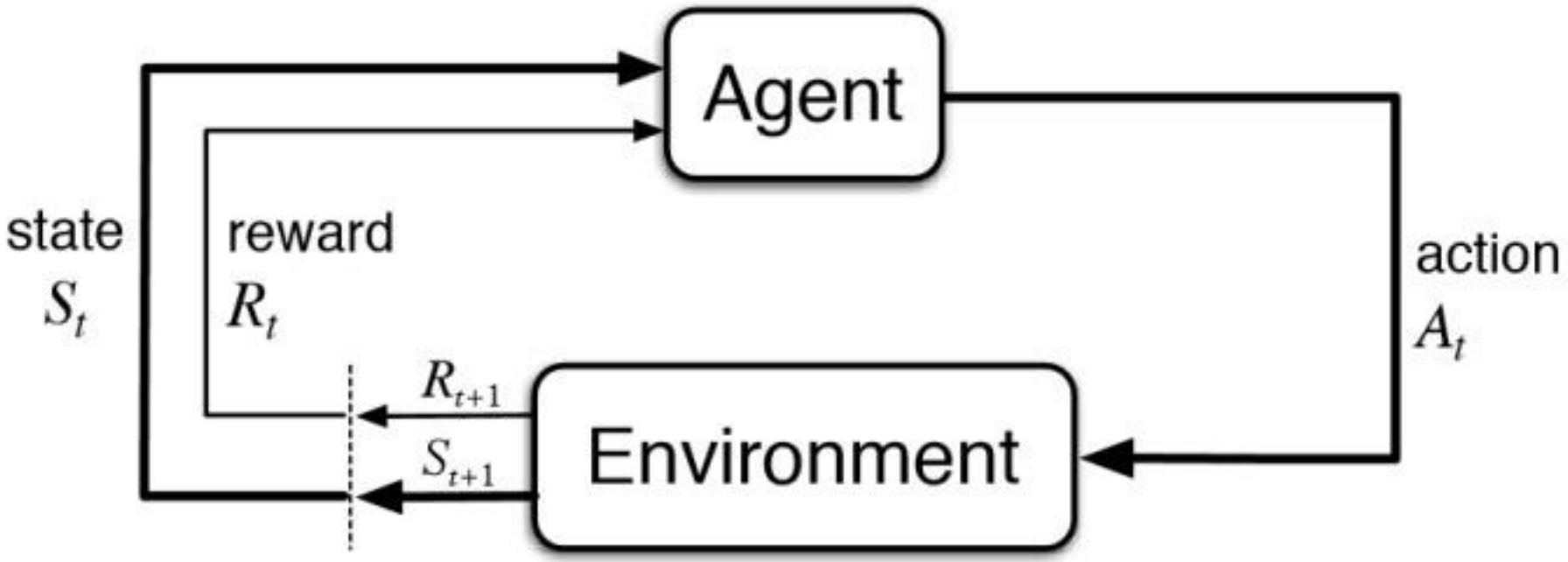
https://github.com/vllm-project/vllm/issues/27433

References

[1]: https://docs.vllm.ai/en/latest/features/batch_invariance/ "Batch Invariance - vLLM"
[2]: https://vllm.ai/ "vLLM"
[3]: https://arxiv.org/html/2506.09501v2 "Understanding and Mitigating Numerical Sources of Nondeterminism in LLM Inference"
[4]: https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/ "Defeating Nondeterminism in LLM Inference - Thinking Machines Lab"
[5]: https://github.com/vllm-project/vllm/issues/27433 "[Feature]: Batch Invariant Feature and Performance Optimization · Issue #27433 · vllm-project/vllm · GitHub"
[6]: https://blog.vllm.ai/2025/11/10/bitwise-consistent-train-inference.html "No More Train-Inference Mismatch: Bitwise Consistent On-Policy Reinforcement Learning with vLLM and TorchTitan | vLLM Blog"
[7]:https://insujang.github.io/2024-01-07/llm-inference-continuous-batching-and-pagedattention/ LLM Inference: Continuous Batching and PagedAttention

Bitwise Consistent On-Policy Reinforcement Learning with vLLM + TorchTitan

Reinforcement Learning (Basics)



Ignore all these variable letter choices :P

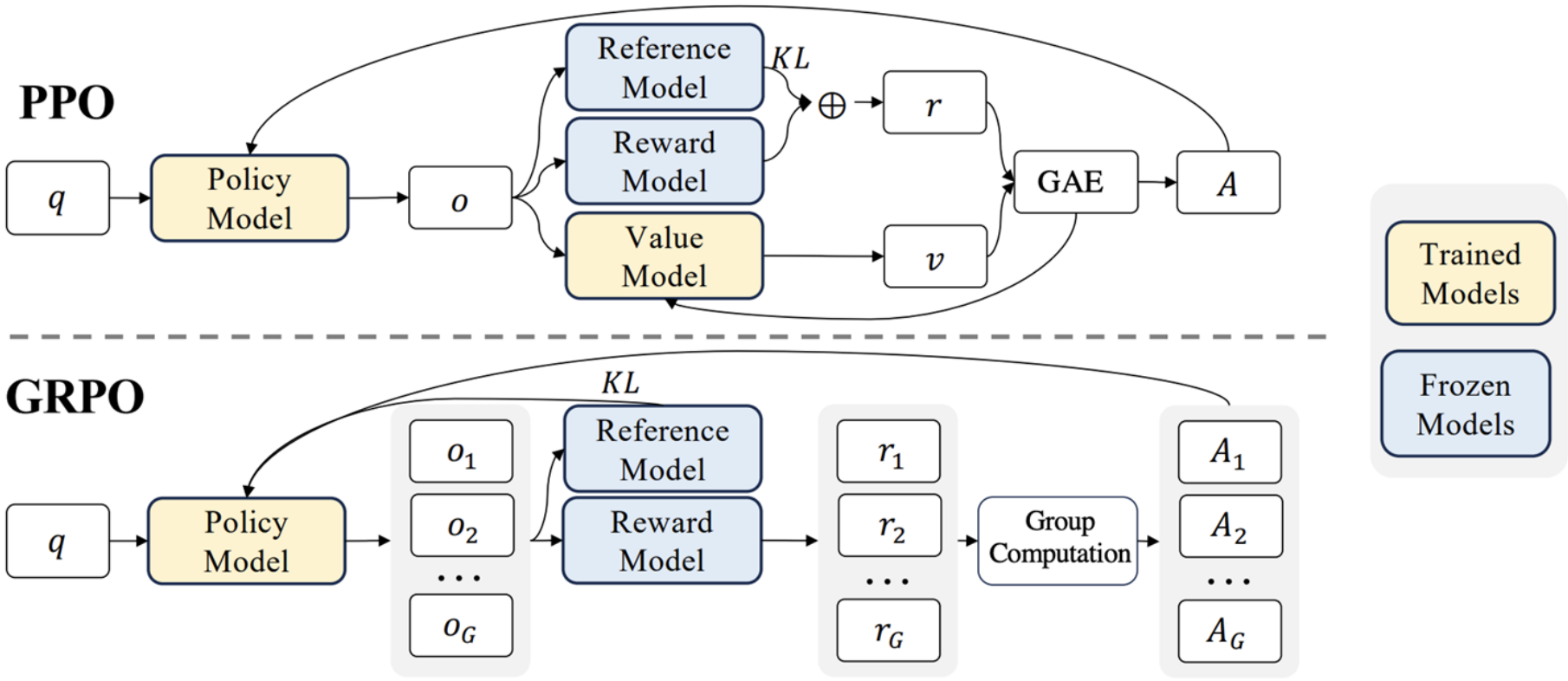
Reinforcement Learning (PPO)

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

https://arxiv.org/abs/1707.06347

Advantage calculation is tricky

Reinforcement Learning (PPO -> GRPO)



https://arxiv.org/pdf/2402.03300

Reinforcement Learning (IS Weights)

Running off-policy needs a correction term!

$$
\begin{aligned}
J(\theta) &= \sum_{s \in \mathcal{S}} \rho^{\pi_{\theta_{\text{old}}}} \sum_{a \in \mathcal{A}} \left(\pi_\theta(a|s) \hat{A}_{\theta_{\text{old}}}(s,a) \right) \\
&= \sum_{s \in \mathcal{S}} \rho^{\pi_{\theta_{\text{old}}}} \sum_{a \in \mathcal{A}} \left(\beta(a|s) \frac{\pi_\theta(a|s)}{\beta(a|s)} \hat{A}_{\theta_{\text{old}}}(s,a) \right) && \text{; Importance sampling} \\
&= \mathbb{E}_{s \sim \rho^{\pi_{\theta_{\text{old}}}}, a \sim \beta} \left[\frac{\pi_\theta(a|s)}{\beta(a|s)} \hat{A}_{\theta_{\text{old}}}(s,a) \right]
\end{aligned}
$$

https://lilianweng.github.io/posts/2018-04-08-policy-gradient/#trpo

Reinforcement Learning (GRPO)

- Sparse rewards, last step in training pipeline
- Huge dependency on the execution of the main model
- Correction terms galore, masking numerical instability
- This blog comes out…

Your Efficient RL Framework Secretly Brings You Off-Policy RL Training

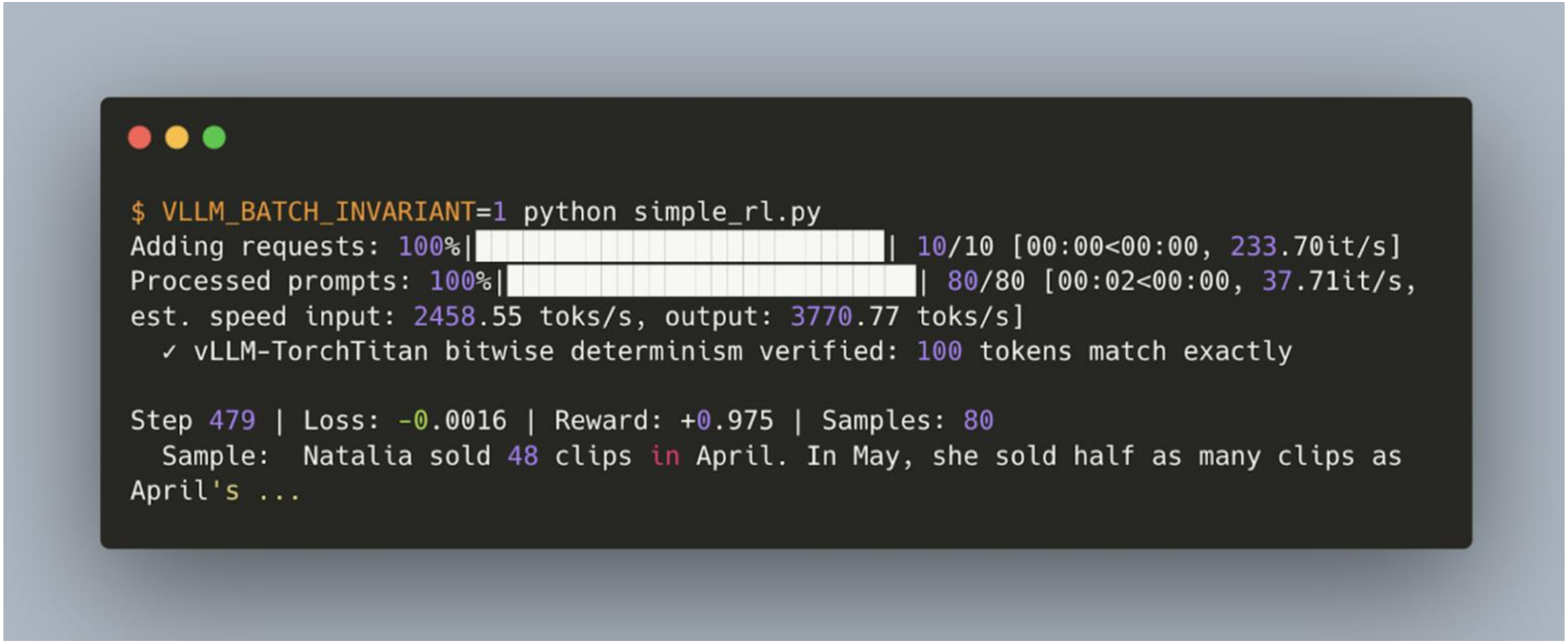
Feng Yao* Liyuan Liu* Dinghuai Zhang Chengyu Dong Jingbo Shang Jianfeng Gao

*: Equal Contributions (Work in Progress)

https://fengyao.notion.site/off-policy-rl

Reinforcement Learning (numerical exactness)

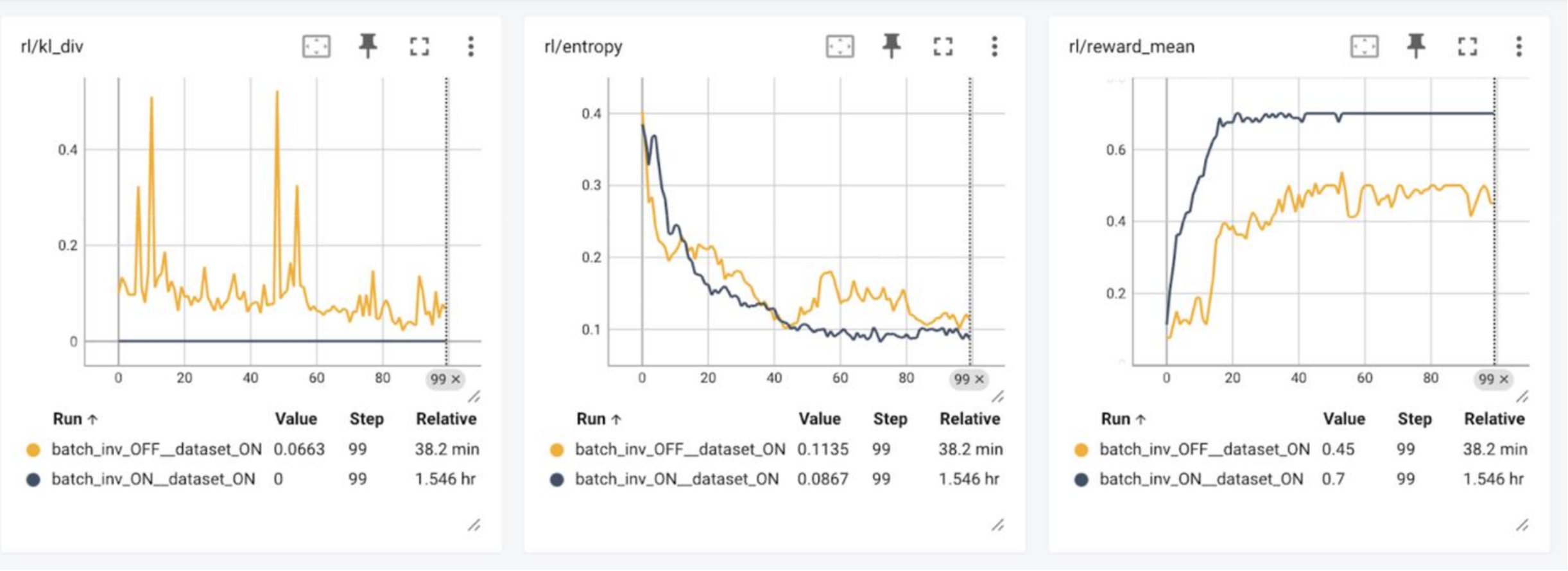
What if we ditch correction terms and use batch invariance?



https://blog.vllm.ai/2025/11/10/bitwise-consistent-train-inference.html

Reinforcement Learning (numerical exactness)

Works!



https://blog.vllm.ai/2025/11/10/bitwise-consistent-train-inference.html

Other stability tricks / Next Steps

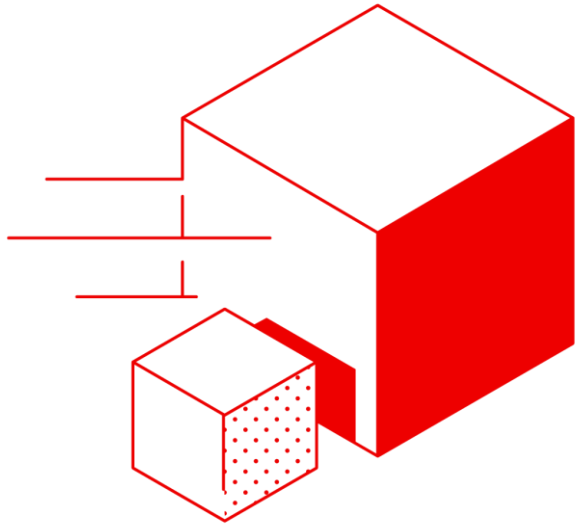
Important to note: there are other ways to stabilize RL training

- Other types of IS weights
- Use fp16 https://arxiv.org/pdf/2510.26788
- Rejection sampling magic

Next Steps

- Improve performance / hardware support
- Improve compilation support (native PyTorch)
- Figure out a unification scheme for training/inference

Get involved with the vLLM Community



Contribute to key vLLM features

Comment and review PRs that are interesting to you. Join the discussion on RFCs. Check out "good first issue" tags.

Give Us Feedback

We'll email you today's recording as soon as it's ready. Respond and tell us what we are doing right and what we can do better with vLLM office hours. Or comment on this slide!

Join vLLM Developer Slack

Ask questions and engage with us via Slack. Join here.

Join Red Hat's vLLM Mission

Red Hat wants to bring open-source LLMs and vLLM to every enterprise on the planet. We are looking for vLLM Engineers to help us accomplish our mission. Apply here.

Thank you, and see you in two weeks!

Michael Goin

Principal Engineer, Red Hat

vLLM Committer

Saša Zelenović

Principal PMM, Red Hat

Wentao Ye

Machine Learning Engineer, Red Hat

vLLM Committer

Bram Wasti

Software Engineer, Meta