

Aggressive LLMs Optimization: Making Them Work on Tiny Devices

Combining Pruning, Quantization, and Architecture Trimming for
Robust Edge Deployment

About us



Max Navrotsky
Sr. Software
Engineer at @VGS



Oleksandr Gordieiev
DSc

Prof of Software Engineering at
Lutsk National Technical
University



The Challenge. Why Optimize LLM's for Weak Devices?

Large-scale language models (LLMs) typically require significant computational resources, limiting their practical deployment on low-end, resource-constrained devices.

In critical applications - including field operations, offline systems, and embedded devices - such requirements are prohibitive.

Research Goal:

Our objective is to investigate whether aggressive model optimization techniques - such as architecture trimming, pruning, and quantization - can enable an LLM to operate efficiently under extreme hardware constraint.

Demonstration Approach:

To practically evaluate our optimization strategy, we use a real-world applied task:

- ◆ **Generating ISO-compliant software requirements.**

This task serves **only as a structured benchmark** to measure the trade-offs between model compression, resource consumption, and output stability.

Model Selection and Alternatives.

Model	Parameters	Open Source	Optimization Potential	Notes
GPT-2	~117M	✓	Very High	Proven generative strength, highly modifiable
DistilGPT-2	~82M	✓	High	Pre-compressed, less room for further compression
GPT-Neo 125M	~125M	✓	Medium	Similar size to GPT-2, newer but rougher edges
GPT-J 6B	~6B	✓	Very Low	Too large for practical edge optimization
OPT-125M	~125M	✓	Medium	Good candidate, but less mature in pruning tools
BLOOMZ-560M	~560M	✓	Low	Multi-lingual strength, too large for extreme compression

Why GPT-2?

When choosing a foundation model for optimization, we deliberately selected **GPT-2**. Why?

1) **GPT-2 is open-source** — freely accessible and widely supported within the research community.

This accessibility enables unrestricted experimentation, fine-tuning, and aggressive optimization without encountering licensing or API limitations.

2) Secondly, GPT-2 is **relatively lightweight** compared to more recent large language models, particularly its 117M parameter variant.

This smaller size facilitates efficient operation and compression on resource-constrained devices without immediately compromising core generative capabilities.

3) Critically, GPT-2 has **proven generative strength**.

Despite its release in 2019, it consistently produces coherent, grammatically sound, and task-aligned outputs, particularly when fine-tuned for specialized domains.

4) Finally, GPT-2 features a **transparent and well-documented architecture**, making it highly compatible with optimization techniques such as pruning, quantization, and architecture trimming.

This intrinsic flexibility renders GPT-2 an ideal candidate for experiments requiring aggressive model reduction.

**...So if we had only one model to carry with us into the forest on an old laptop
—it would beGPT-2.**

The Plan

Theory

Part 1. Theory – we describe there theoretical concepts that are vital for better knowledge and process understanding

Practical results

Part 2. Practical Results – we describe there our practical research and notes.

Conclusion

Part 3. Conclusions – allows us to structure everything in accurate summary of things that we achieved.

Part 1: Theory

Student Model: The Core Idea

Student Model



Layers



Neurons



Inputs

Parameters → **Weights**

Main idea:

Smaller, Faster, Specialized.

Compressed version of a larger "Teacher" model

Fewer parameters → layers, neurons, heads are reduced

Optimized aggressively: Pruned, Quantized, Architecture Trimmed

Focused on ONE task: ISO-standard Software Requirements generation



Metrics & Optimization Methods

How do we measure success?

- ◆ **Perplexity – Model accuracy** (**lower is better**)
- ◆ **CPU & Memory Usage – Critical for low-resource devices**
- ◆ **Inference Speed – Faster responses**
- ◆ **Model Size – Smaller is always better for weak devices**

General pattern: smaller / faster == better. (usually)

Metrics & Optimization Methods

Types of Optimization:



Non-Aggressive Methods

Knowledge Distillation: Student mimics Teacher closely, minimal changes.



Aggressive Methods

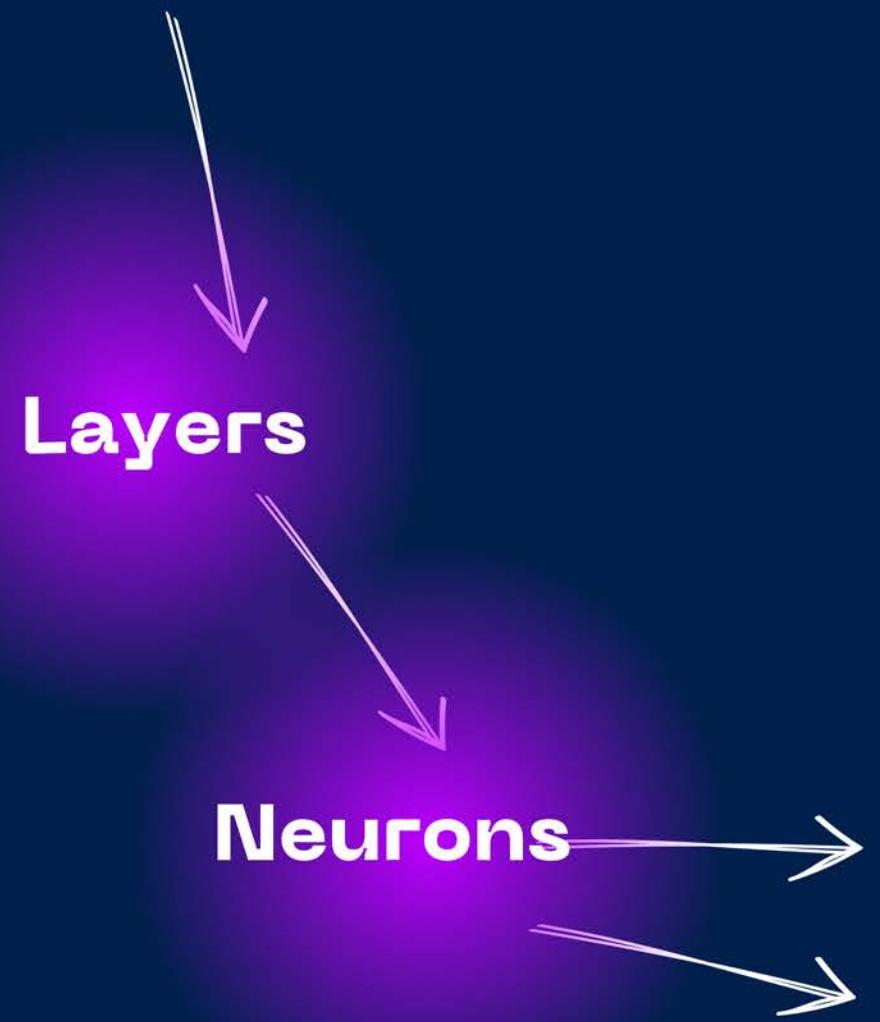
Architecture Trimming: Reducing layers & neurons.

Pruning: Removing less important parameters.

Quantization: Changing weights from float32 → INT8.

Metrics & Optimization Methods: Architecture Trimming

Student Model



Main Concept: "If it's too big—make it smaller."

Imagine your model as a building. The original model is like a skyscraper—lots of floors(layers), many rooms (neurons).
Architecture trimming = removing unnecessary floors and rooms.
Result: smaller, faster model that's easier to run on weak hardware.

What Exactly We Optimize:

Layers: We reduce the number of processing stages (transformations) in the model.
Neurons: We decrease the number of computational units per layer.
Hidden dimensions: We shrink the size of internal representations(embedding and intermediate states).

Glossary:

Layer: A step or stage in neural network processing.
Neuron: A small computational unit inside a layer that performs calculations.

Metrics & Optimization Methods:

✖ Pruning

Main Concept:

"If it's not important – cut it off."

Think of your model like a tree, where each branch represents a connection between neurons, and each branch has a weight (parameter).

Some branches are critical for the tree's health (model's performance), others barely contribute.

Pruning = cutting away the weakest, least useful branches (lowest-weighted connections).

What Exactly We Optimize:

Weights (Parameters):

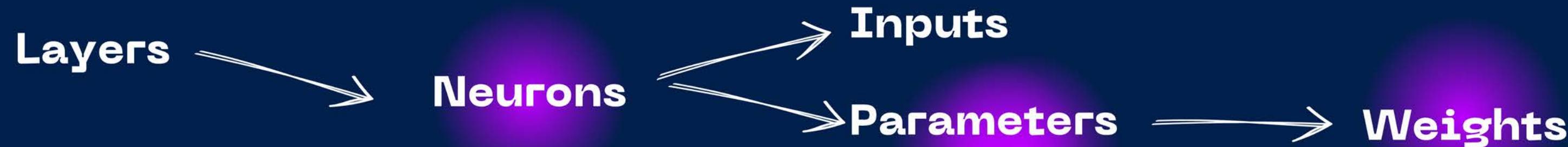
We remove connections (weights) that have the smallest absolute values.

These small weights have minimal influence on the neuron's output and on the model's decision-making process.

Indirectly, Neurons:

If pruning removes most incoming/outgoing connections to a neuron, that neuron effectively becomes inactive (and can later be removed entirely in deeper optimization stages).

Student Model



Glossary:

Weights (Parameters): Numbers that determine the strength of connections between neurons.

Pruning: Technique to remove unnecessary or weak connections without harming model performance.

Metrics & Optimization Methods:

⚖️ Quantization

Main Concept: "If precision isn't critical, why use heavy tools?"

Imagine carrying water:

Float32 (original weights) = huge bucket (very precise, but heavy).

INT8 (quantized weights) = smaller bucket (less precise, but lighter and faster to carry).

Quantization = changing big heavy weights (Float32) into smaller lightweight ones (INT8).

Result: faster calculation, less memory usage, slightly less precise—but usually still good enough.

What Exactly We Optimize:

Weights:

Instead of using 32-bit floating-point numbers for each parameter, we store them as 8-bit integers.

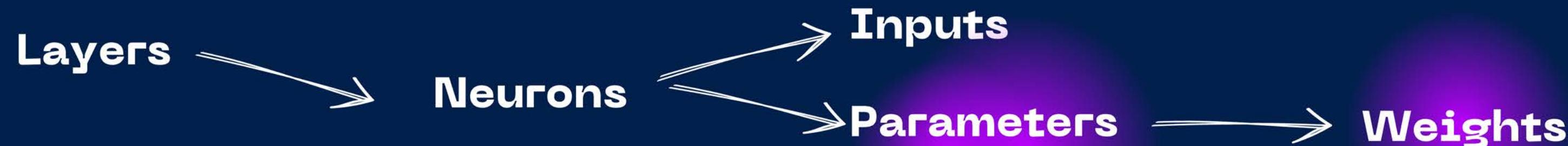
Model size:

Quantization reduces the memory needed to store weights by 4x.

Computation speed:

CPUs and some hardware accelerators can perform 8-bit operations much faster than 32-bit floating-point operations.

Student Model



Metrics & Optimization Methods:

❖ Knowledge Distillation (Non-Aggressive)

Main Idea: "Learning by imitation, not reinvention."

Imagine a student copying notes from an experienced teacher. Teacher model = original, complex model.

Student model = simpler, smaller model learning from the teacher's outputs.

Distillation = student tries to copy exactly how teacher behaves, without drastic changes.

Good if you have resources, but fails if you aggressively reduce size too much.

So, it's VERY important that student's model architecture is the same or almost the same as teacher's one.

What Exactly We Optimize:

Knowledge Representation:

Instead of optimizing the model's architecture or its parameters directly, distillation optimizes how knowledge is encoded within a smaller model.

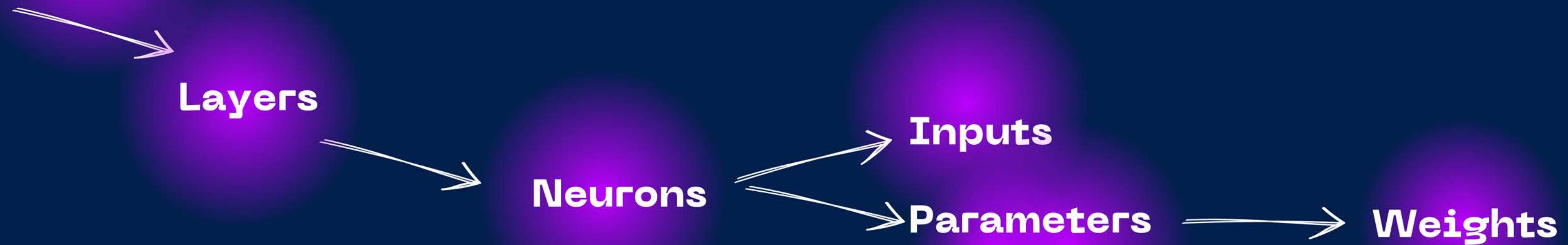
Training Targets:

The student model learns not just from raw labels (as in standard supervised learning), but from the teacher's soft outputs (probability distributions), which contain richer information about task relationships.

Parameter Efficiency:

Through imitation, the student compresses essential knowledge into fewer parameters without necessarily needing to match the teacher's size.

Student Model



Glossary:

Teacher Model: The larger, original model with lots of knowledge and parameters.

Student Model: Smaller model, learning by mimicking the teacher.

Distillation: Training method where the student tries to copy the teacher's behavior.

Metrics & Optimization Methods: General methods comparison

Method	Type	Complexity	Impact on accuracy	Impact on speed & size	Notes
Architecture Trimming	⚡	🟡	🔴	🟢	Fewer layers and neurons
Pruning	⚡	🟢	🟢	🟡	Removes unnecessary parameters
Quantization	⚡	🟡	🟡	🟢	Float32 → INT8, lighter weights
Knowledge Distillation	🧠	🔴	🟢	🟡	Teacher-student imitation

Part 2: Practical research

Order of experiments

Fine-tuning GPT-2



Architecture Trimming



Pruning



Quantization

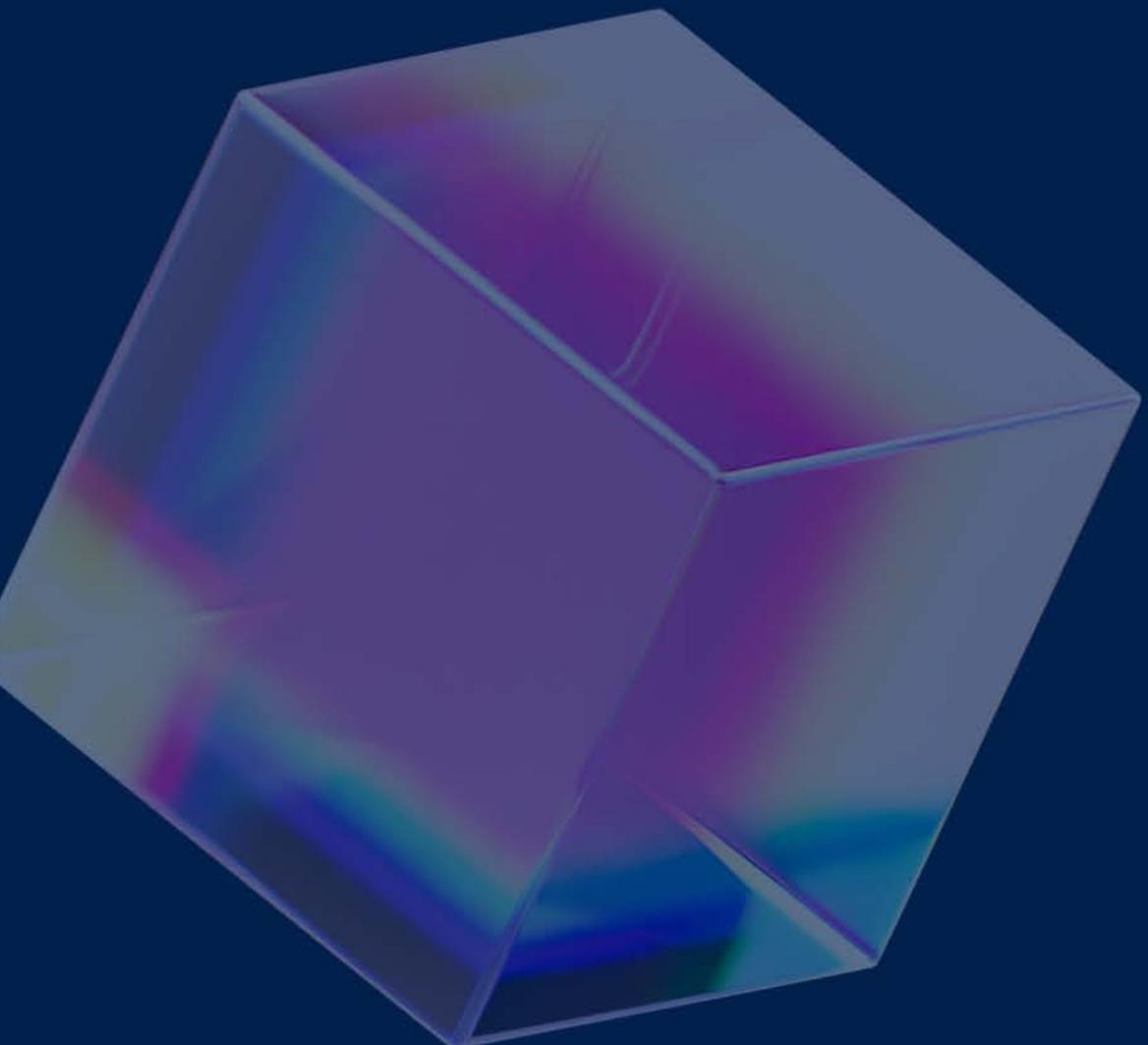
This part is split into several steps:

1. **Baseline – Fine-Tuned GPT-2**
2. **Architecture Trimming Only**
3. **Distillation**
4. **Pruning**
5. **Quantization**
6. **Combined: Trimming + Pruning + Quantization**

And Later as a Final Result: Comparison Graphs and Key Observations

Before Fine-Tuning – Raw GPT-2

- 🧠 Pre-trained GPT-2 (117M) without fine-tuning
- 🚫 Not specialized — generic language understanding
- ▼ Struggles with task-specific phrasing & structure



Before Fine-Tuning – Raw GPT-2

Performance Snapshot

Metric	Value
Perplexity	~1.02
CPU Usage	~45.10%
Memory Usage	~980.83 MB
Inference Speed	~0.14 sec/gen
Model Size	~510 MB

Use Case Example Prompts (Generated):

Prompt: The software must be able to

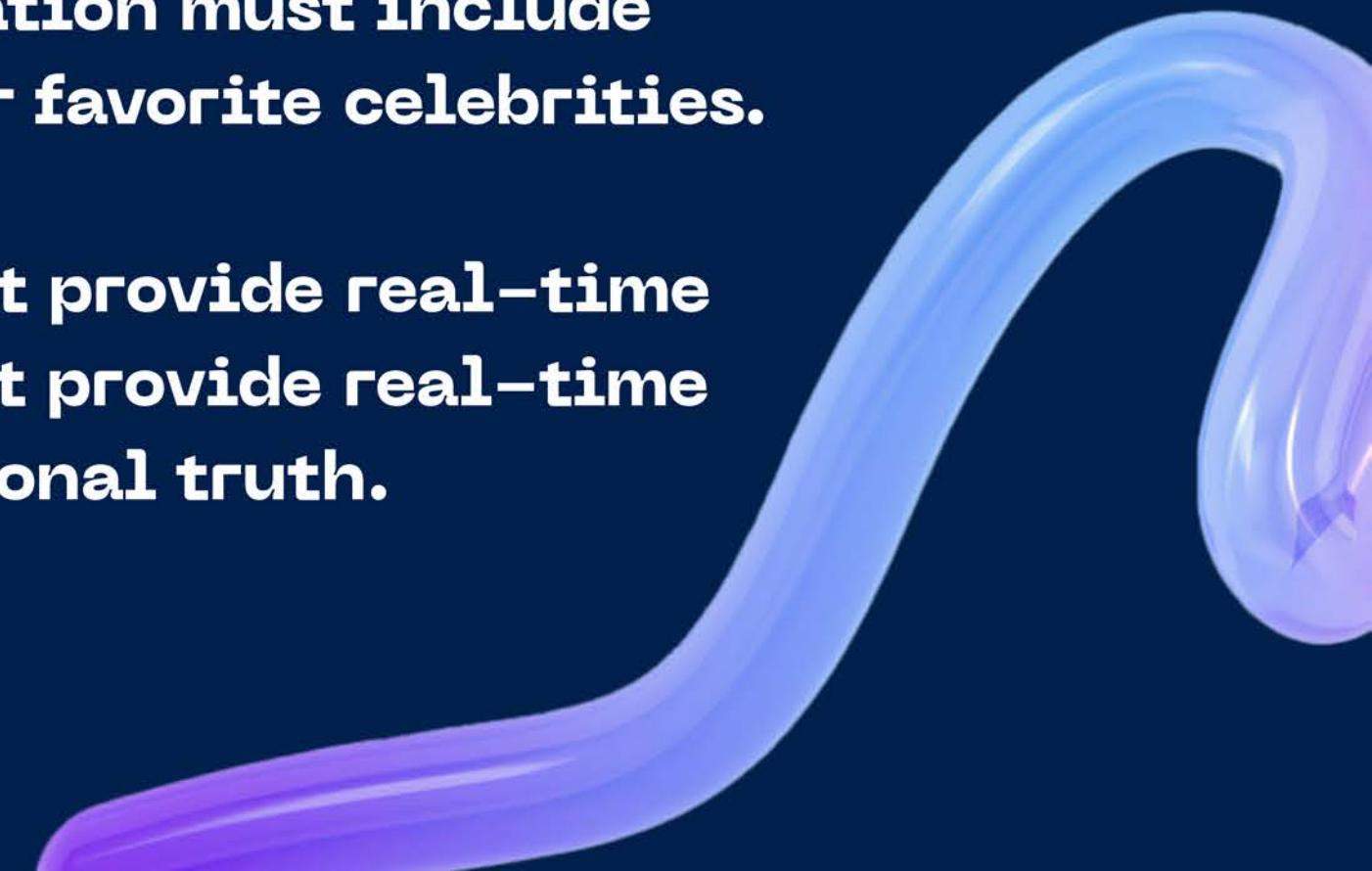
Output: The software must be able to understand how the music and weather influence the morning.

Prompt: User authentication must include

Output: User authentication must include the ability to enjoy your favorite celebrities.

Prompt: The system must provide real-time

Output: The system must provide real-time representation of emotional truth.



Step 0: Baseline – Fine-Tuned GPT-2

- ✓ No optimizations applied
- 🔧 Model is fine-tuned on ISO-compliant software requirement dataset
- 🧠 Full GPT-2 architecture (117M params)
- 🧪 Acts as our reference point for comparisons

Use Case Example Prompts (Generated):

Prompt: The software must be able to

Output: The software must be able to 10,000 transactions per second.

Prompt: User authentication must include

Output: User authentication must include user credentials before granting access.

Prompt: The system must provide real-time

Output: The system must provide real-time notifications.

After Fine-Tuning – Raw GPT-2

Performance Snapshot

Raw GPT-2 Model

Metric	Value
Perplexity	~1.02
CPU Usage	~45.10%
Memory Usage	~980.83 MB
Inference Speed	~0.14 sec/gen
Model Size	~510 MB

Fine-tuned GPT-2 Model

Metric	Value
Perplexity	~1.03
CPU Usage	~37.50%
Memory Usage	~942.83 MB
Inference Speed	~0.12 sec/gen
Model Size	~490 MB

Step 1: Architecture Trimming

- Removed half of GPT-2's layers and reduced hidden size
- From 12 → 6 layers, 768 → 384 dimensions, 12 → 6 heads
- Same inputs, lighter architecture, faster execution



How did we pick which layers to trim?

Layer Selection Strategy:

We removed layers symmetrically from the center, preserving:

Bottom layers – for basic syntax & token-level understanding

Top layers – for higher-level structure and output composition

This balanced trimming helps maintain both low-level and high-level processing.

(But to be honest, trimming evenly usually works well — random trimming is less stable, but not always worse.)

Architecture Trimming

Performance Snapshot

Raw GPT-2 Model

Metric	Value
Perplexity	~1.02
CPU Usage	~45.10%
Memory Usage	~980.83 MB
Inference Speed	~0.14 sec/gen
Model Size	~510 MB

Architecture-trimmed
GPT-2 Model

Metric	Value
Perplexity	~1.11
CPU Usage	~24.60%
Memory Usage	~1025.50 MB
Inference Speed	~0.07 sec/gen
Model Size	~295 MB



Generation Examples:

Prompt: The software must be able to
Output: The software must be able to
10,000 transactions per second.

Prompt: User authentication must include
Output: User authentication must
include user credentials before access.

Prompt: The system must provide real-time
Output: The system must provide
real-time notifications.

CPU vs. Memory – The Optimization Tradeoff

- ▼ When we reduce CPU usage, we often increase RAM usage
- ⚙️ Optimizations like quantization and layer trimming reduce processing time
- ⚠️ But they can increase intermediate tensor sizes, or cause repetition that needs more cache
- 🧠 Memory is fast, CPU is expensive — especially on weak devices

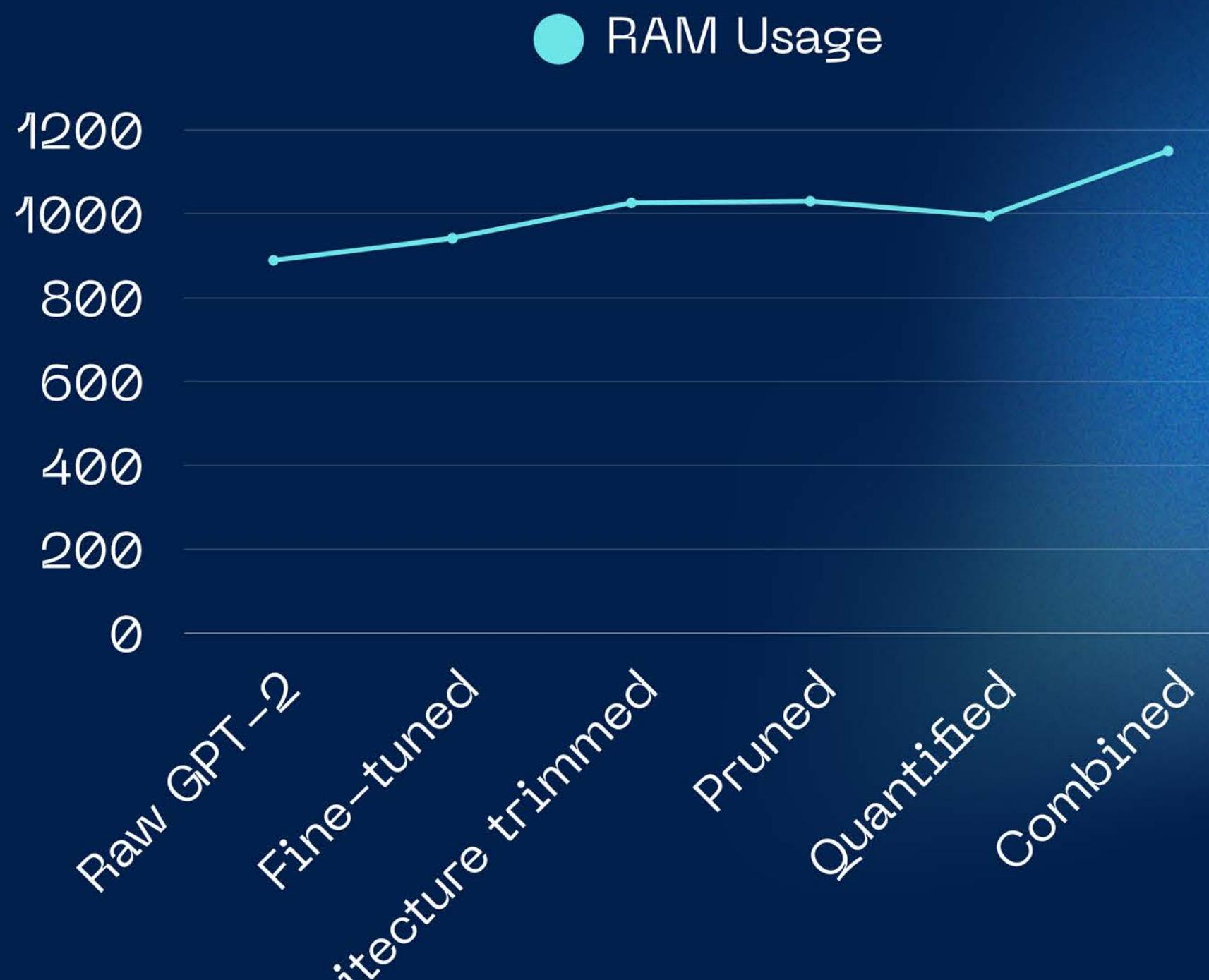
💡 Why This Matters:

- CPU time = energy cost

On battery-powered or embedded devices,
CPU is the biggest power draw

- RAM is more available but not unlimited
Cheap to access, but limited
on old laptops or microcontrollers

CPU vs. Memory – The Optimization Tradeoff

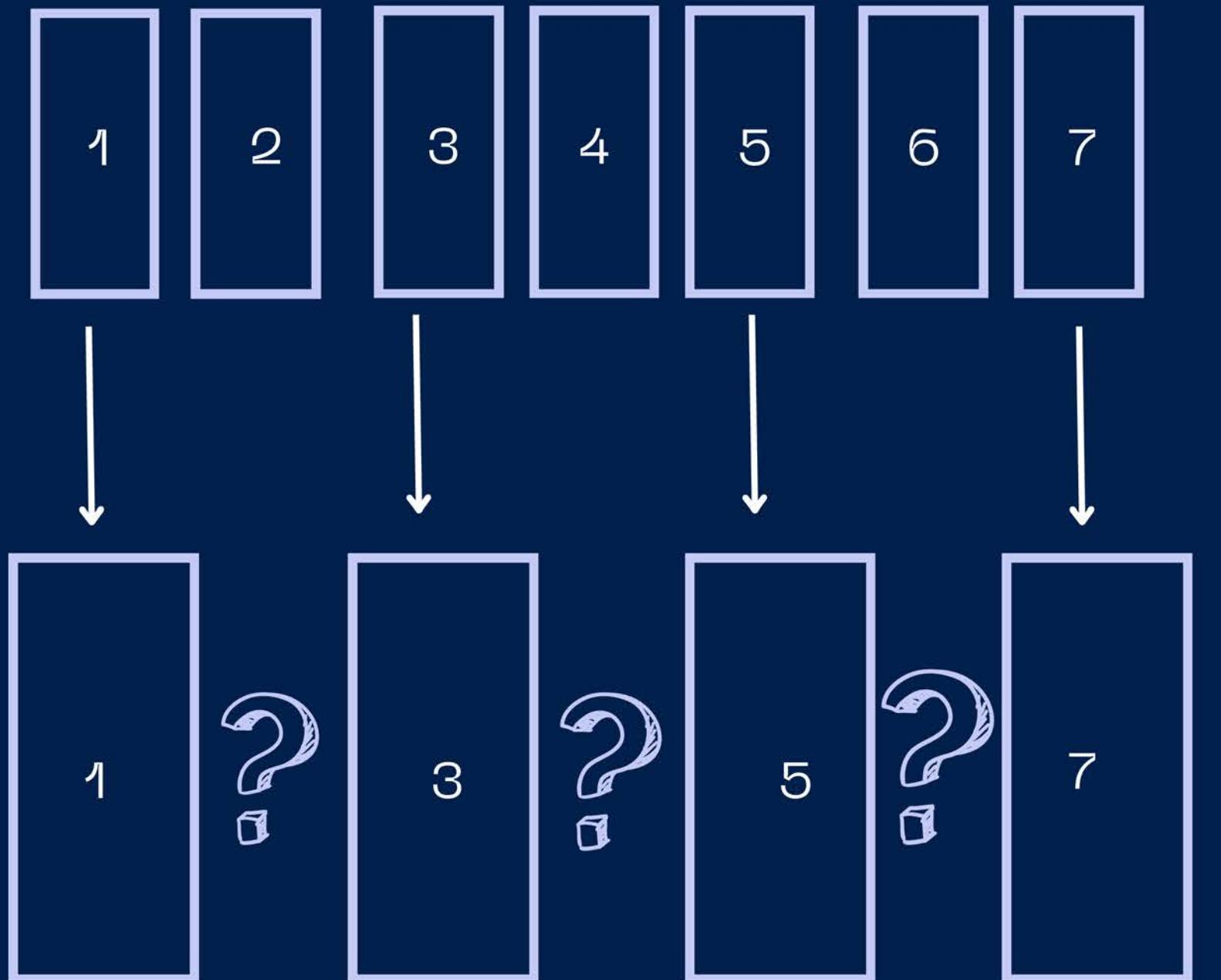


- 1) **Larger intermediate tensors:** Optimized models may increase activation sizes to maintain numerical stability.
- 2) **Recomputation avoidance:** To save CPU effort, intermediate results are cached in RAM instead of being recomputed.
- 3) **Batch processing overhead:** Parallel execution of operations increases temporary memory usage during inference.

Distillation – When It Breaks

- **Student model trained to mimic a bigger Teacher model
But the student is too small to understand the teacher
Combined with architecture trimming → catastrophic failure**
- **What Went Wrong?**
**Student doesn't have enough layers/neurons to "understand" teacher outputs
Loss doesn't converge — student starts repeating or hallucinating
Compression + distillation = incompatible when capacity is too low**

Distillation – When It Breaks



Student Model Collapse:

- The teacher model has a **full set of neurons** to process and represent complex knowledge.
- The student model, being aggressively trimmed or pruned, lacks enough neurons to match and mimic all critical pathways.
- As a result, some knowledge pathways are lost entirely — the student cannot reconstruct or imitate them.
- This leads to hallucinations (generating nonsense) and repetitions (getting stuck in loops).

As a result we finish in critical failure point.

Distillation – When It Breaks

● Performance Snapshot

Raw GPT-2 Model

Metric	Value
Perplexity	~1.02
CPU Usage	~45.10%
Memory Usage	~980.83 MB
Inference Speed	~0.14 sec/gen
Model Size	~510 MB

**Distilled GPT-2 Model
(after architecture trimming)**

Metric	Value
Perplexity	~4.05
CPU Usage	~40.60%
Memory Usage	~1005 MB
Inference Speed	~0.06 sec/gen
Model Size	~440 MB



Broken Generation Examples:

Prompt: The software must be able to
Output: The software must be able to
to to to to to to to to...

Prompt: User authentication must
include

Output: User authentication must
include must must must must must
must must...

Prompt: The system must provide
real-time

Output: The system must provide
real-
time time time time time time time...

Pruning – Cut the Branches, Keep the Brain

斧 We remove parameters that have low or no contribution

🧠 Model learns to ignore them during training

Pruning = deleting those "dead weights"

✓ Architecture stays the same — just fewer active connections

🔍 How We Choose Weights to Prune:

We apply L1-unstructured pruning — this means:

We rank all weights by their absolute value

Then remove the smallest ones (those closest to zero)

Example: amount = 0.4 → we keep only 60% of the strongest weights

💡 Think of it like muting the quietest voices in a noisy room — the important signals still come through loud and clear.

Why It Works:

Many weights in a large model are close to zero

Removing them has little impact on output

Less memory & faster computation — but only slightly



Pruning – Cut the Branches, Keep the Brain

Performance Snapshot

Raw GPT-2 Model

Metric	Value
Perplexity	~1.02
CPU Usage	~45.10%
Memory Usage	~980.83 MB
Inference Speed	~0.14 sec/gen
Model Size	~510 MB

Pruned GPT-2 Model

Metric	Value
Perplexity	~1.11
CPU Usage	~33.80%
Memory Usage	~1029.31 MB
Inference Speed	~0.08 sec/gen
Model Size	~400 MB

Output Examples:

Prompt: The software must be able to
Output: The software must be able to
10,000 transactions per second.

Prompt: User authentication must
include
Output: User authentication must
include

Prompt: The system must provide real-
time
Output: The system must provide
real-time notifications.

Quantization – Big Gains with Small Numbers

- 🎯 Converts model weights from float32 → int8
- 🧠 Float32 = very precise but heavy
- 💡 INT8 = less precise but much lighter → faster inference, less CPU load
- ✓ No architectural changes needed
- ✗ Performed after fine-tuning and pruning



Why It Works:

CPUs process 8-bit math much faster than 32-bit

Less RAM used per number

Slight drop in accuracy – but for fine-tuned models, it's acceptable



Extra Glossary Note:

Post-training quantization: Quantization applied to a fully trained model (common & easier)

QAT (Quantization-Aware Training): A more complex method we did not use

Quantization – Big Gains with Small Numbers

Performance Snapshot

Raw GPT-2 Model

Metric	Value
Perplexity	~1.02
CPU Usage	~45.10%
Memory Usage	~980.83 MB
Inference Speed	~0.14 sec/gen
Model Size	~510 MB

Quantified GPT-2 Model

Metric	Value
Perplexity	~1.12
CPU Usage	~12.50%
Memory Usage	~994.45 MB
Inference Speed	~0.04 sec/gen
Model Size	~320 MB

Output Examples:

Prompt: The software must be able to
Output: The software must be able to 10,000 transactions per second.

Prompt: User authentication must include

Output: User authentication must include

Prompt: The system must provide real-time

Output: The system must provide real-time notifications.

Combined Optimization – Our Best Result

Trimming + Pruning + Quantization = 💎

● We apply all methods in the right order:

Fine-Tune GPT-2 for one task (ISO requirements)

✂ Trim model to 6 layers, 384 neurons

✂ Prune 40% of smallest weights (L1)

⚖ Quantize to INT8 for CPU efficiency

🔍 We don't use distillation at all:

It just does not stack with aggressive methods combined above.

💡 In this case there is no methods that will allow distillation work with aggressive methods – as architecture difference is too drastic.

Combined Optimization – Our Best Result

Performance Snapshot

Raw GPT–2 Model

Metric	Value
Perplexity	~1.02
CPU Usage	~45.10%
Memory Usage	~980.83 MB
Inference Speed	~0.14 sec/gen
Model Size	~510 MB

GPT–2 Model optimized with combined methods.

Metric	Value
Perplexity	~1.03
CPU Usage	~22.00%
Memory Usage	~1135.25
Inference Speed	~0.0306 sec/gen
Model Size	~115.74 MB

Output Examples (Still Crystal Clear):

Prompt: The software must be able to

Output: The software must be able to
10,000 transactions per second.

Prompt: User authentication must include

Output: User authentication must include user credentials before granting access.

Prompt: The system must provide real-time

Output: The system must provide real-time notifications.

Combined Optimization – Our Best Result

CPU Usage



Optimized Student model reduces CPU load by almost half compared to raw GPT-2.

This makes it significantly more suitable for battery-powered or CPU-only devices, where energy efficiency is critical.

▼ Less CPU → faster response times + lower energy cost

Combined Optimization – Our Best Result

Memory Usage



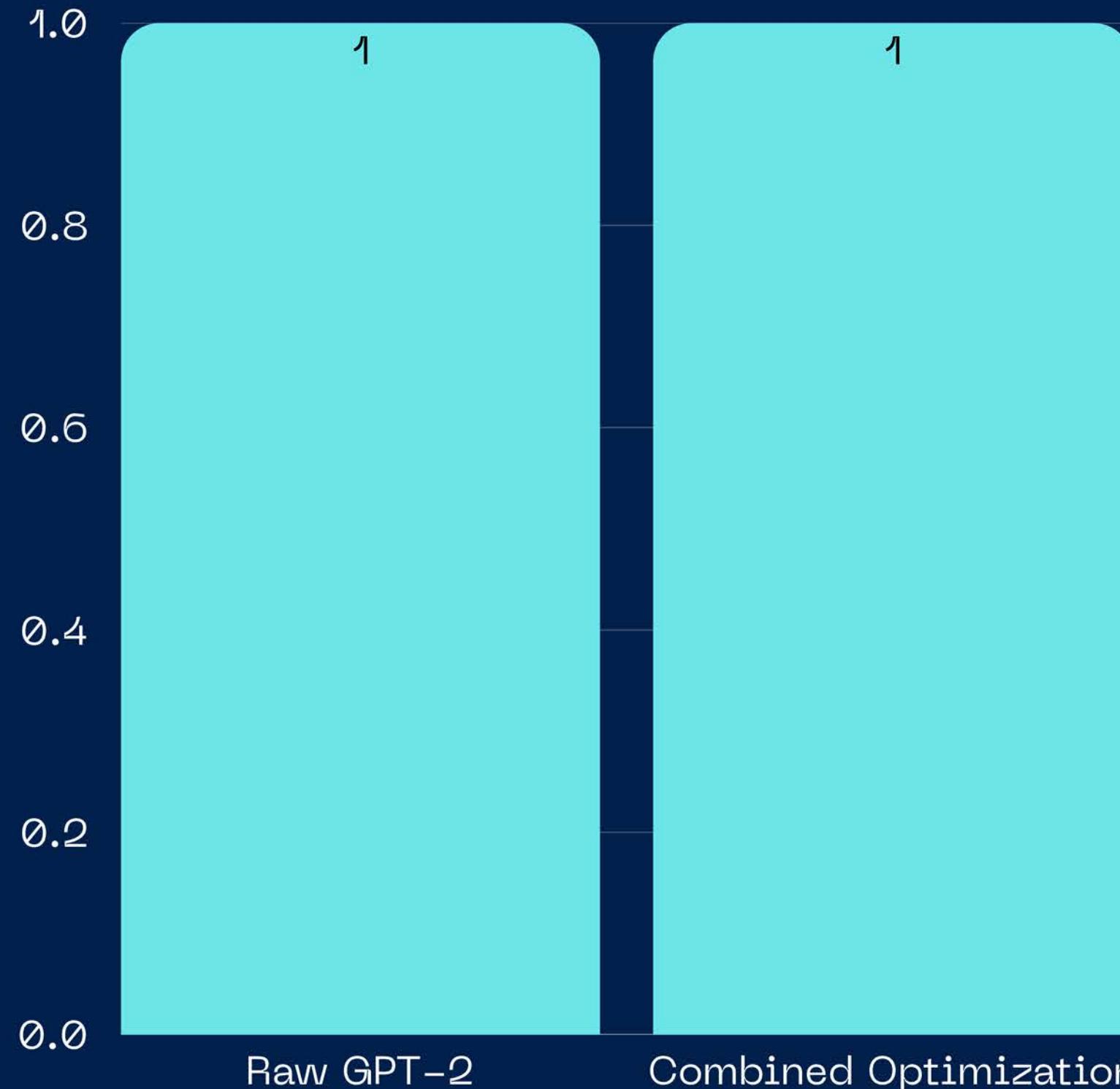
While the Student model uses less CPU, it requires slightly more RAM due to architectural changes and optimization overhead (like intermediate tensor sizes during quantization).

⚠ Trade-off: Lower CPU cost comes with a modest RAM increase

Still acceptable on most low-end laptops or embedded boards

Combined Optimization – Our Best Result

Perplexity



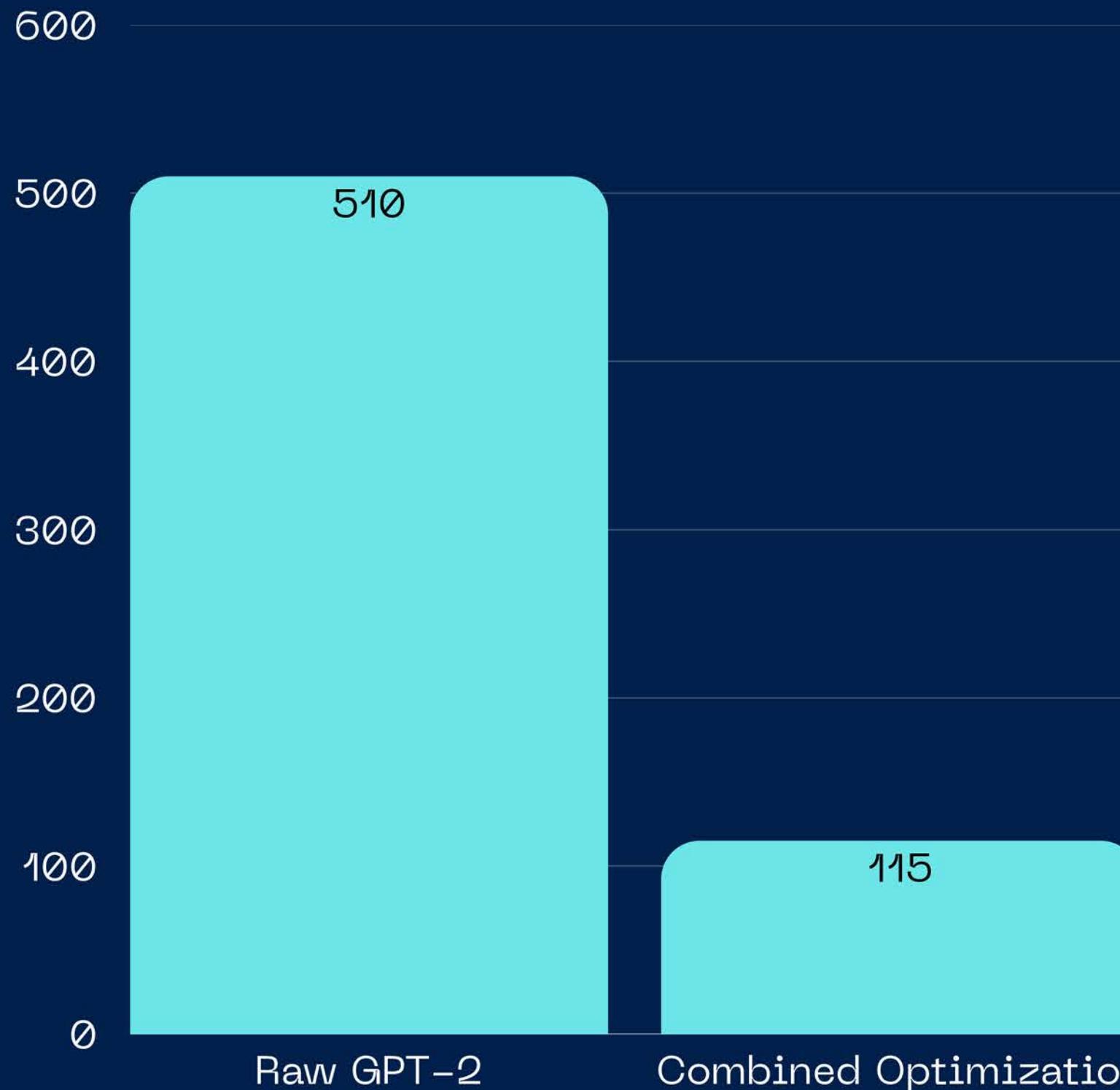
Despite aggressive optimization, both models maintain the same perplexity (~1.03).

This confirms that fine-tuning protects output quality, even after trimming, pruning, and quantization.

✓ Optimization ≠ degradation — if fine-tuning is done right

Combined Optimization – Our Best Result

Model size



Description:

Model size dropped from 490MB → 115MB, making it much easier to store, transfer, and deploy — especially in offline environments (USB, IoT, airgapped systems).

 **Smaller model, same brain**

Part 3: Conclusion

If These Methods Are So Good... Why Not Always Use Them?

"If quantization, pruning, and trimming work so well — why aren't they just applied to all models out-of-the-box?"

Answer:

- ✗ Generic models aren't built to survive heavy surgery.
- ✓ Fine-tuned models are.

Why Fine-Tuning Matters

- **Fine-tuning aligns the model to a single, atomic task**
- **That task-specific focus makes the model resistant to aggressive compression**
- **Out-of-the-box models are generalists → trying to do everything = fragile**



Our Case:
We trained the model to generate ISO-style software requirements — one narrow, well-structured task → easy to compress, hard to break.

If These Methods Are So Good... Why Not Always Use Them?

Resistance – Why Some Models Survive Optimization Better

📘 **Model Resistance** = How well a model maintains output quality after aggressive compression

🧠 High Resistance:

Fine-tuned for one clear task (e.g., ISO requirement generation)

Strong internal signal → low dependency on full architecture

Quality drops slowly even with heavy pruning or quantization

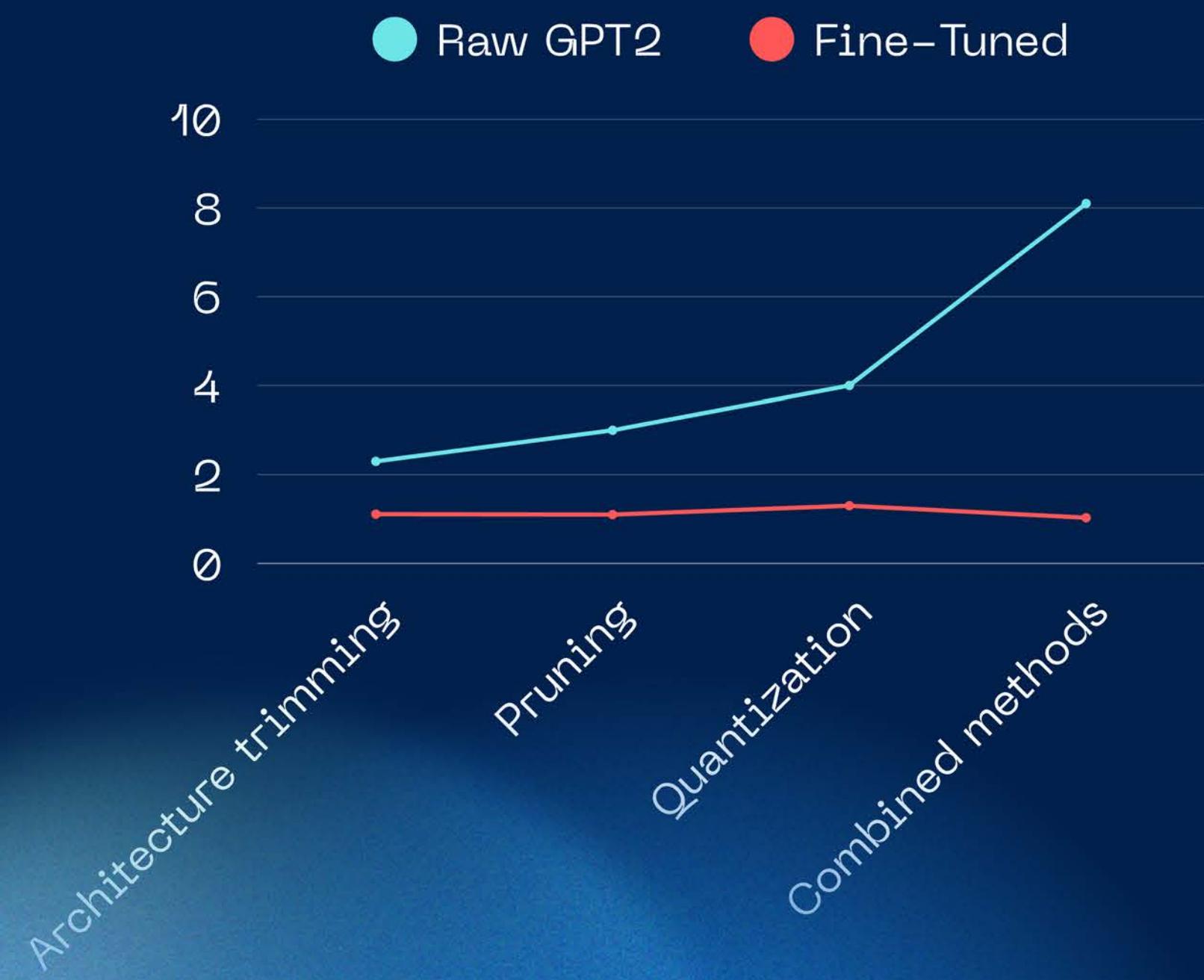
🌟 Low Resistance:

General-purpose model (chat, summarization, Q&A, etc.)

Many overlapping skills → higher fragility

Small optimization = sharp drop in performance

If These Methods Are So Good... Why Not Always Use Them?



The same optimization methods produce dramatically different results depending on the model.

- ◆ Fine-tuned models remain stable — perplexity grows slowly.
- ◆ General-purpose models break fast — perplexity skyrockets.

🧩 Why?

Fine-tuning focuses the model on a single task, reducing internal noise.

General models are too broad — even small cuts disrupt everything.

✓ Optimizations work best when the model knows what it's supposed to do.

Fine-Tuning: The Most Powerful Optimization of All

We talk a lot about trimming, pruning, and quantizing...

But what enables all of it to work?

Fine-tuning.



Why Fine-Tuning Is Key:

It transforms the model into a specialist

Removes ambiguity → less reliance on full architecture

Makes the model more resistant to degradation

Enables combining aggressive methods without collapse

Without Fine-Tuning:

Pruning breaks meaning

Quantization introduces noise

Trimming deletes critical reasoning paths

With Fine-Tuning:

Everything stays coherent — even at 75% compression

"Fine-tuning isn't just an extra step — it's what makes optimization possible."

Final Takeaways & Terminology Recap

📌 Main Points to Remember:

- 🧠 **Fine-tuning is the true enabler — it makes compression possible without collapse**
- ✂️ **Architecture Trimming reduces depth and size**
- 🔪 **Pruning removes low-impact weights**
- ⚖️ **Quantization boosts CPU efficiency**
- 💥 **Distillation fails when applied to aggressively minimized students**
- 🧬 **Resistance is key: fine-tuned models resist degradation far better than general models**

"We didn't just shrink a model — we built a focused, efficient specialist.
And fine-tuning made it all possible."

Thank You!



