

Engineering open LLMs for production



Dylan Bristot
Product Marketing Manager
Nebius
dylan@nebius.com



MaAI Developer Advocate
Nebius
sujee@nebius.com

Agenda

Introduction

Infrastructure for inference

Model-shaping optimizations

Model-serving optimizations

Nebius at a glance

Full-stack AI cloud infrastructure, purpose-built for the global AI industry.

NASDAQ: NBIS

Publicly listed on Nasdaq.
HQ in Amsterdam.

\$2B

Strategic investment from NVIDIA
for a full-stack AI cloud partnership.

5+ GW

NVIDIA systems target
by the end of 2030.



Owned data
centers across the
EU and US —
bare-metal, not
shared cloud.

Early adopter of
NVIDIA Rubin, Vera
CPUs and BlueField
storage — customers
get Day 1 access to
the latest chips.

First European
cloud to run both
NVIDIA Blackwell Ultra
platforms (HGX B300
and GB300 NVL72)
in production.



~\$17.4B Microsoft
deal + \$27B Meta
capacity agreement —
trusted by the largest
AI builders globally.

Vertical integration
from silicon to
Serving. One vendor
from GPU training
clusters to optimized
inference tokens.



Nebius Token Factory

Build AI engineered for your product.

The complete AI engineering platform to run open-source models in production, with full control over performance, cost and behavior.

The problem you face today

Neither option scales cleanly

Today, teams choose between:

OpenAI, Anthropic

Closed APIs

- Fast to start, but zero customization.
- Costs scale linearly, but margins compress.
- Opaque performance, low rate limits.
- Can't tune for your workload or data.

Self-hosting

- Full control, but massive infrastructure burden.
- Requires a dedicated inference R&D team.
- Months to production, then constant maintenance.
- GPU procurement cycles slow you down.

The Token Factory shift

Dedicated open inference you can operate.
Managed by us, controlled by you over:



Performance

p99 latency, TTFT, scaling behavior. Dedicated GPUs, predictable throughput. Sub-200ms latency.



Cost

Token efficiency, KV cache optimization, speculative decoding, distillation. Up to 50% cost reduction.



Behavior

Managed inference: We proactively optimize your model for your workload, chips and traffic patterns. Synthetic benchmarks during onboarding. Continuous fleet tuning.

Full-stack GenAI platform

Inference

- 60+ models • Dedicated endpoints • Fast and base flavors
- Batch API • Structured output • Function calling

Data Lab

- Inference log import • SQL-like dataset filtering
- Dataset versioning • One-click export • Batch evaluation

Post-training

- LoRA and full fine-tuning • Model distillation • Custom spec. decoding training
- Quantization and calibration • Preference optimization (RL/GRPO)

Deployment

- Custom Weights Hub • GPU type selection • Region control
- Autoscaling config • Observability surfaces • FT dashboard

Security and compliance

Reliability and scale

Model ops and innovation

What we handle so you don't have to

We don't just host models. We help you engineer AI systems.

Optimization layer

Cache-aware routing → 90% KV cache hit rates in multi-turn workloads

Prefill/decode disaggregation → optimized TTFT for long-context

KV cache offloading → up to 50% cost savings on chat workloads

Speculative decoding → ~30% performance boost (higher with custom data)

Custom forks on TensorRT-LLM / vLLM / SGLang — engine selection per workload

Workload-aware autoscaling with real-time fleet metrics

We optimize latency, throughput and cost at the engine level.

AI cloud infrastructure

Dedicated NVIDIA H100 / H200 / B200 / B300 clusters

Multi-region inference (EU and US-owned data centers)

99.9% SLA with proactive system health monitoring

800 Gbps InfiniBand (Quantum-X800) interconnect

Bare-metal performance, not shared cloud instances

You control behavior. We run the infrastructure.

Enterprise foundation

SOC 2 Type II + HIPAA + ISO 27001 certified

RBAC + SSO for team access control

Zero data retention (data never trains our models)

Project separation and unified enterprise billing

OpenAI-compatible API for seamless migration

Production-ready from day one.

Continuous model improvement loop

Start with the right model and run it at production scale.

Choose and run models

60+ open-source LLMs
GPT OSS, DeepSeek, Kimi, Qwen, Mistral, Llama, GLM...
Batch and real-time inference
Structured outputs and function calling
Dedicated endpoints

Inference

Turn production traffic into training — minutes from logs to training.

Observe and capture

Auto-capture inference logs
SQL-like filtering and cleaning
Version datasets
One-click export to training

Data Lab

Optimize quality, cost and latency inside the same platform.

Improve the model

LoRA and full fine-tuning
Model distillation
Custom speculative decoding
Quantization + calibration
90% quality at lower cost

Post-training

Design AI behavior in production, not just the model.

Deploy as a system

Custom Weights Hub
GPU type and region selection
Min/max replica autoscaling
Real-time observability
Zero-retention option

Dedicated endpoints

Continuous improvement loop — every cycle makes your AI more specific, faster and cheaper

Scaling open models to production

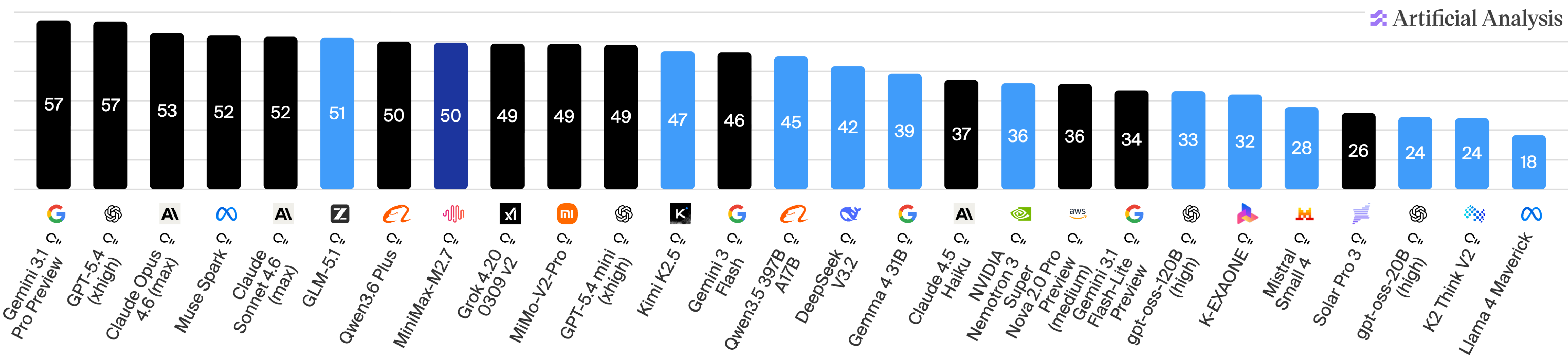
**Great inference starts with
great models + infrastructure**

Nebius Token Factory has SOTA open models!

Artificial Analysis Intelligence Index by Open Weights / Proprietary

Artificial Analysis Intelligence Index v4.0 incorporates 10 evaluations: GDPval-AA, τ^2 -Bench Telecom, Terminal-Bench Hard, SciCode, AA-LCR, AA-Omniscience, IFBench, Humanity's Last Exam, GPQA Diamond, CritPt

■ Proprietary ■ Open Weights ■ Open Weights (Commercial Use Restricted)



OS models on Token Factory

Large models

Great intelligence

Makes great “teacher” models

Model	Parameters
GLM 5.2	735B
Kimi 2.6	1.1T
DeepSeek V4 Pro	862B
Qwen3.5-397B	400B

Smaller models

Nimble, can be great for specific tasks

Good “student” models

Model	Parameters
Qwen-3-30B	30B
Nemotron Nano	30B

Infrastructure + optimizations

Inference stack

Model shaping

Fine-tuning

Model distillation

Quantization
INT8, FP8, FP16

Model serving

Speculative
decoding

Prefill /
decoding

Caching
Prompt cache,
embedding reuse

Parallelism
Tensor, pipeline,
model

Serving engine
vLLM, SGLang,
TensorRT-LLM

Infrastructure

NVIDIA GPUs

Fleet management

Auto-scale

Routing

Infrastructure: Hardware

Hardware optimizations

Latest NVIDIA Accelerated
Computing

**HGX B200 and
GB200 NVL72
systems**
are deployed

Continuous
benchmarking of

**hardware
and kernels**

and optimizing
runtimes and kernel

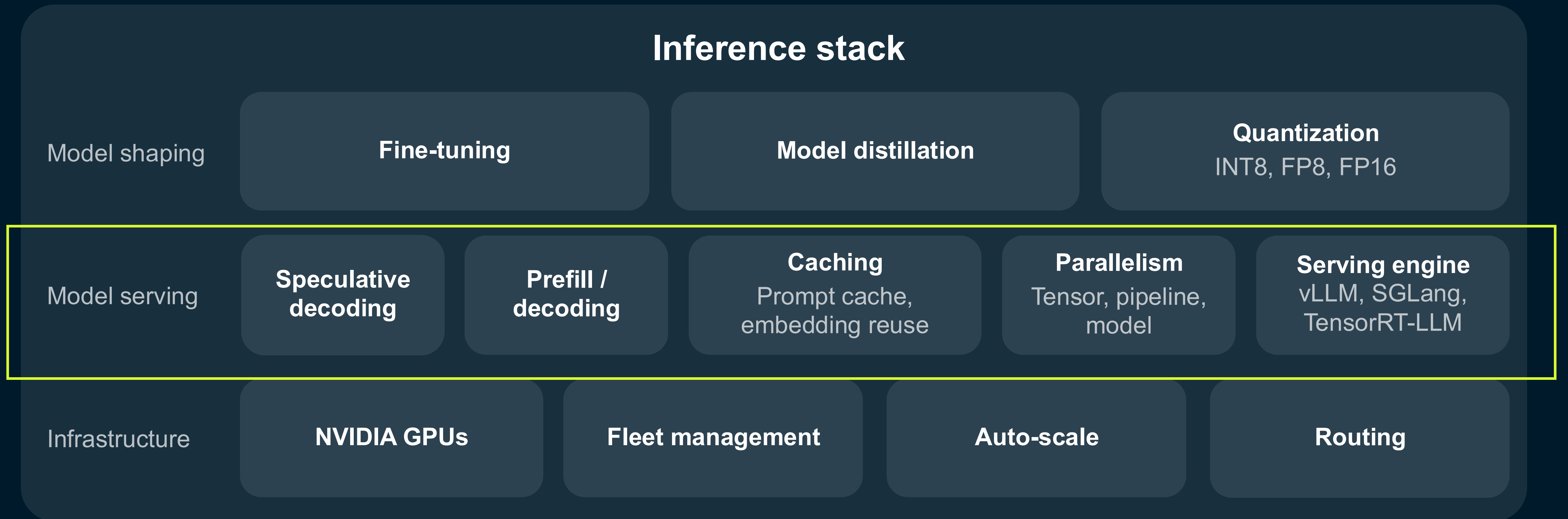
We use advanced low-precision
formats such as

NVFP4

to take advantage of the
NVIDIA Blackwell platform

Model serving

Model serving



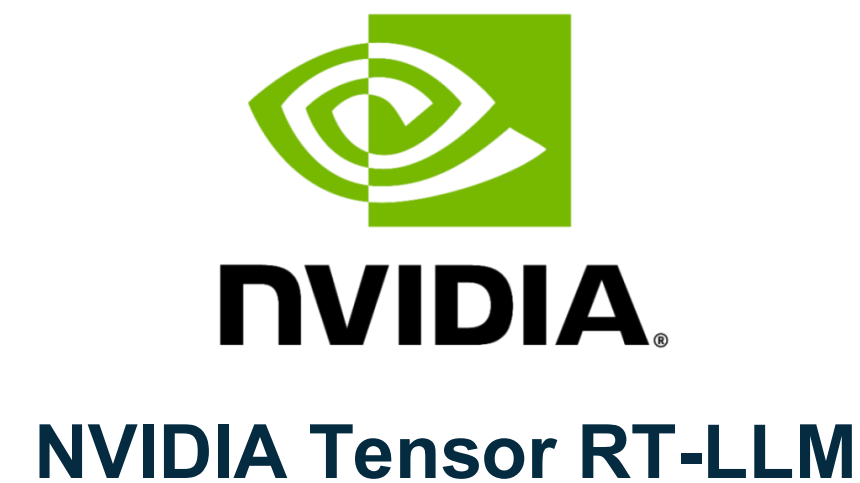
Inference engines

Customers provide performance and quality requirements, and we determine which engine and configuration best fits those constraints.

We maintain internal forks to improve observability and enable additional performance optimizations.

Contribute changes back to open source, when they are not platform- or customer-specific.

Popular engines



Fleet management

Load balancing and auto scaling

Load balancing

Load balancing LLM inference is much more complex than 'web traffic.'

Challenges:

- Traffic patterns may vary significantly over time.
- Inference requests are often highly heterogeneous, with orders-of-magnitude differences in input size, output length and compute cost.
- Simple round-robin routing doesn't work.

see next slide

Inference workloads example

Input (small)

Write a haiku about cats

Input (small)

Write a report on electric cars of 2026

Input (large)

Coding agent gives a large context ... and asks
"summarize this project"

Input (large)

Coding agent gives a large code base... and asks
"refactor this code base"

Output (small)

Whiskers catch the sun ...

Output (large)

Large report....

Output (small / medium)

Project summary

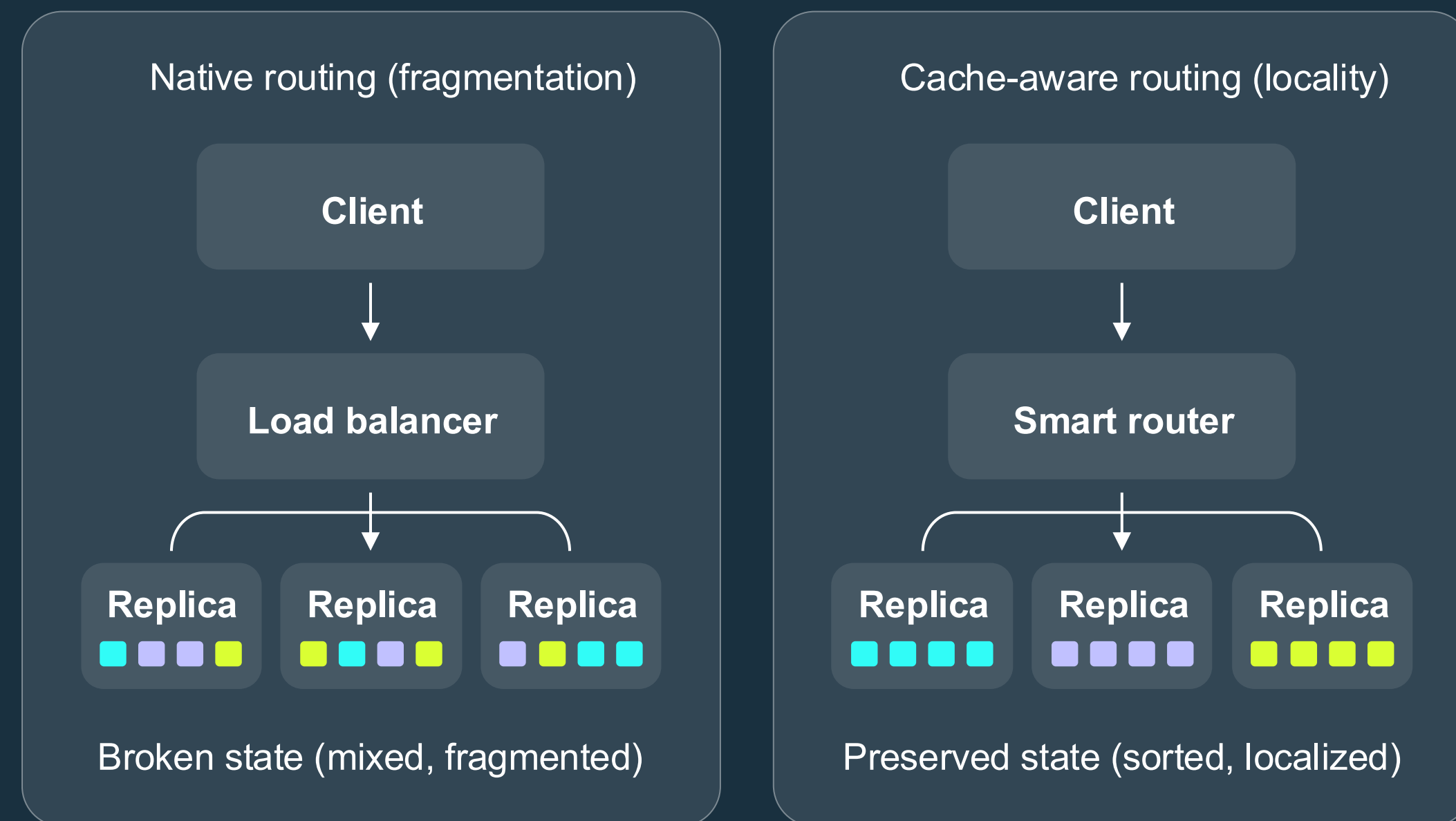
Output (large)

Refactored code

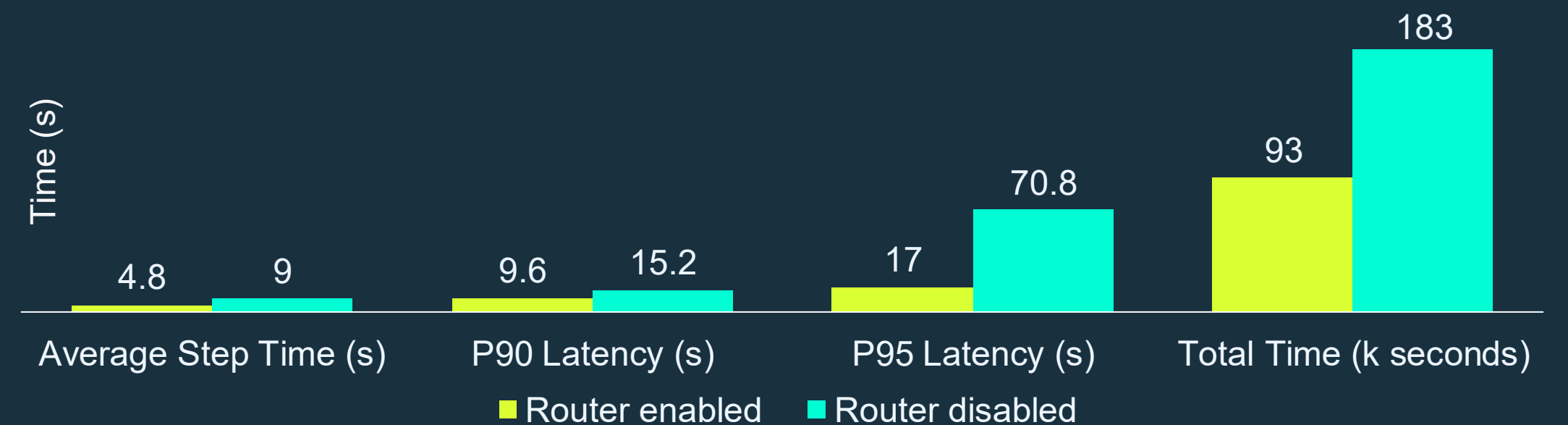
Cache-aware routing

For each request, the router attempts to select a replica with the best KV cache match for the request.

Context fragmentation vs. context locality



Inference latency and runtime comparison: Router-enabled vs. disabled



With cache-aware routing

95%

Percentage of tokens served from cache

Without cache-aware routing

65%

Tokens served from cache

Optimization: Speculative decoding

Latency problem

The challenge

- Many benchmarks focus on tokens per minute and requests per minute.
- But what users experience is end-to-end latency.

Root cause

Generating tokens in large models (MoE) is time-consuming.

Speculative decoding

Large models
generating tokens
in **sequence**
is expensive!

Traditional autoregressive decoding



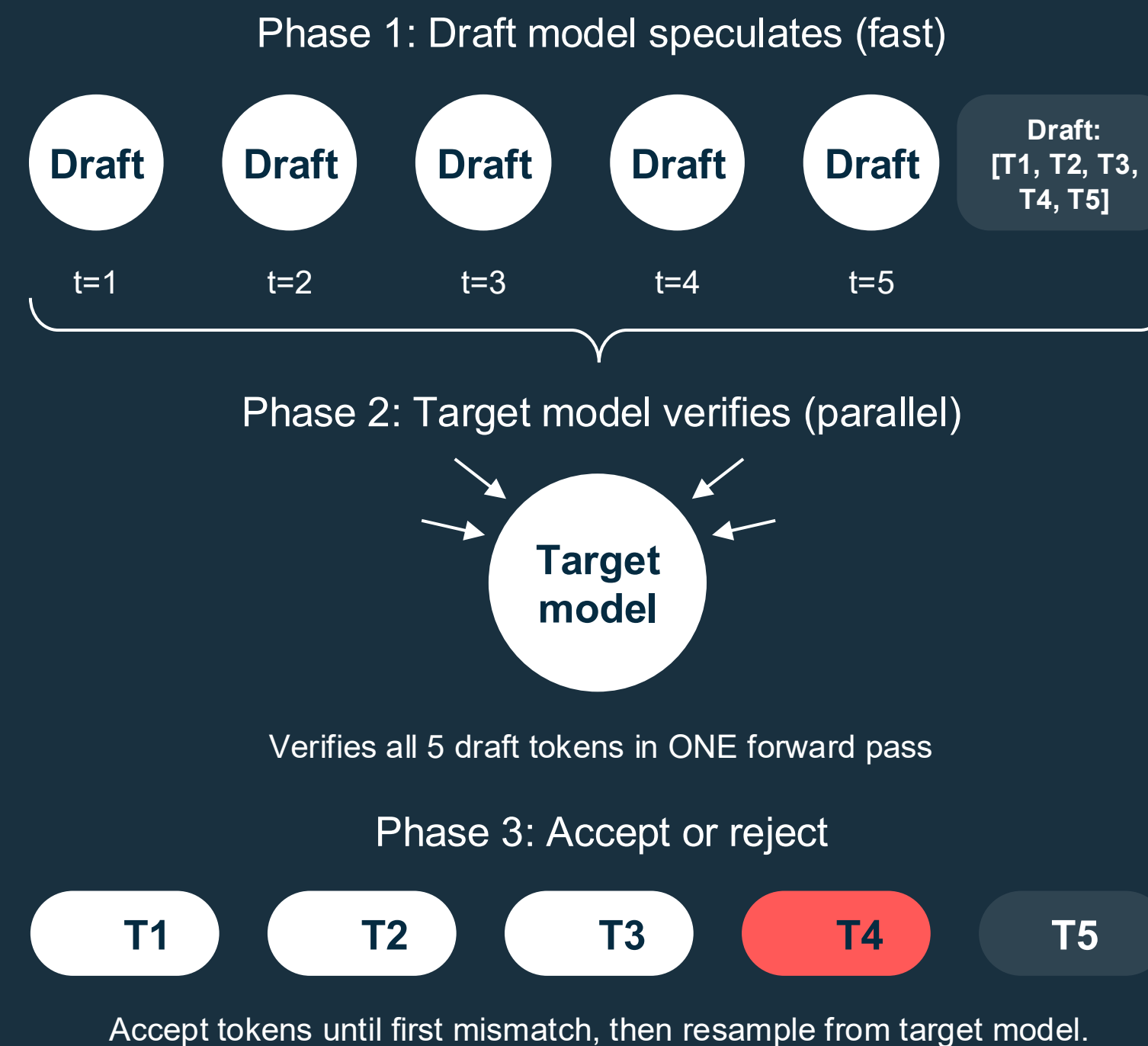
Problem: Each token requires a full forward pass through the large model
5 tokens = 5 sequential forward passes (SLOW)

Speculative decoding

In speculative decoding

- A smaller model generates tokens (in batches) – faster than a large model.
- Larger model verifies generated tokens and accepts / rejects – faster!

Speculative recording process



Performance comparison

Traditional decoding



Speculative decoding

Draft 5 tokens
($5 \times T_{\text{small}}$)

Verify all 5 tokens in parallel
($1 \times T_{\text{large}}$)

Total time: ($5 \times T_{\text{small}}$) + ($1 \times T_{\text{large}}$)

Speculative decoding at Token Factory

We **train draft models by using a blended dataset**

built from batched inference inputs, and we can incorporate customer-specific data.

Using a generic dataset, speculative decoding can deliver up to a **30% performance improvement**, while **incorporating customer data often yields significantly higher gains**.

After dataset preparation, the speculator model is trained and integrated into the inference pipeline.

Optimization:
KV cache

The problem: Redundant computation

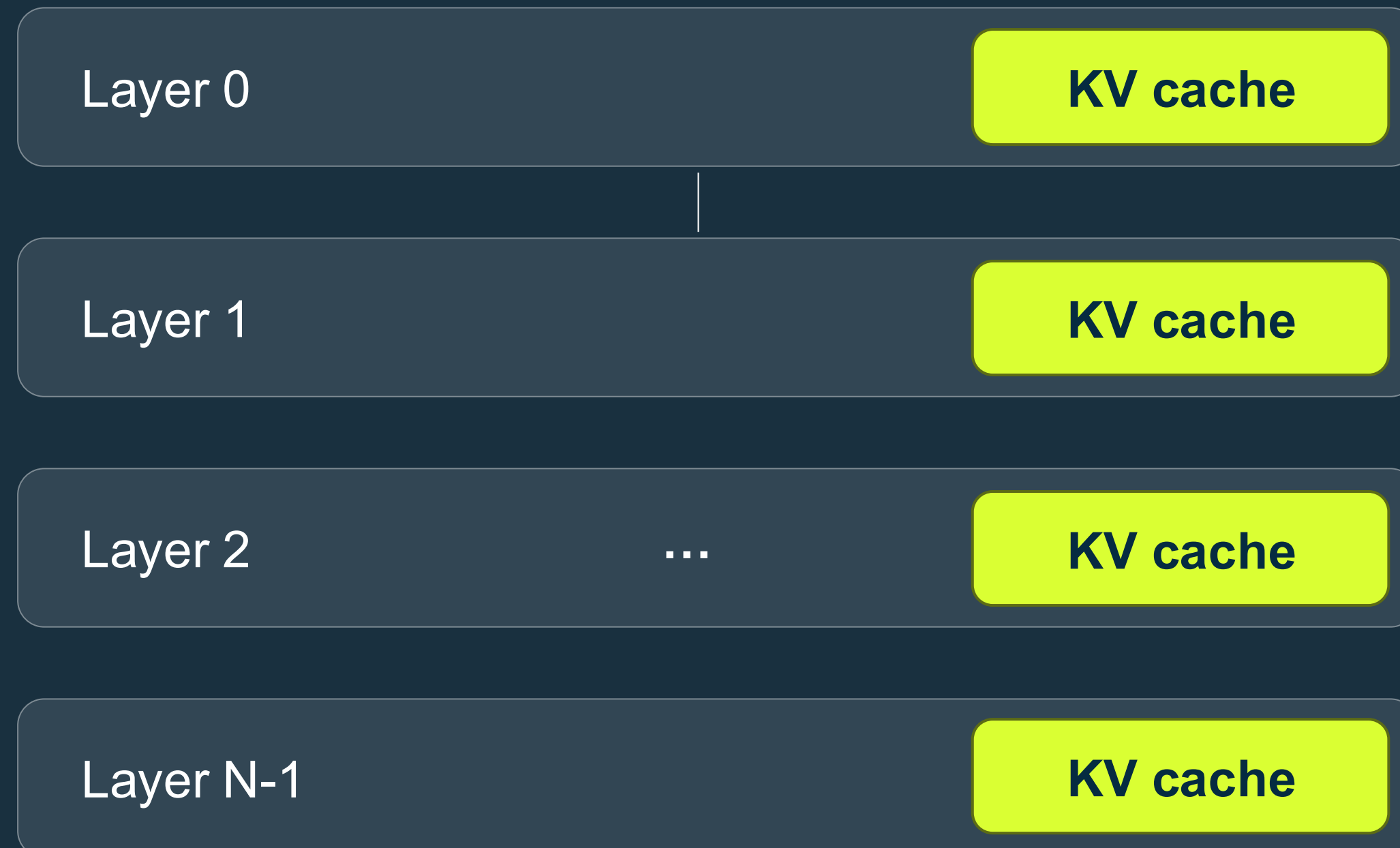


Every new token requires recomputing attention to all previous tokens.

For example:

- To generate token-1 : compute K,V for 1 token (1)
- To generate token-2: compute K,V for 2 tokens (2)
- To generate token-10: compute K,V for 10 tokens (10)

Solution: KV cache



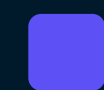
Insight: Key (K) and Value (V) matrices for previous tokens never change!

- Compute K and V once per token
- Store them in cache
- Reuse cached values for all future tokens

Each transformer layer maintains their own KV cache.

Each layer learns different representation.

Understanding KV cache

 Input token  New token  Cached K,V

Step 0: Initial prompt processing

Layer 0. Transformer block

Query (Q)

Key (K)

Value (V)

The

cat

sat

The

cat

sat

Layer 1. Transformer block

Query (Q)

Key (K)

Value (V)

The

cat

sat

The

cat

sat

Step 1: First generation — Computing **on**

Layer 0. Transformer block

Query (Q)

Key (K)

Value (V)

on

The

cat

sat

The

cat

sat

Layer 1. Transformer block

Query (Q)

Key (K)

Value (V)

on

The

cat

sat

The

cat

sat

KV cache



Input token



New token



Cached K,V

Step 2: Caching K,V for 'on'

Layer 0. Transformer block

Query (Q)

Key (K)

Value (V)

The cat sat
on

The cat sat
on

Layer 1. Transformer block

Query (Q)

Key (K)

Value (V)

The cat sat
on

The cat sat
on

Step 3: First generation — Computing **the**

Layer 0. Transformer block

Query (Q)

Key (K)

Value (V)

the

The cat sat
on

The cat sat
on

Layer 1. Transformer block

Query (Q)

Key (K)

Value (V)

the

The cat sat
on

The cat sat
on

KV cache



Input token



New token



Cached K,V

Step 6: Final state

Layer 0. Transformer block

Query (Q)

Key (K)

Value (V)

The	cat	sat
on	the	mat

The	cat	sat
on	the	mat

Layer 1. Transformer block

Query (Q)

Key (K)

Value (V)

The	cat	sat
on	the	mat

The	cat	sat
on	the	mat

KV cache

Speed

5–10x

Faster generation compared to recomputing attention for all tokens.

Efficiency

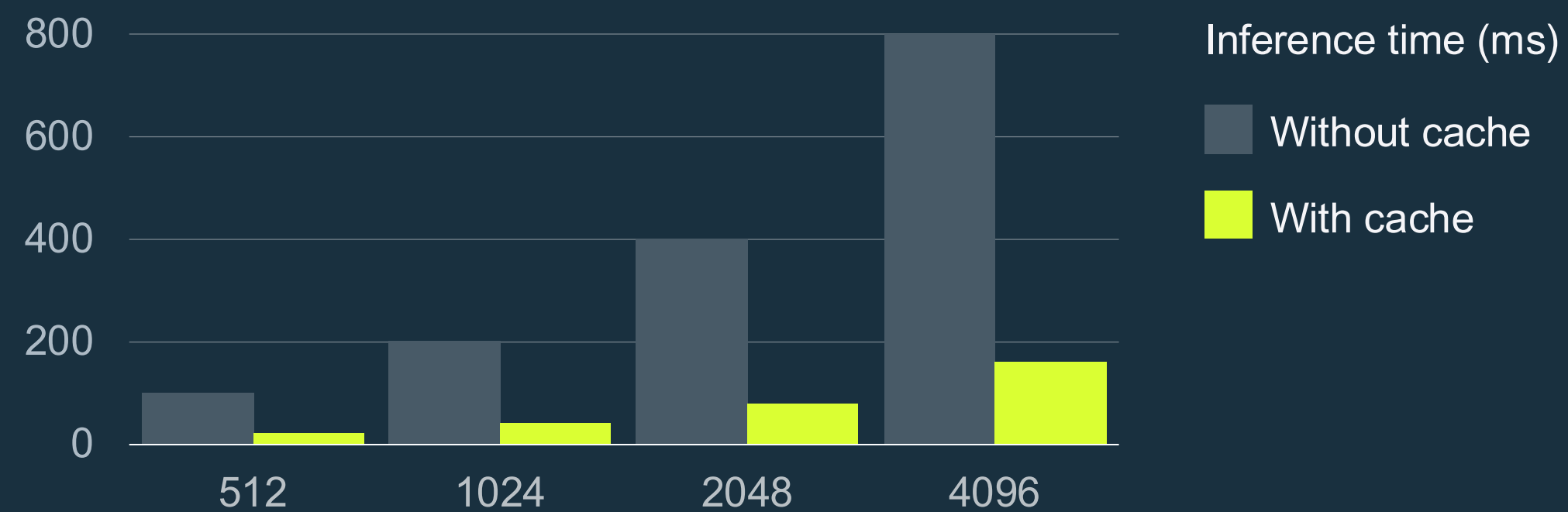
$O(n)$

Linear complexity per token instead of $O(n^2)$ without caching.

Throughput

2–3x

Higher tokens/second enabling better user experience.



Memory requirements

Cache grows linearly with sequence length.

e.g., Llama-2-7B

- 32 layers x 32 heads, head_dimension = 128
- 2048 tokens = ~ 1 GB cache
- 8192 tokens = ~ 4 GB cache

Long contexts can get expensive!

KV cache is the economic engine

In chat, most tokens are repeats from previous turns.

Cache hit rate:

~5%



30%+



50–60% after tuning

KV cache at Token Factory

**KV cache is very effective,
but memory intensive!**

GPU memory is precious — used to store model weights and buffers.

→ KV cache can get evicted due to tight memory!

**At Nebius Token Factory, we support
off-loading KV cache to CPU memory
or disk.**

Our cache can be larger (not constrained by limited GPU memory), allowing effective KV cache hit rates to approach their theoretical limits.

That led to up to 50% cost optimization.

Optimization: Prefill decoding

Disaggregated prefill

Problem

- In long sequences, prefill (populating data) needs more **compute time**.
- Decoding requires more **memory bandwidth**.
- And they compete for resources.

Input prompt

The quick brown fox jumps ...

Prefill phase

Process ALL input tokens in parallel
COMPUTE-BOUND High GPU utilization

KV
cache

Decode phase

Generate output tokens one-by-one
MEMORY-BOUND Low GPU utilization

Parallelism high | Duration short
FLOPs/token high | Bottleneck compute
Ideal hardware high-core GPU

Output tokens

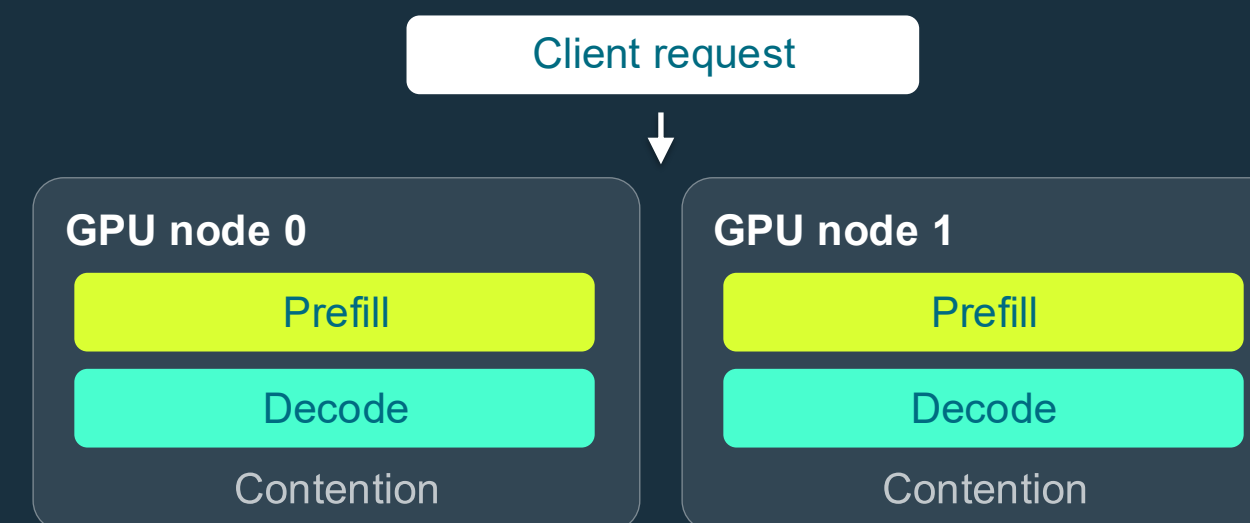
leaps over the ...

Disaggregated prefill

Solution

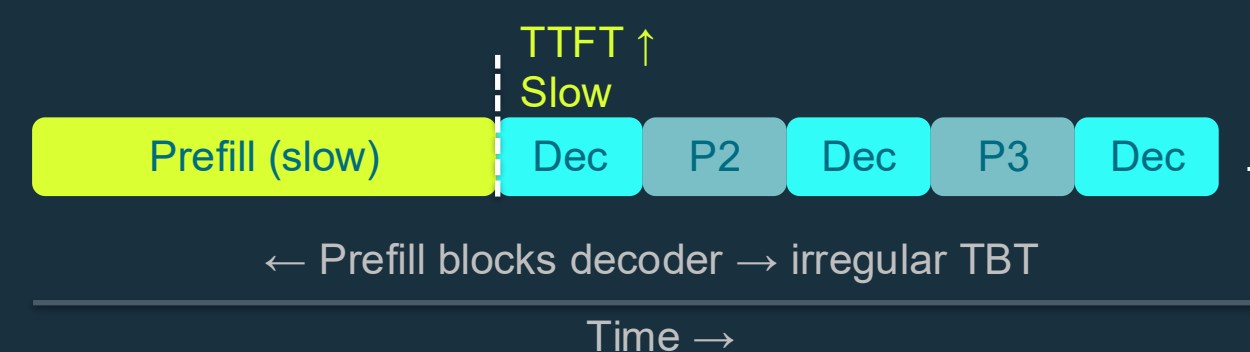
- Separate these prefills.
- ONE GPU pool for prefill (lots of compute, like an H100).
- Another GPU pool for decode (needs good memory bandwidth for KV cache).

Coupled (traditional)

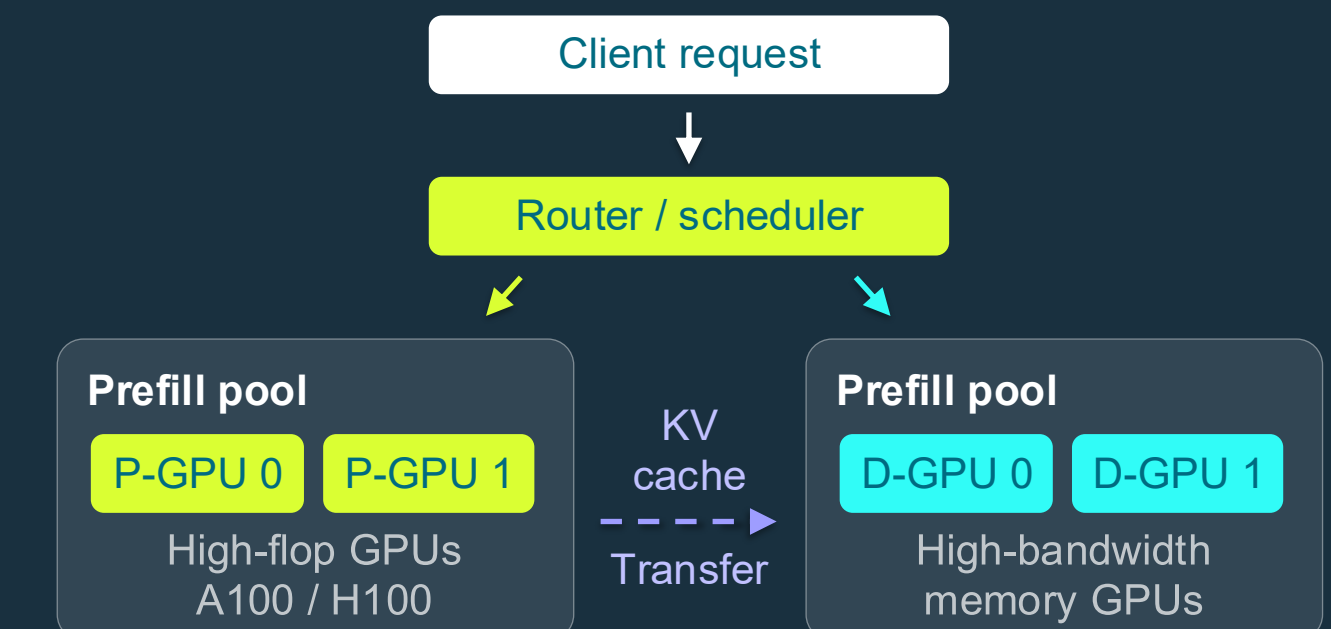


Problems

- TTFT spikes when decode starves prefill
- GPU under-utilized (mixed workloads)
- Can't scale prefill/decode independently

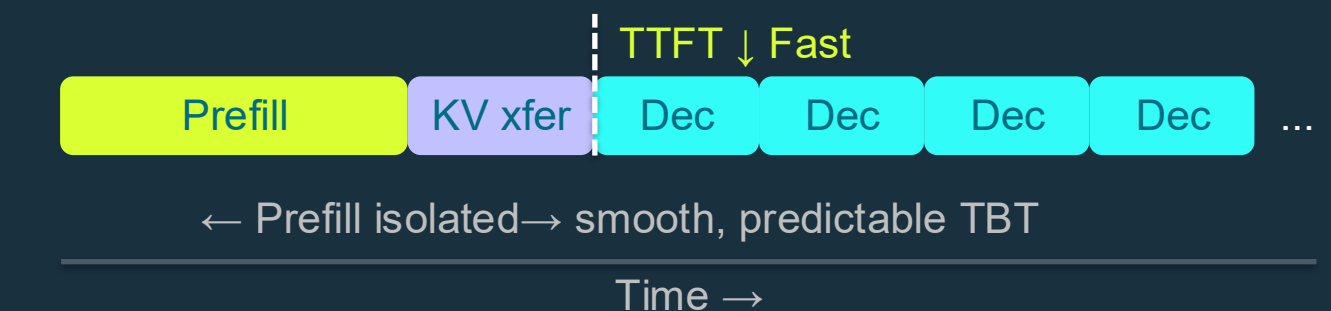


Disaggregated



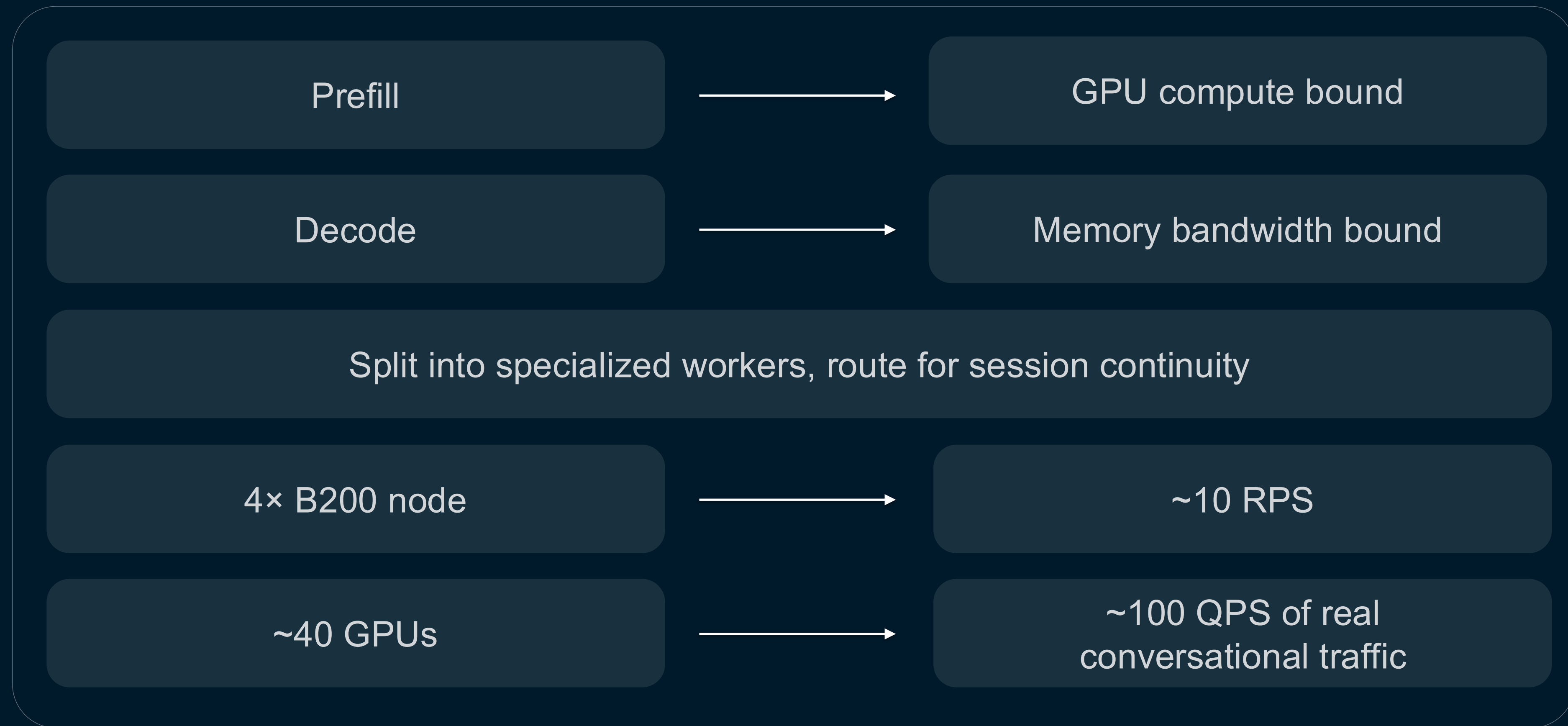
Benefits

- Prefill and decode scale independently
- No TTFT/TBT interference
- Right hardware for each phase



■ Prefill ■ Decode ■ KV transfer

Disaggregate prefill and decode



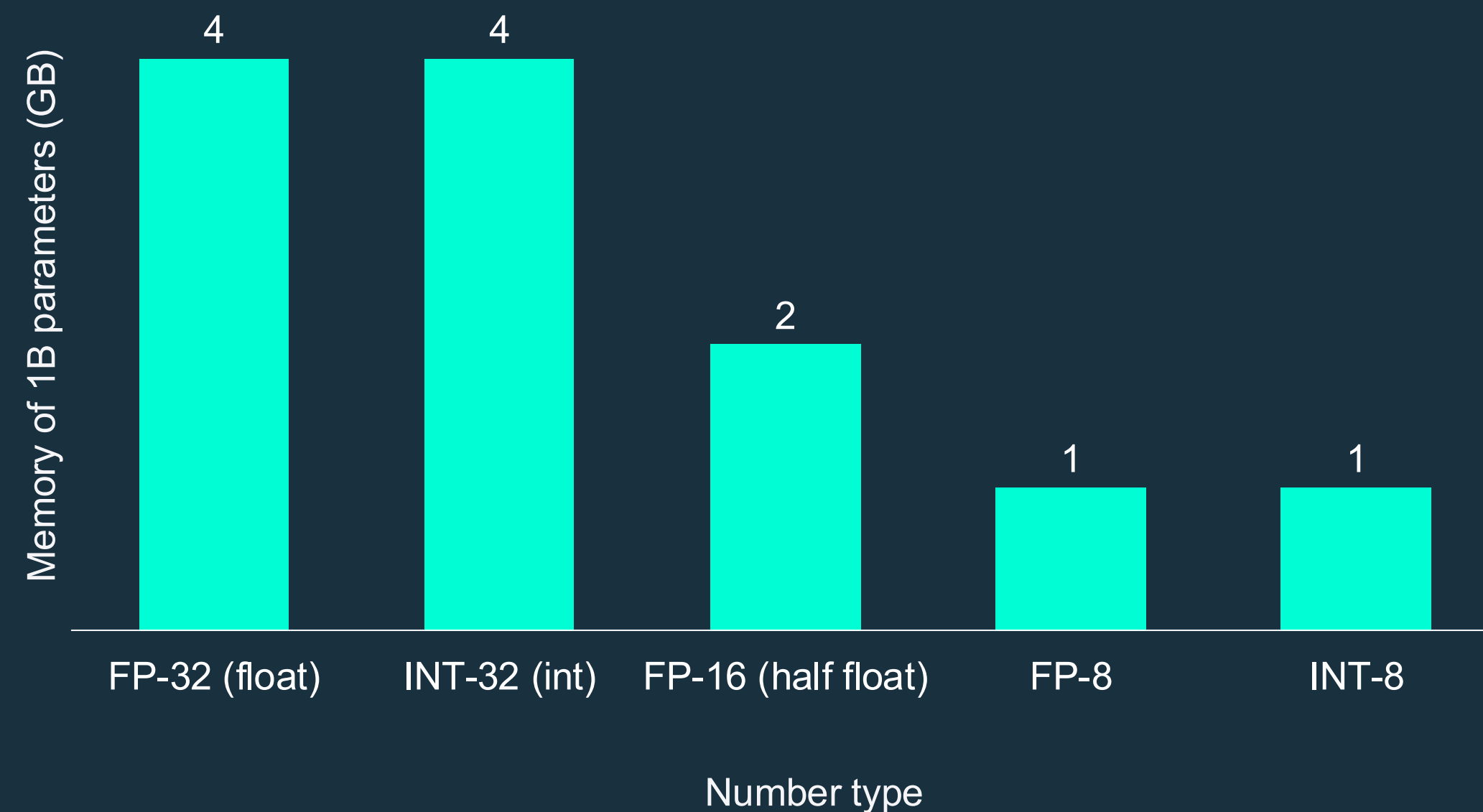
Optimization: Quantization

Quantization reduces precision of model weights

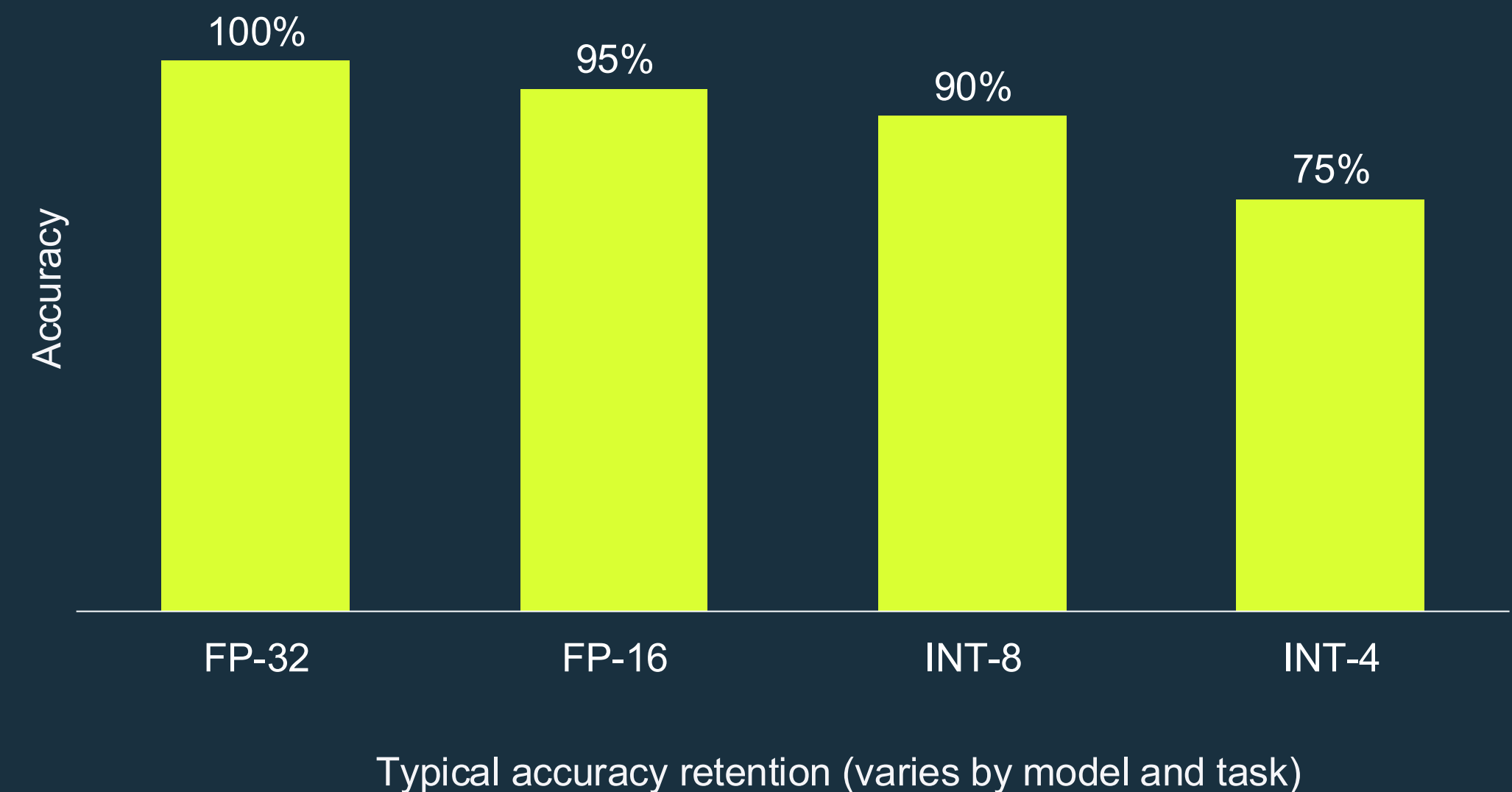
Example: FP-32 (32 bits) → INT-8 (8 bits)

This can dramatically reduce model size and speed up inference. However, the quality of the model suffers.

Memory for 1B params (GB) vs. number type



Quantization and model accuracy



Quantization at Token Factory

KV cache quantization is particularly effective for long-context and high-concurrency workloads, as it reduces cache memory usage and enables higher parallelism.

While quantization can improve performance, it can degrade model quality.

At Token Factory, we validate model quality against public benchmarks (AIME, GSM8k) and customer-specific benchmarks.

We make sure that the quality sacrifice is within acceptable model parameters.

What we covered today

Inference stack

Model shaping

Fine-tuning

Model distillation

Quantization
INT8, FP8, FP16

Model serving

**Speculative
decoding**

**Prefill /
decoding**

Caching
Prompt cache,
embedding reuse

Parallelism
Tensor, pipeline,
model

Serving engine
vLLM, SGLang,
TensorRT-LLM

Infrastructure

NVIDIA GPUs

Fleet management

Auto-scale

Routing

Getting started

Token Factory

Tokenfactory.nebius.com

Token Factory cookbook

github.com/nebius/token-factory-cookbook

Nebius Discord (support)

discord.gg/AuQWb3nhM9

Thanks + Q&A



Dylan Bristot
Product Marketing Manager
Nebius
dylan@nebius.com



Sujee Maniyam
AI Developer Advocate
Nebius
sujee@nebius.com