INTRO TO INFERENCE

How to Run AI Models on a GPU

A Google Cloud x NVIDIA Learning Path

Dr. Nova, I've been using ChatGPT and LLMs all day... but I have no idea how they work behind the scenes.

Great question, Kai! Every AI model has two phases: Training and Inference.

Training is when the model learns from data — adjusting its weights through millions of examples.

TRAINING

Think of training as studying for an exam...

...and inference is taking the exam. The model applies what it learned.

INFERENCE

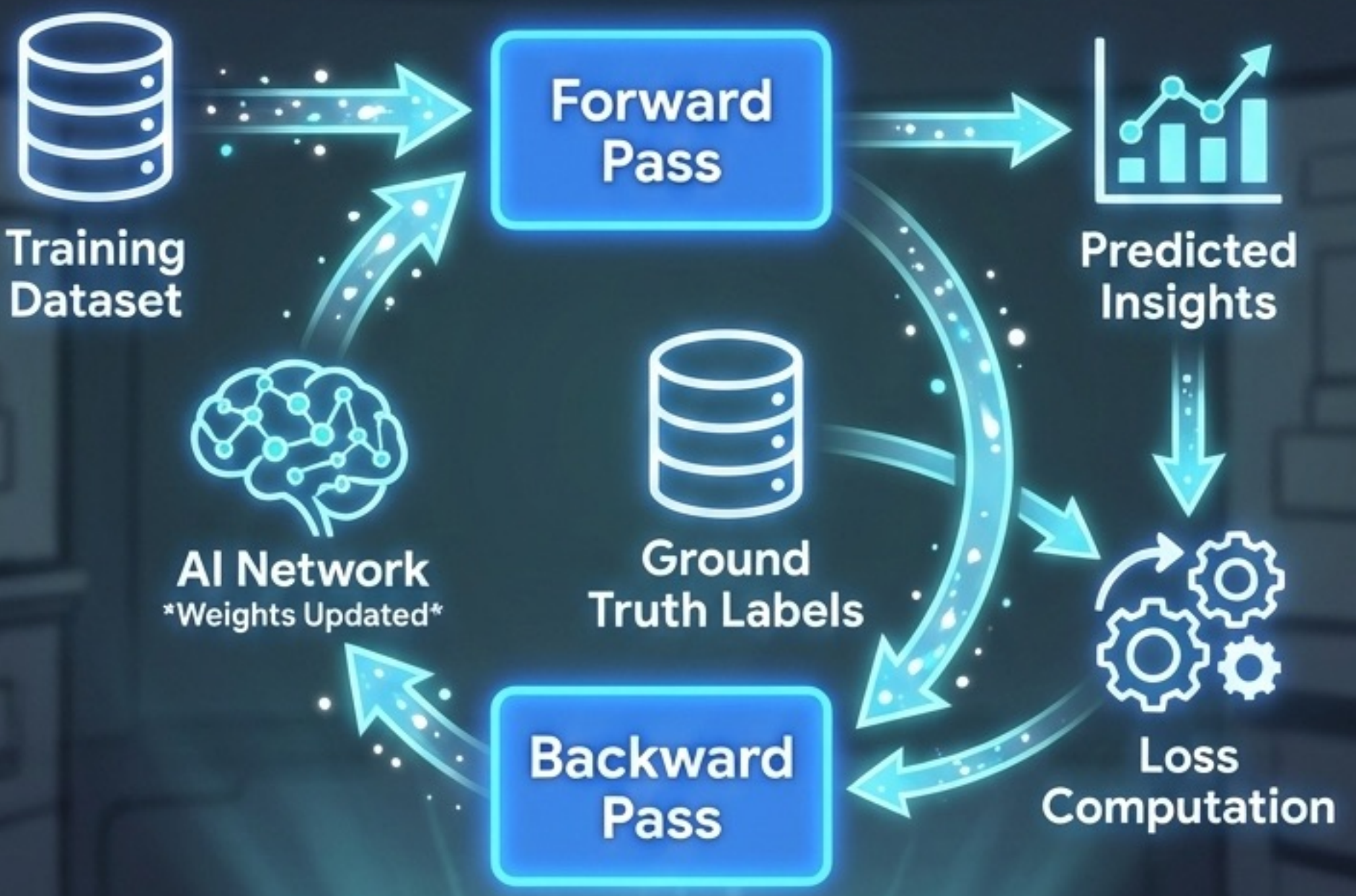
PREDICTION: "This is a cat"

OBJECT: VEHICLE (99%)

So inference is the part users actually interact with!

Training has two key steps: a Forward Pass and a Backward Pass.

The forward pass makes predictions. The backward pass compares them to the truth, computes the error, and adjusts the weights.



So the model keeps going around this loop, getting better each time?

Exactly! But once training is done, inference is much simpler...

Inference is just the forward pass. No backward pass, no weight updates. The model's knowledge is frozen.

Training Dataset

Forward Pass

Predicted Insights

AI Network
Weights Frozen

But if millions of people are using an LLM at once, how does it handle all that?

Those are the three big questions of serving AI in production. And that's where GPUs come in!

How FAST can we respond?

How MANY users at once?

How MUCH compute?

GPU

GPUs are hardware accelerators designed for massive mathematical operations.

I'm built for parallel processing! Thousands of cores working simultaneously!

CPUs process data sequentially — one operation at a time.

One... at... a... time...

GPUs with thousands of parallel cores multiply and add tensors at scale.

I handle thousands of matrix multiplications in parallel!

Without GPU acceleration, modern LLMs wouldn't be practical. But raw power isn't enough — we need a structured pipeline.

A pipeline?

The inference pipeline is a structured sequence that turns raw input into useful output.

Google Cloud

1. PREPROCESSING

How does AI work?

AI

Tokenization: breaking text into tokens the model understands.

2. BATCHING & PADDING

Multiple requests are grouped into rectangular tensors for efficient GPU processing.

3. MODEL FORWARD PASS

The trained network computes predictions from the batched inputs.

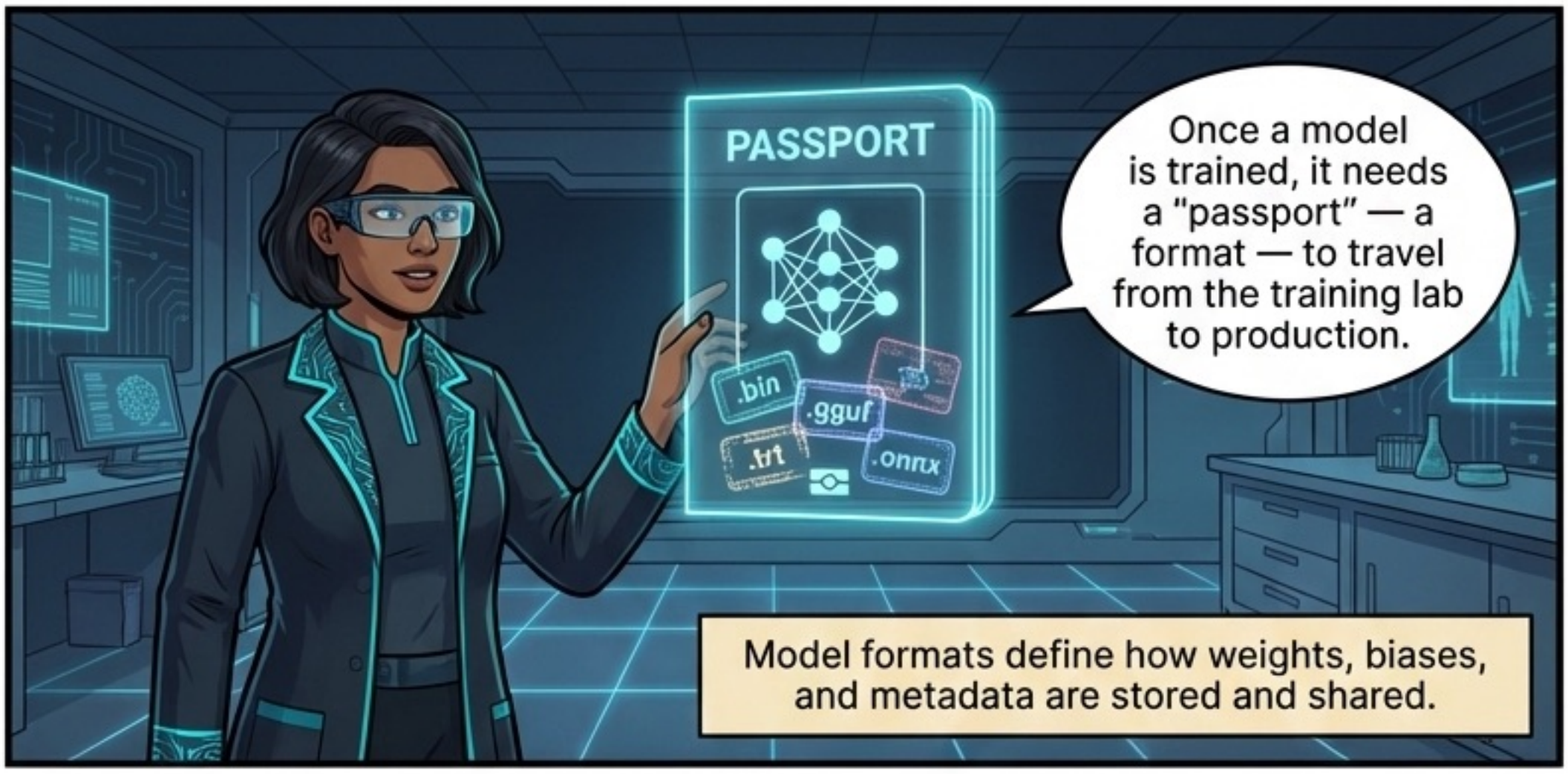
4. DECODING

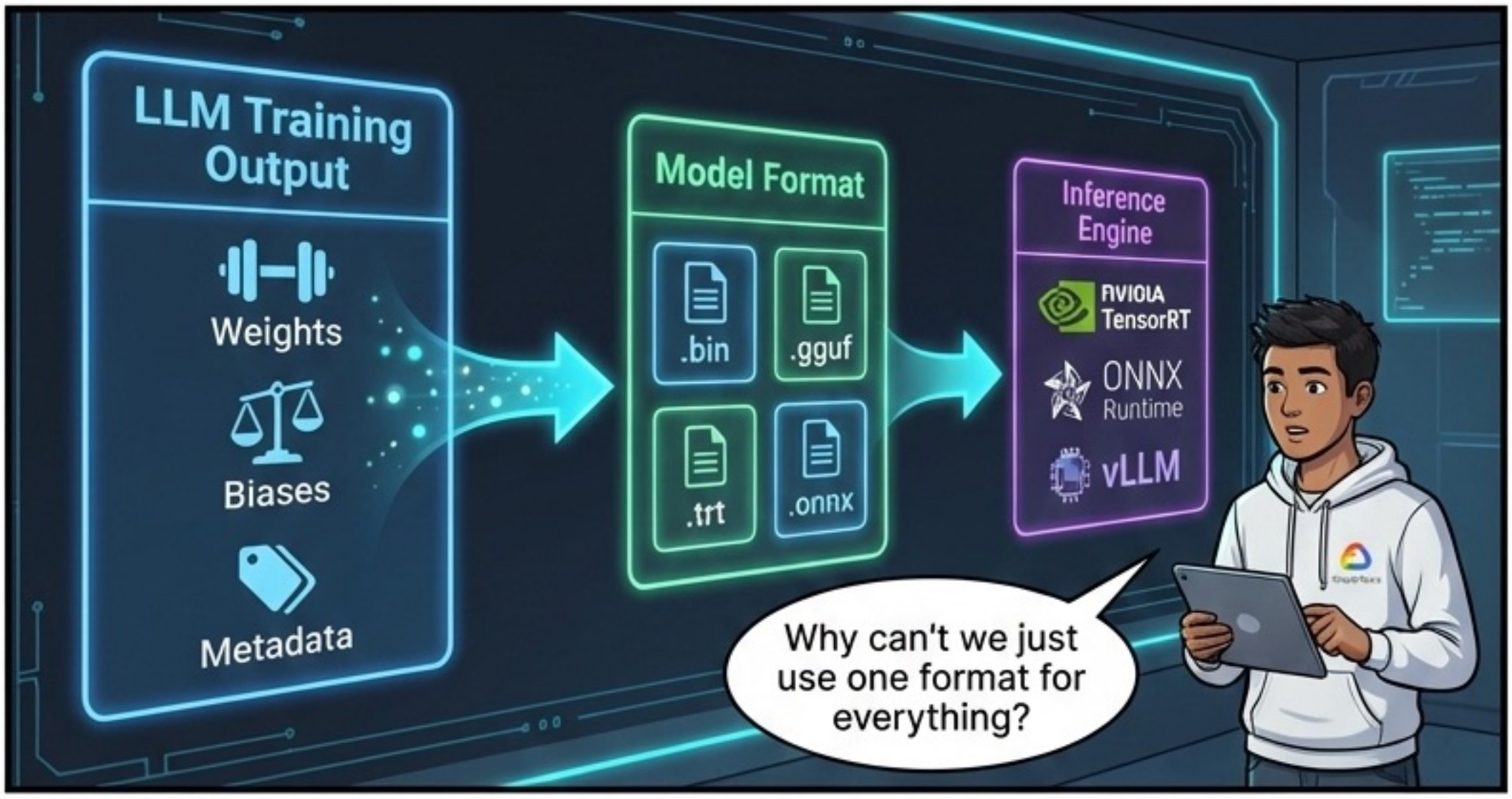
5. POSTPROCESSING

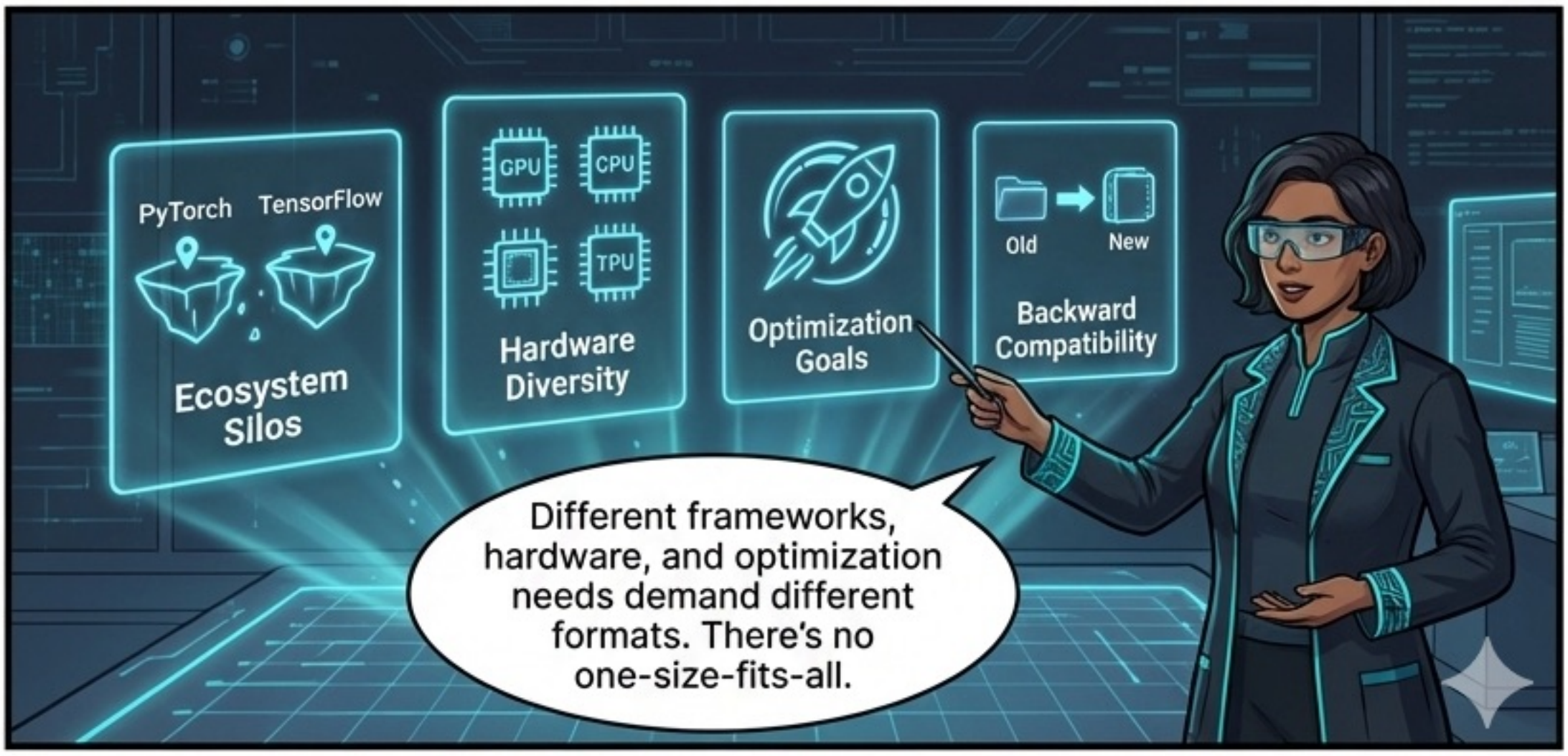
A complex neural network...

6. SERVING

So the pipeline takes my question and delivers an answer through all these steps!


PASSPORT
.bin
.gguf
.trt
.onnx
Once a model is trained, it needs a "passport" — a format — to travel from the training lab to production.
Model formats define how weights, biases, and metadata are stored and shared.


LLM Training Output
Weights
Biases
Metadata
Model Format
.bin
.gguf
.trt
.onnx
Inference Engine
NVIDIA TensorRT
ONNX Runtime
vLLM
Why can't we just use one format for everything?


PyTorch
TensorFlow
Ecosystem Silos
GPU
CPU
TPU
Hardware Diversity
Optimization Goals
Old
New
Backward Compatibility
Different frameworks, hardware, and optimization needs demand different formats. There's no one-size-fits-all.

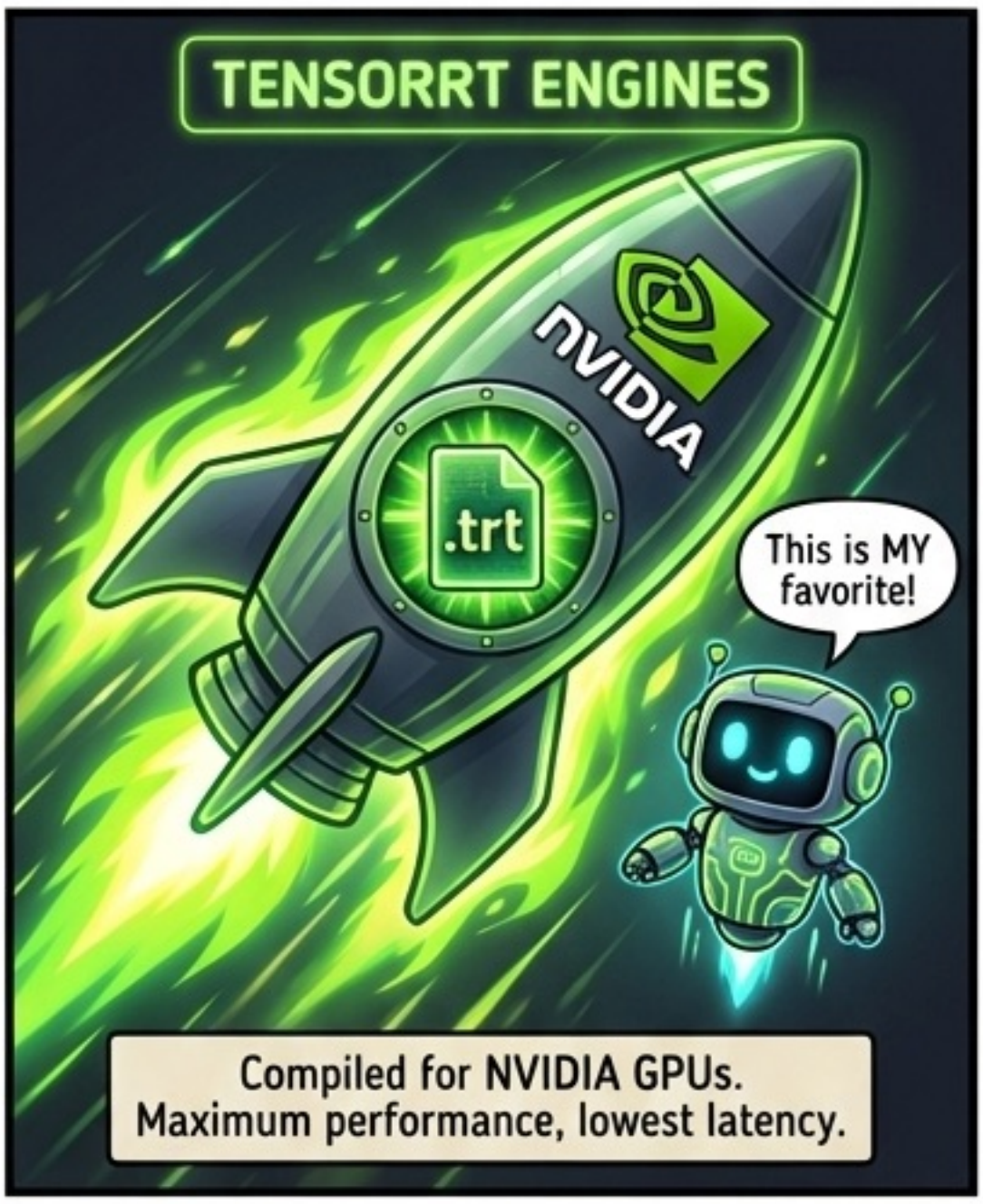
SAFETENSORS

Fast, secure. The default for sharing on Hugging Face. Prevents code execution on load.

GGUF

Compact, quantized. Powers 'LLMs on your laptop' with llama.cpp.

TENSORRT ENGINES

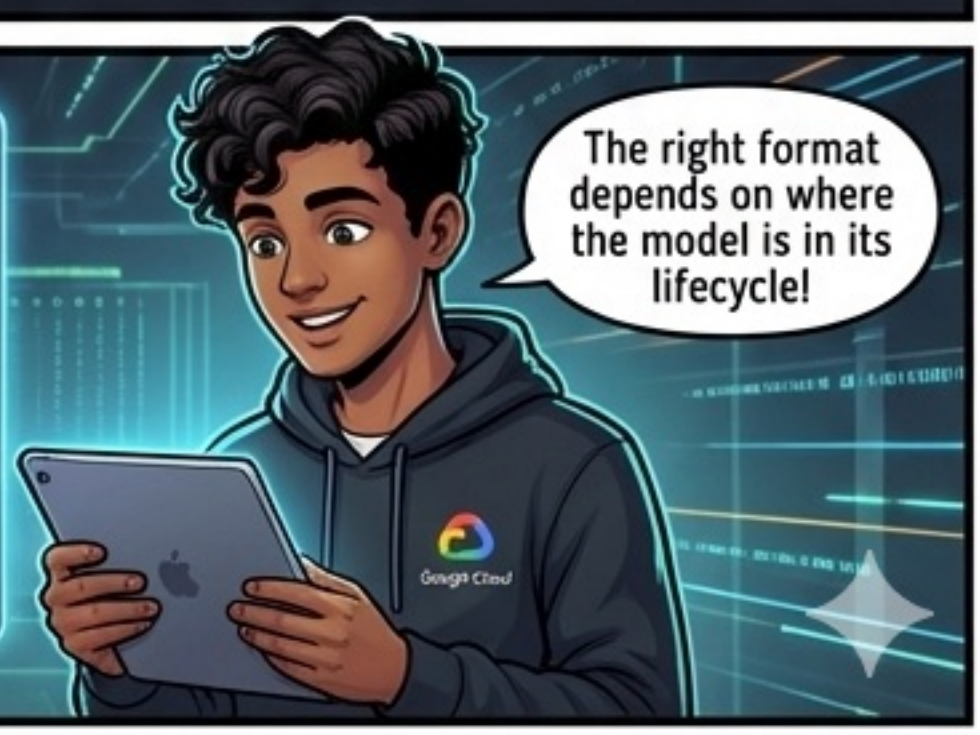


Compiled for NVIDIA GPUs.
Maximum performance, lowest latency.

ONNX

The universal bridge.
Best for mixed-hardware environments.

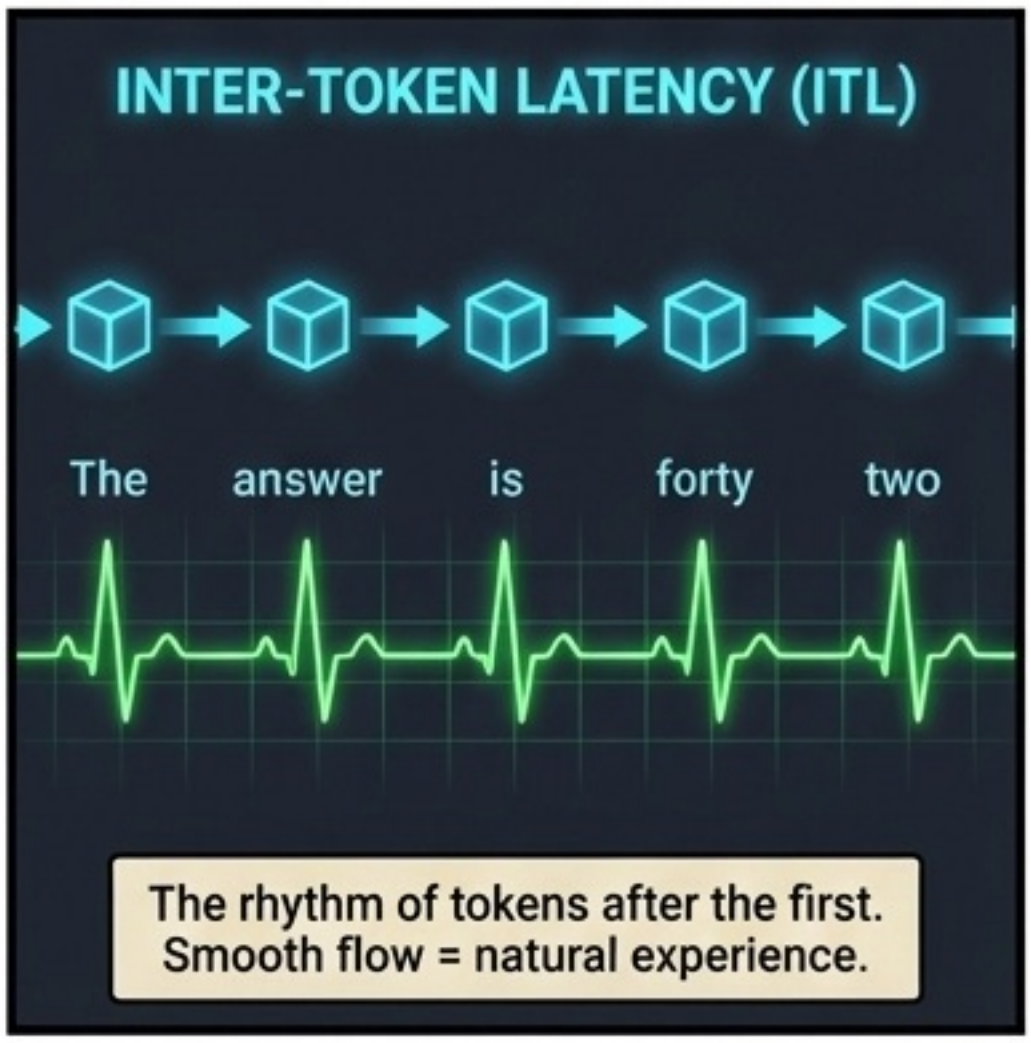
Research	→	Framework-native
Sharing	→	Safetensors
Local	→	GGUF
Production	→	TensorRT/ONNX

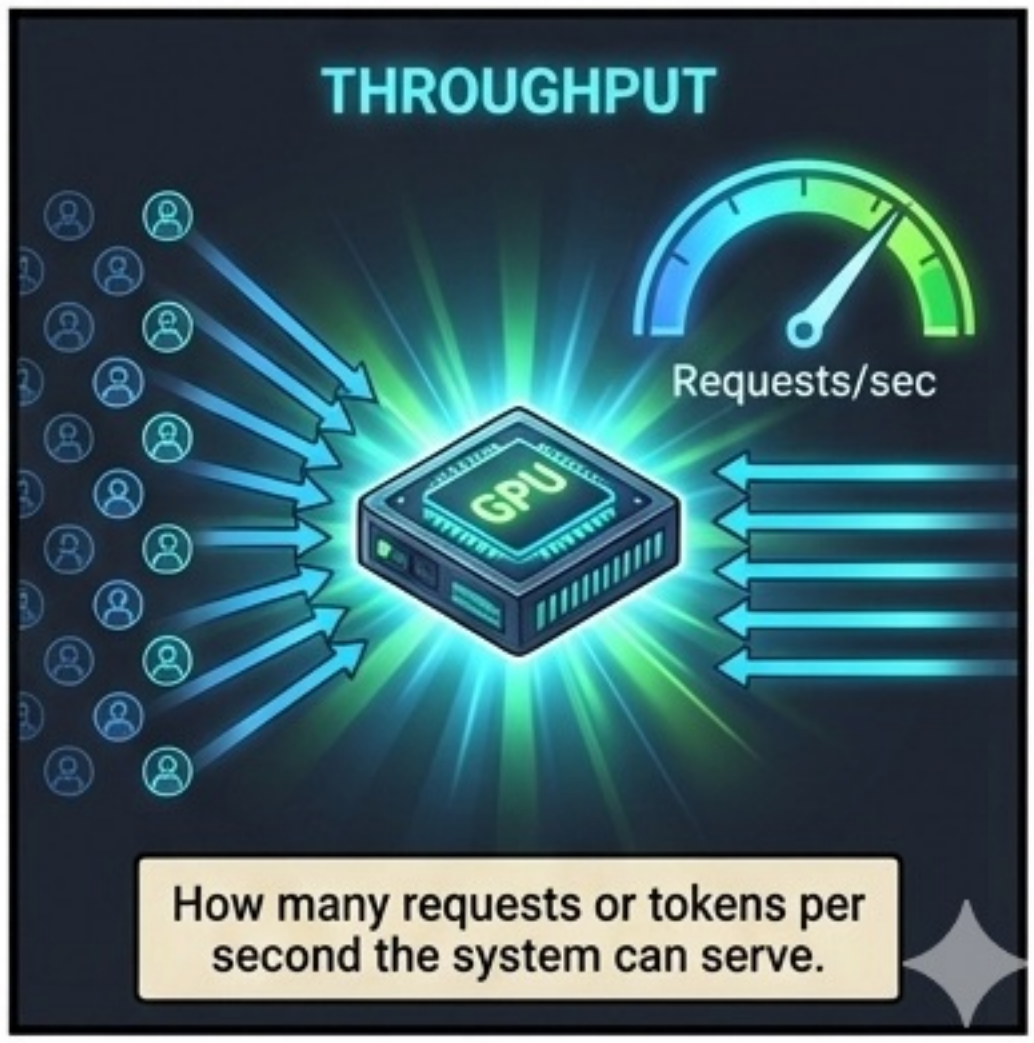



METRICS
12,720
23.26
9.75
METRICS
2,829
METRICS
Let's talk about how we measure inference performance.
Why do some AI apps feel instant while others lag?


Chat:
AI response:
42, The answer,
Request
Response
TOTAL LATENCY
0.5s
End-to-end response time — from request to complete answer.


TIME TO FIRST TOKEN (TTFT)
Chat: Hi
ZIP!
The
How fast the first word appears. Critical for chatbots — makes the app feel alive.


INTER-TOKEN LATENCY (ITL)
The
answer
is
forty
two
The rhythm of tokens after the first. Smooth flow = natural experience.


THROUGHPUT
Requests/sec
GPU
How many requests or tokens per second the system can serve.

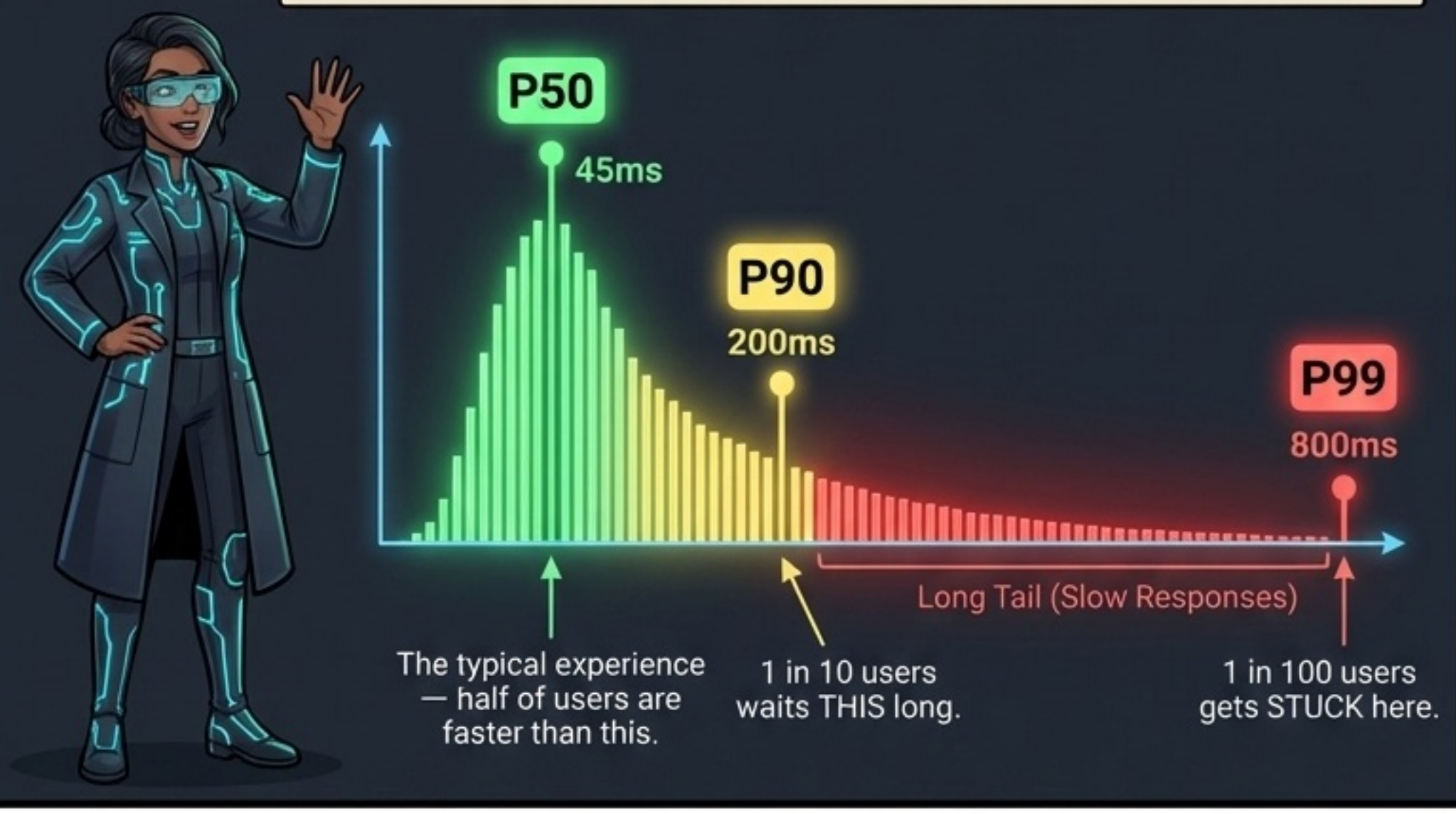
Average latency is 50ms — that's great!

50ms

Average Latency: 50ms

Not so fast. Averages hide the truth.

Percentiles reveal the real distribution. P50 is the median. P90 and P99 show the worst-case experiences that averages hide.



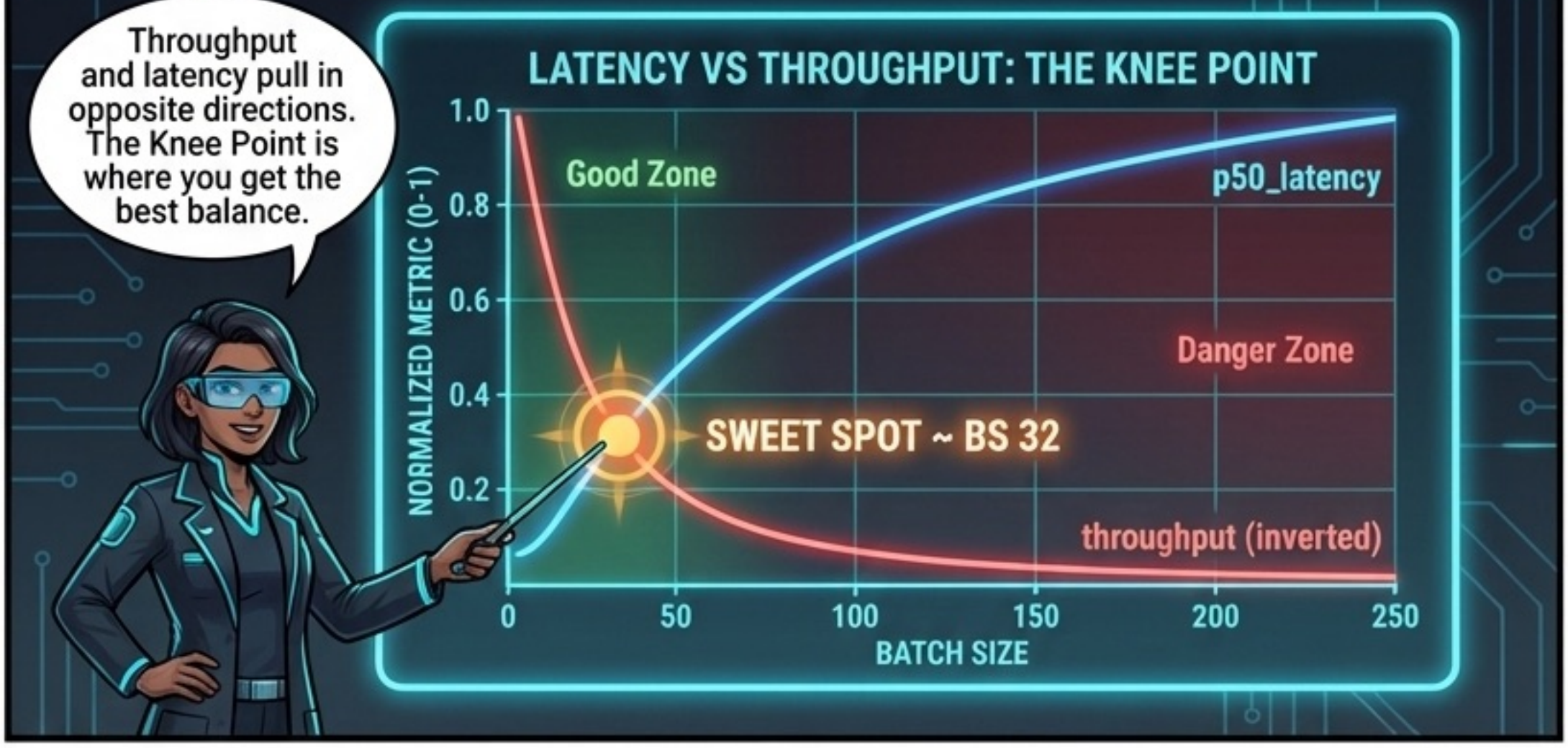
✓ 30ms ✓ 45ms ✓ 50ms ✓ 50ms ✓ 60ms ✓ 60ms

800ms

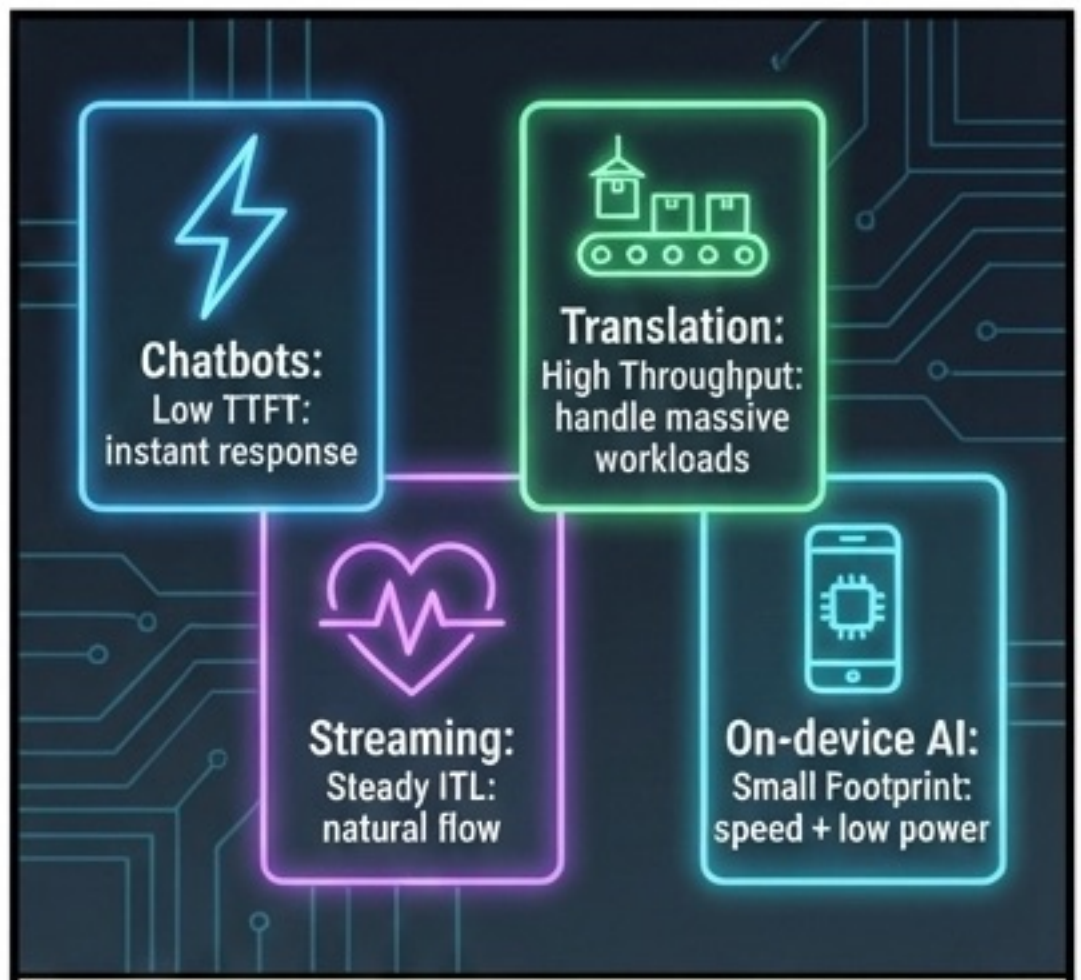
So even if most users are happy, some are having a terrible time?

Exactly. P90 and P99 ensure fairness — no user should get stuck waiting.

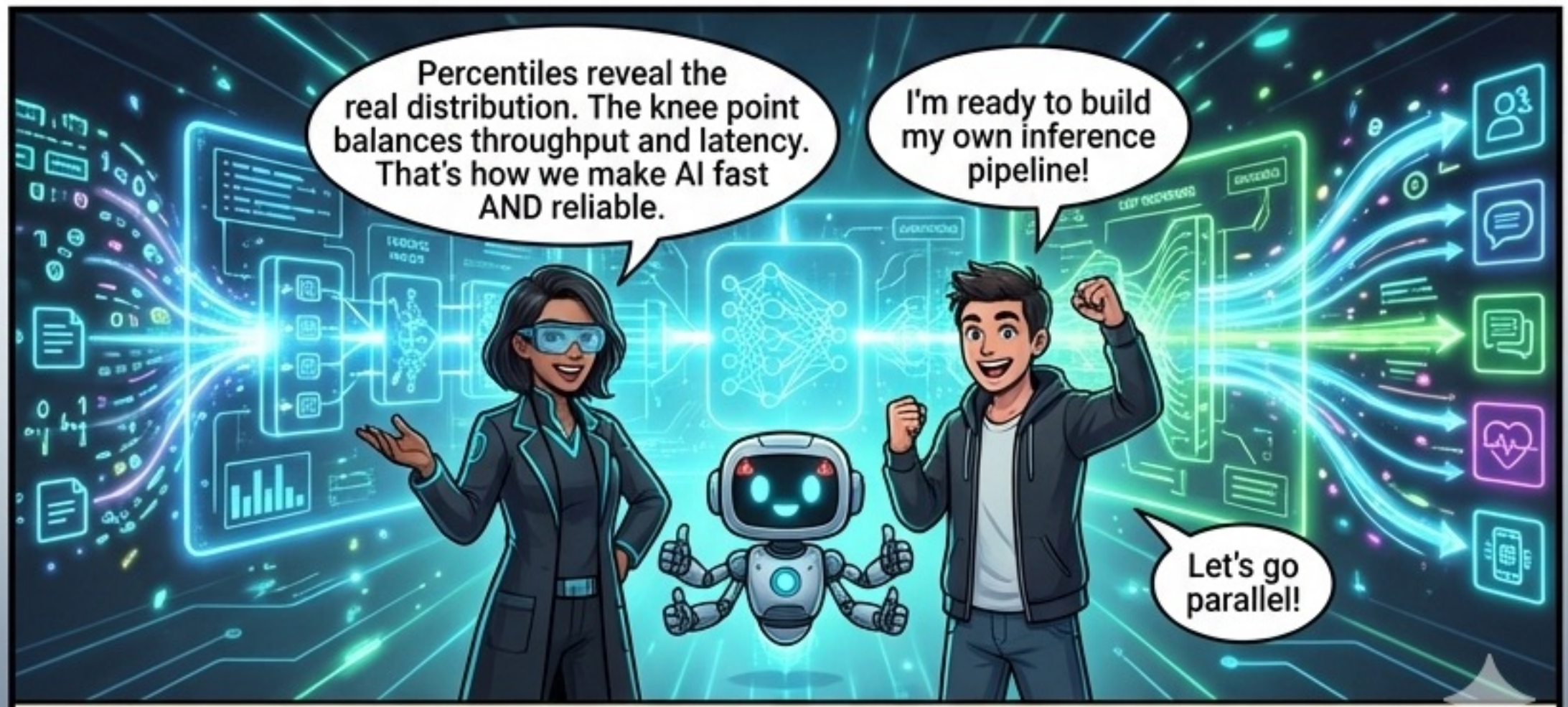
Throughput and latency pull in opposite directions. The Knee Point is where you get the best balance.



Past the knee point, throughput gains flatten while latency skyrockets. Users suffer.



Different applications optimize for different metrics.



Training teaches the model. Inference applies it. Now go build something amazing.