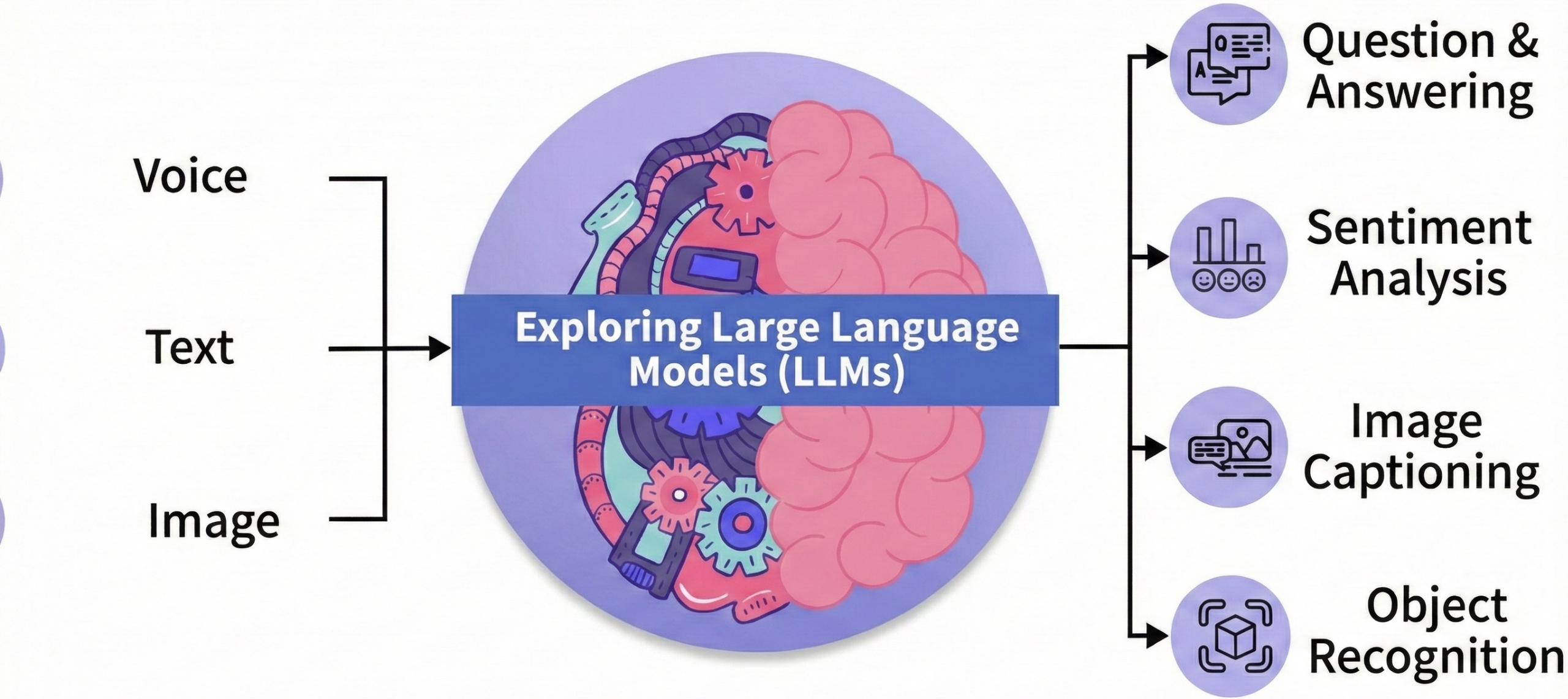

Voice
Text
Image
Exploring Large Language Models (LLMs)
Question & Answering
Sentiment Analysis
Image Captioning
Object Recognition

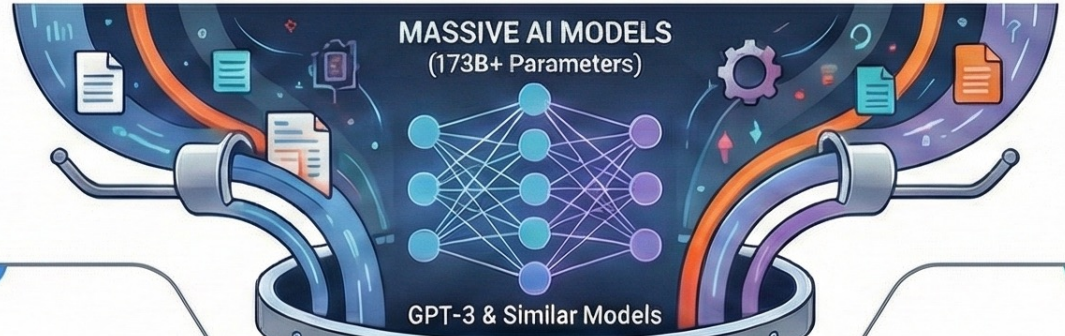
Teaching a Smart Assistant New Skills

- Your assistant knows **everything general**
- You want it to specialize in **Law, Medicine, or Coding**
- **Problem:** How to teach new skills **without retraining the whole brain**

Fine-Tuning Large Language Models

- GPT-3 has **175 billion parameters**
- Full fine-tuning for each task is **expensive & impractical**
- **Storage** & **compute** requirements are **massive**
- **Question:** How can we adapt them **efficiently**?

The Bottleneck: Why Full Fine-Tuning Fails at Scale



HARDWARE & MEMORY CONSTRAINTS

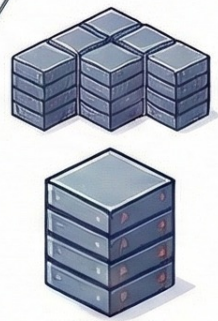
1.2 TERABYTES OF VRAM REQUIRED

Updating all 1758 parameters requires memory for model, gradients, and optimizer states.

Exceeds Consumer Hardware Capacity

Requirements for surpass standard 24GB GPU capacities found in high-and consumer hardware.

SCALING & STORAGE INEFFICIENCY



FULL-SIZED MODEL COPY

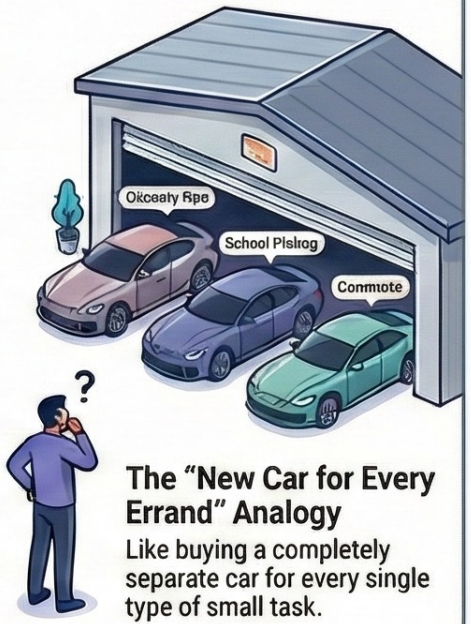
Text Summarization

Customer Service

Code Generation

350GB
STORAGE WASTE PER TASX

Every specialized task requires saving a unique, full-sized copy of the model.



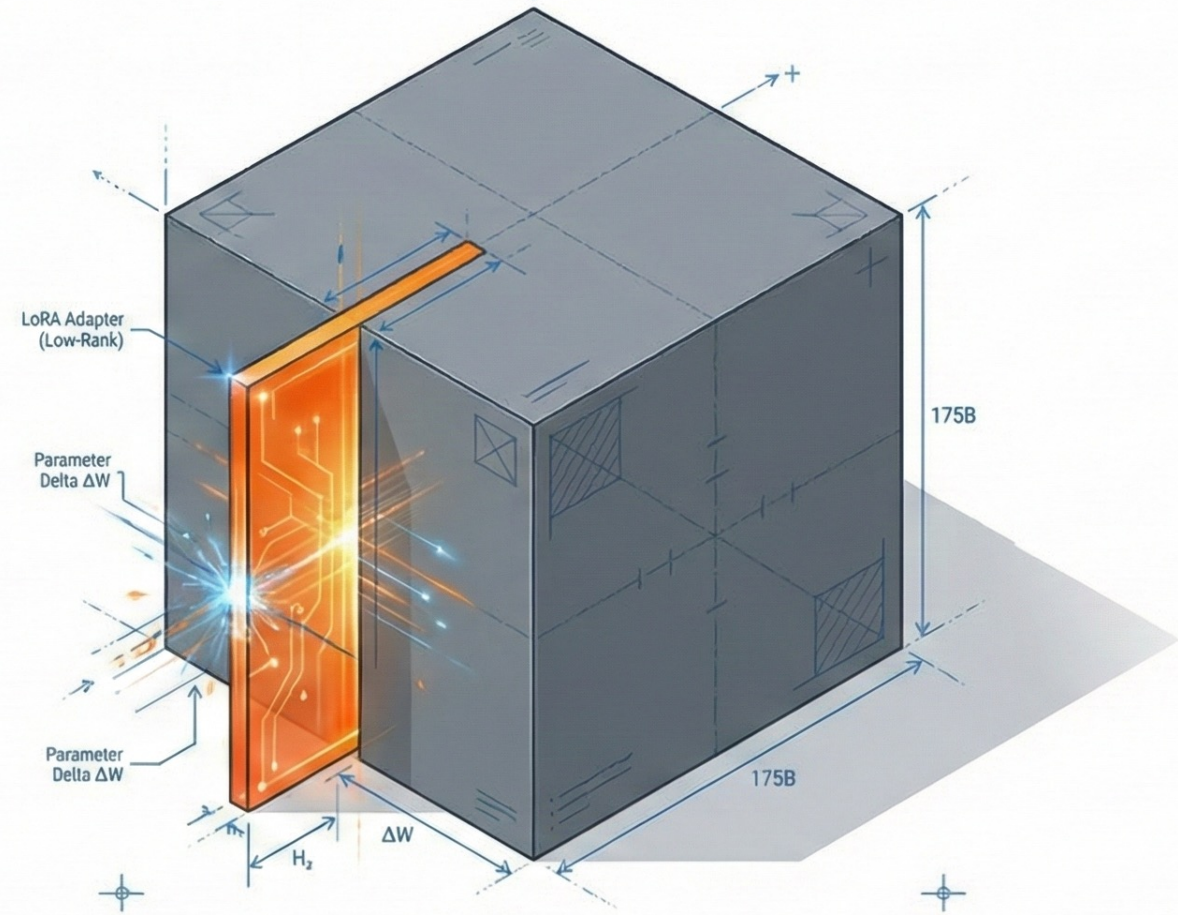
The "New Car for Every Errand" Analogy

Like buying a completely separate car for every single type of small task.

FULL FINE-TUNING PROCESS
(Adaptation for Specific Tasks)

The current approach of full fine-tuning creates unsustainable costs and infrastructure demands, hindering widespread adoption and scaling of large language models for diverse applications.

LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS



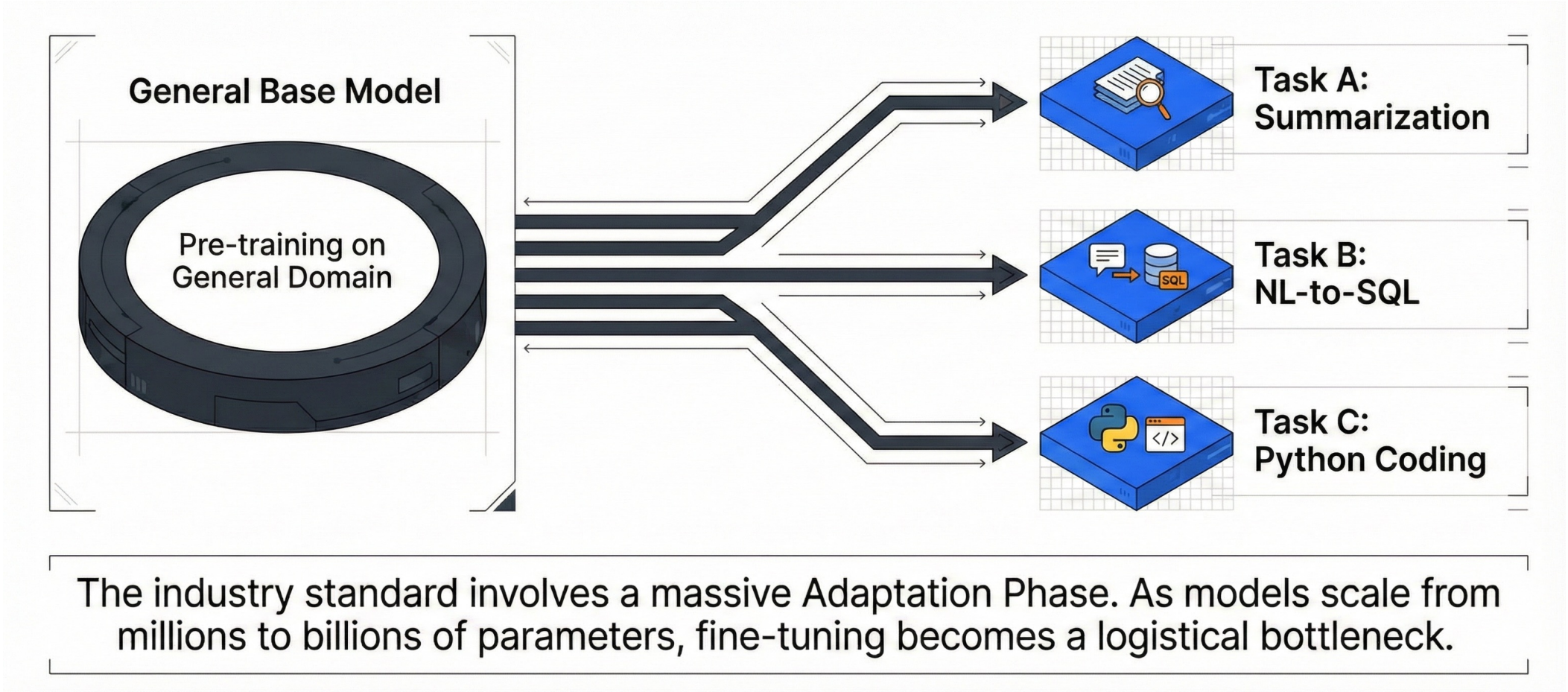
10,000x
Fewer Parameters

3x
Reduced Memory

0ms
Added Latency

Based on research by Hu, Shen, Wallis, Allen-Zhu, Li, Wang, Wang, & Chen (Microsoft Corporation)

The Paradigm Shift: Pre-train Once, Adapt Everywhere

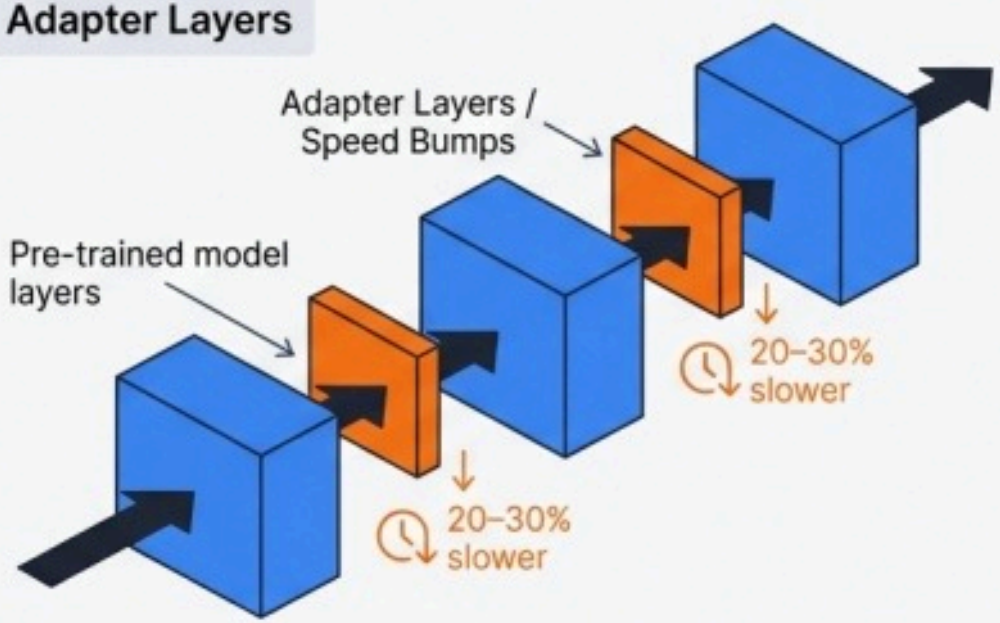


The industry standard involves a massive Adaptation Phase. As models scale from millions to billions of parameters, fine-tuning becomes a logistical bottleneck.

Existing Solutions Trade Performance for Efficiency

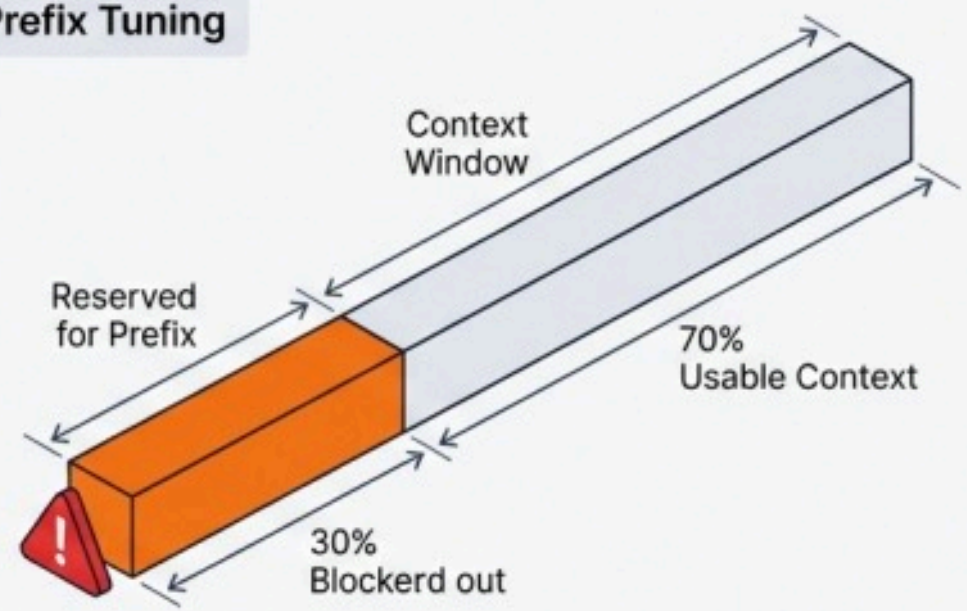
We need a method that is efficient *without* sacrificing inference speed or context memory.

Adapter Layers



The Trade-off: Inference Latency. Adapters insert extra sequential steps between layers. This increases response time by 20–30% in real-time applications.

Prefix Tuning



The Trade-off: Context Loss. Prefix tuning consumes valuable space in the context window. It reduces the memory available for user prompts and is notoriously unstable to train.

Conclusion: We need a method that is efficient *without* sacrificing inference speed or context memory.

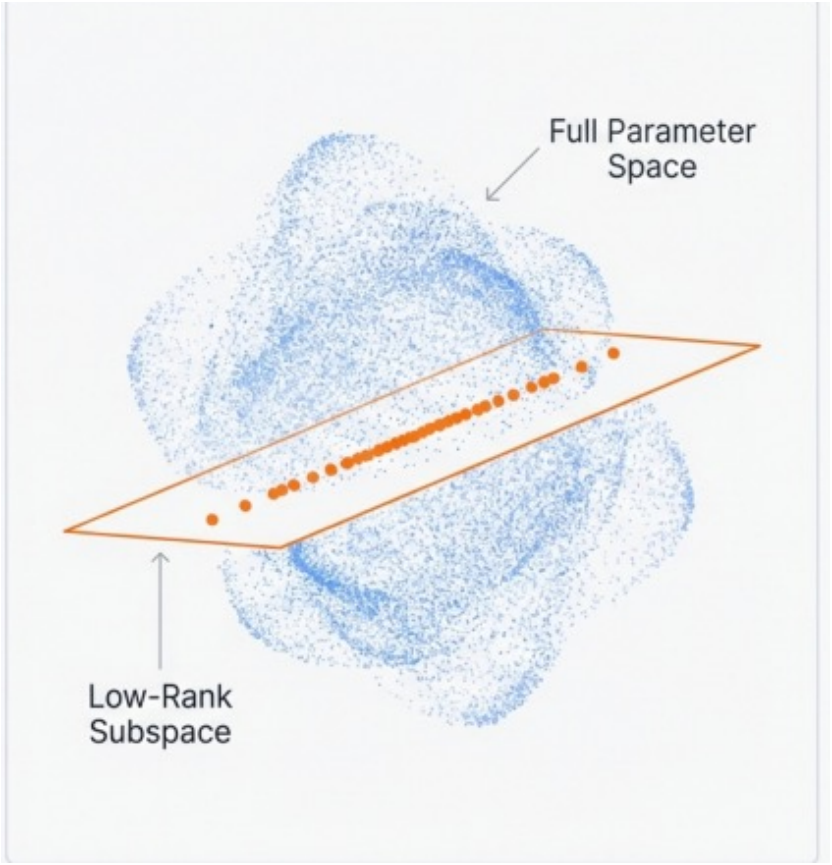
The Hypothesis: Adaptation Has a Low Intrinsic Rank

The Insight

Large pre-trained models are over-parameterized. While their weights are full-rank, **the updates needed for task adaptation lie in a low-dimensional, low-rank subspace**.

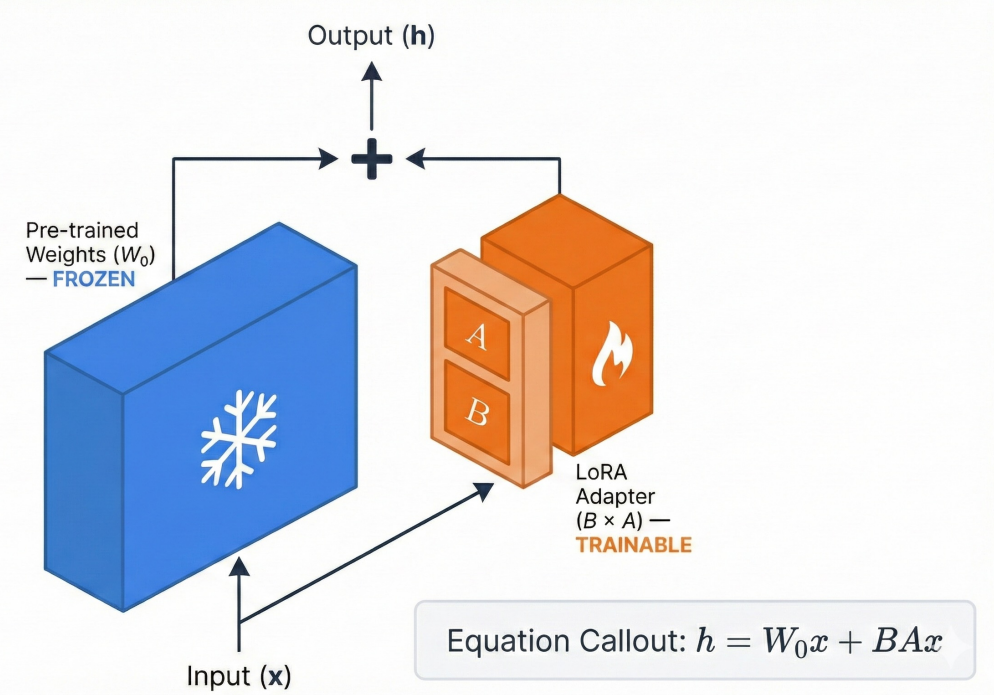
The Implication

Instead of updating all 175B parameters, we **freeze the original weights** and train only small low-rank update matrices, enabling efficient and stable adaptation in a tiny parameter space.



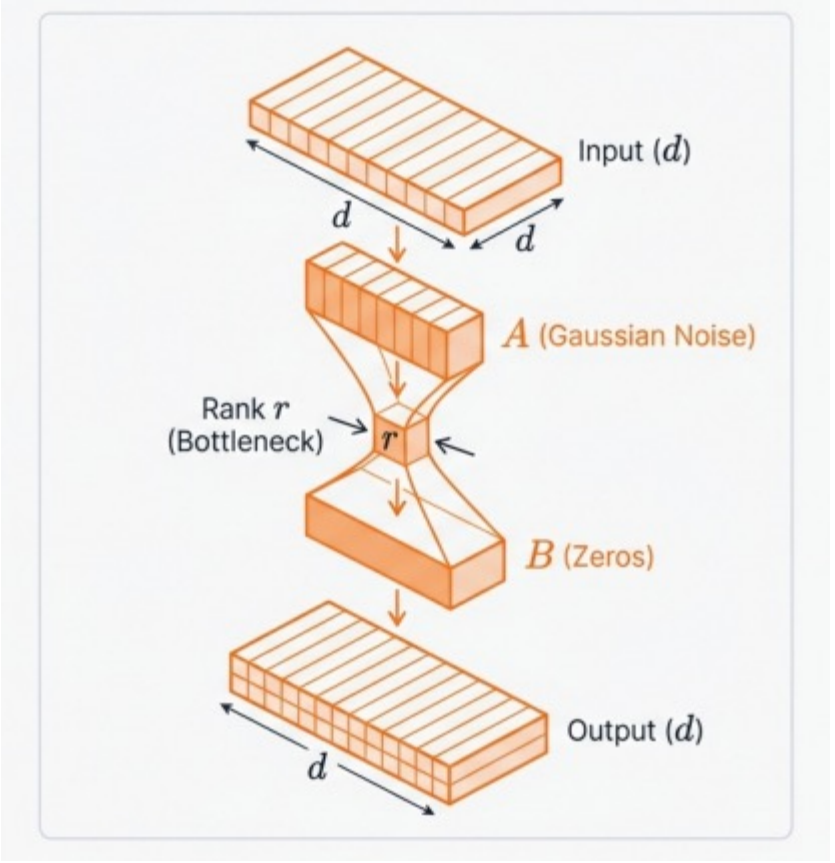
The LoRA Architecture: Bypassing the Bulk

- **Freeze:** We lock the **175B parameters** of the pre-trained model (**W_0**).
- **Inject:** We add trainable **rank-decomposition matrices** (**A** and **B**) as a **side path**.
- **Together:** They are added **in parallel** to (**W_0**).
- **Optimize:** During training, we **only update the orange path** (**A** and **B**).



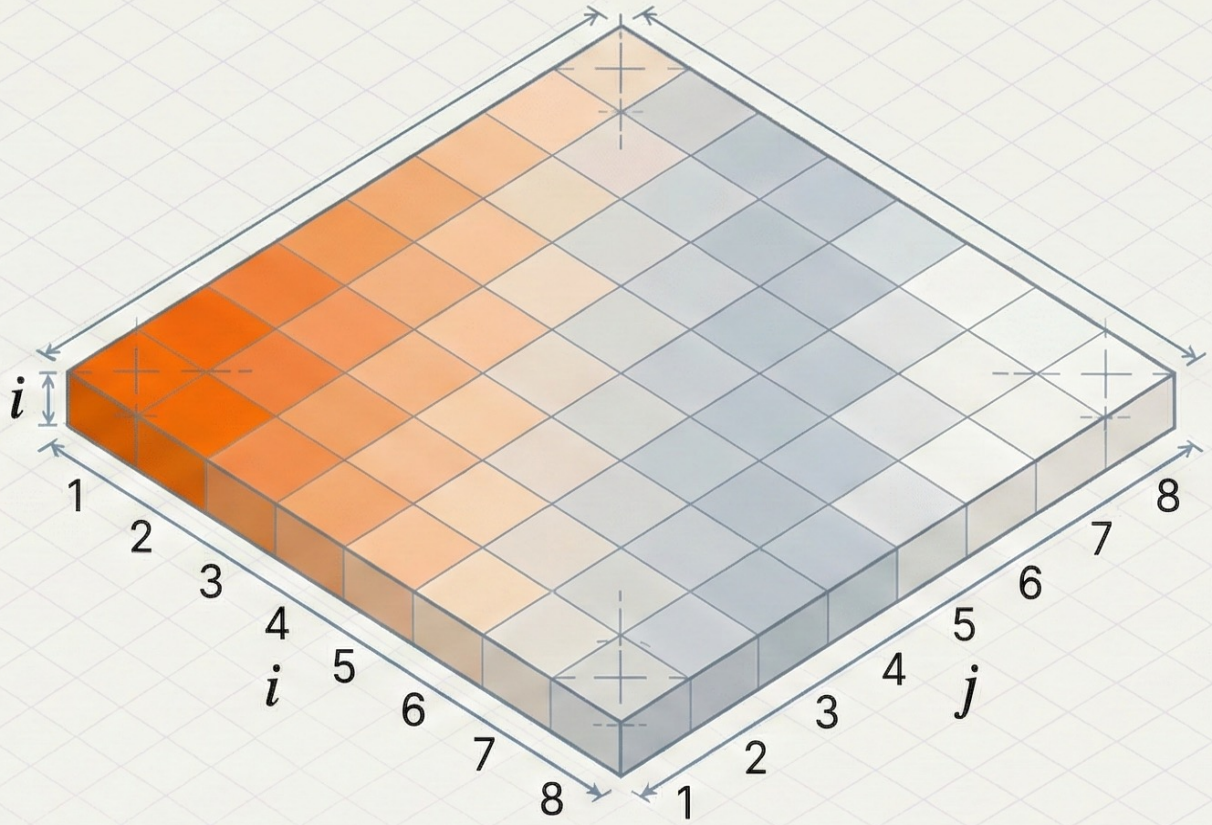
Under the Hood: Matrix Decomposition and Initialization

- **Matrix A:** Initialized with random Gaussian noise.
- **Matrix B:** Initialized with zeros.
- **The Safety Mechanism:** Because **B = 0**, the product **BA = 0** at the start of training. This ensures the model behaves **exactly like the original pre-trained model** at step zero, preventing the instability often seen when introducing new layers.
- **The Math (Forward Pass):**
$h = W_0x + \Delta W x = W_0x + BAx$
- **Dimensions:**
  - **W_0:** d × k (Full Rank, e.g., 12,288)
  - **B:** d × r (Low Rank)
  - **A:** r × k (Low Rank)
- **Rank Constraint:**
- **r < d** (e.g., r = 1 or 2)



How Low Can We Go? The Rank Analysis

Rank (r)	Trainable Params	WikiSQL Accuracy
$r = 1$	4.7M	**73.4%**
$r = 2$	9.4M	73.3%
$r = 64$	301M	73.5%



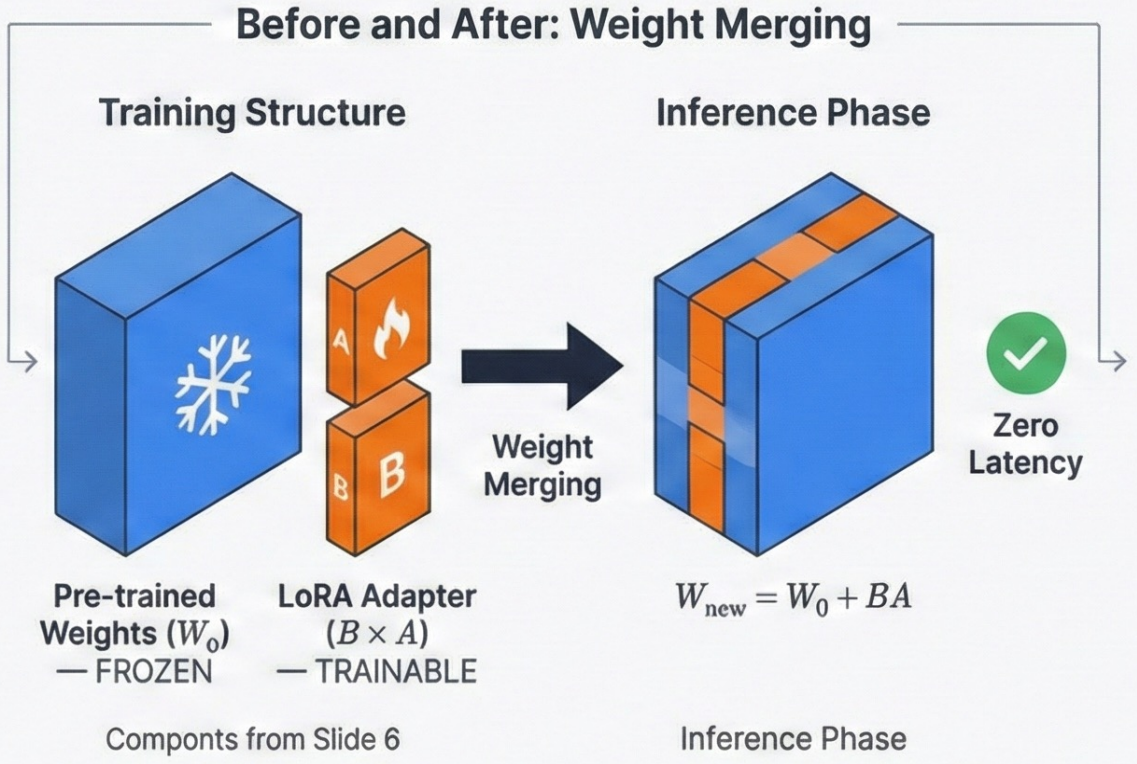
Subspace Similarity between $r = 8$ and $r = 64$.

A rank as small as 1 is often sufficient. Increasing rank **yields diminishing returns** because the intrinsic dimension of the task is tiny.

The Killer Feature: Zero-Latency Inference

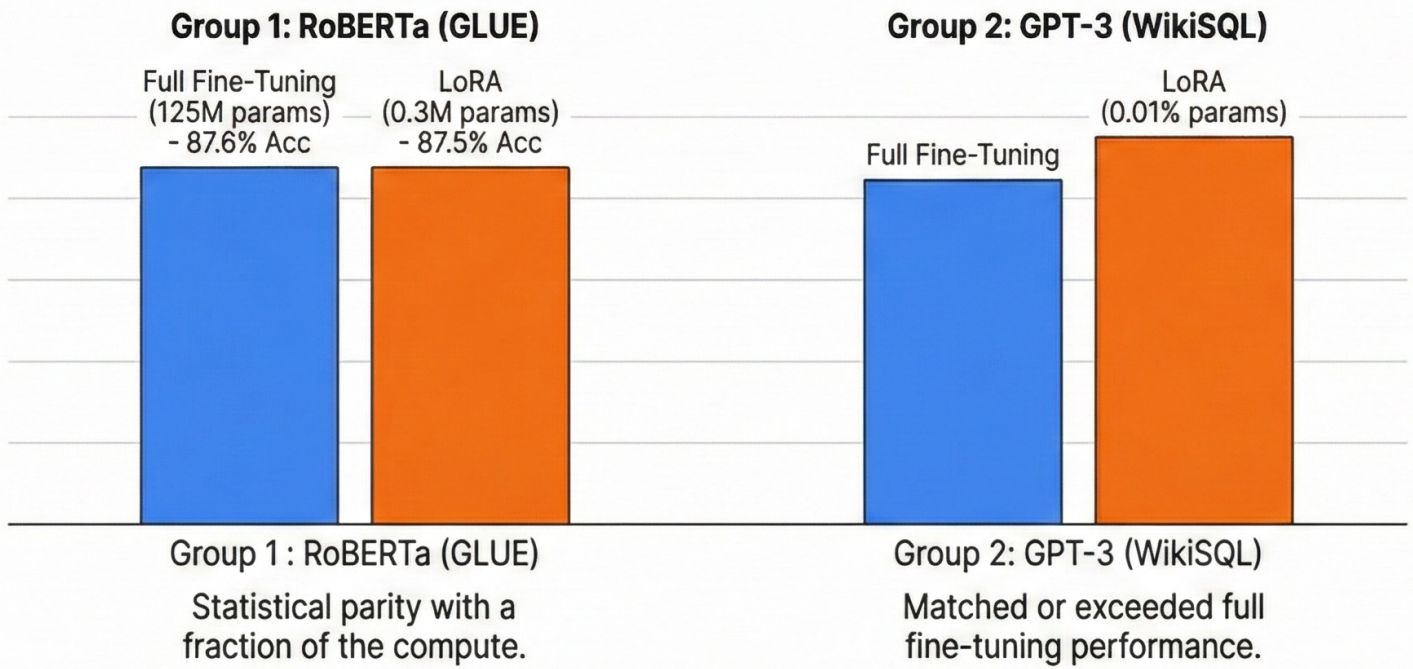
Because the LoRA update is linear (BAx), we can mathematically add the learned matrices directly into the original weights.

- The final deployed model has the exact same architecture as the original.
- **Zero Speed Bumps**: Unlike Adapter layers, there are no extra steps during inference.
- **Full Speed**: The model runs at the native speed of the base model.



Validation: Efficiency Without Compromise

Takeaway: The “Low Rank” hypothesis holds true. We can achieve state-of-the-art results by training less than 1% of the model parameters.

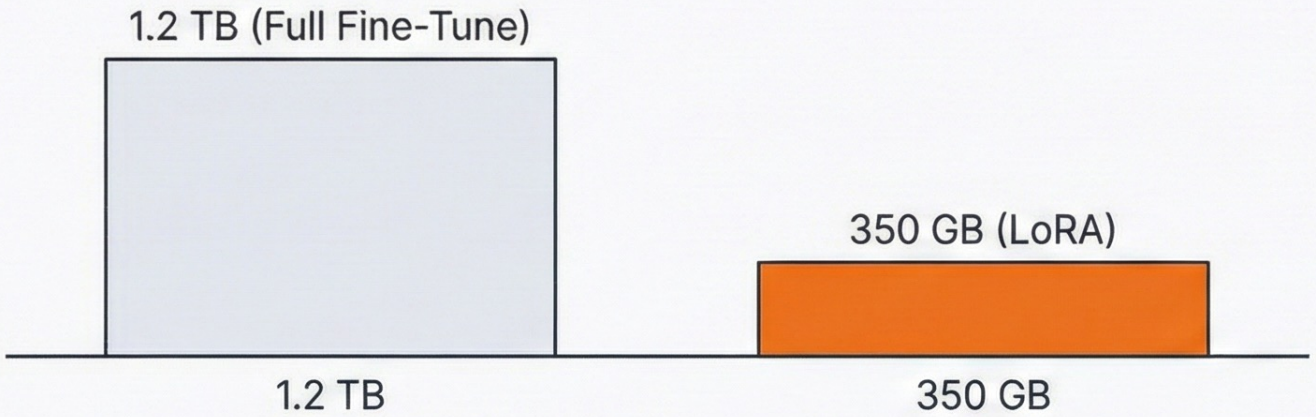


Technical Context

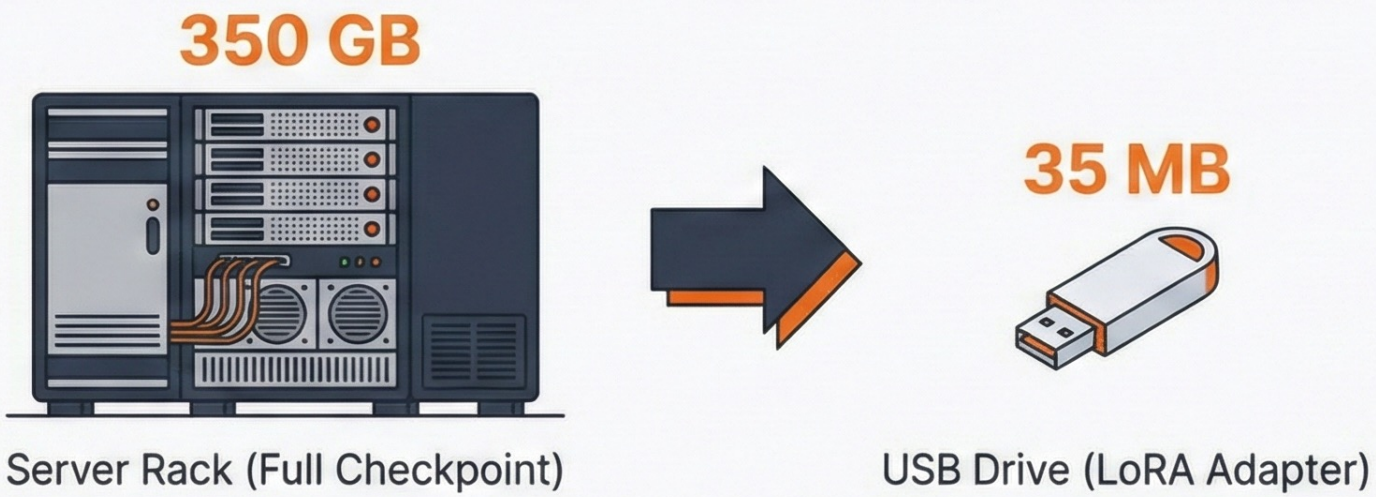
- **The “Low Rank” Hypothesis:** Foundational models possess an intrinsic low-dimensional structure.
- **Parameter Efficiency:** LoRA significantly reduces the number of trainable parameters without sacrificing model capacity.
- **Implication:** Enables fine-tuning on consumer-grade hardware and rapid deployment across diverse tasks.
- **Note on Accuracy:** Minor differences in accuracy are often statistically insignificant compared to the computational gains.

The Efficiency ROI: Doing More with Less

VRAM Reduced by 2/3rds. Storage Reduced by 10,000x.



VRAM: Training Memory Footprint



Technical Impact

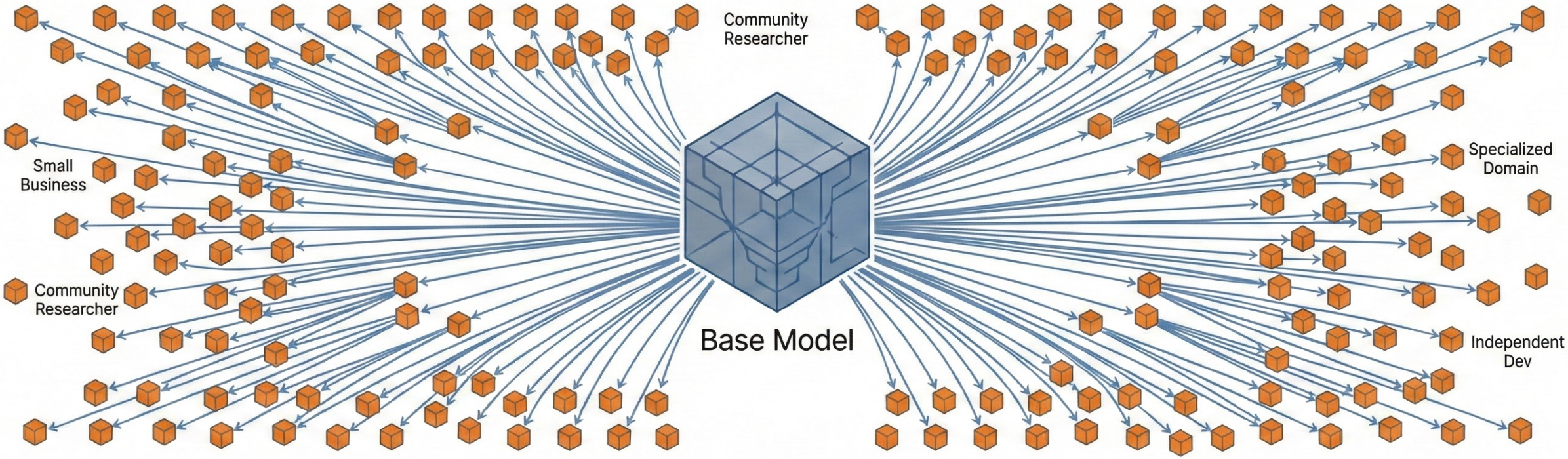
- **The 10,000x Factor**

  LoRA reduces the file size for a task-specific checkpoint by up to 10,000 times (350GB → 35MB).

- **Operational Impact**

  This enables “Model-as-a-Service” where switching from a Legal model to a Medical model involves loading a tiny 35MB file rather than swapping a massive backend.

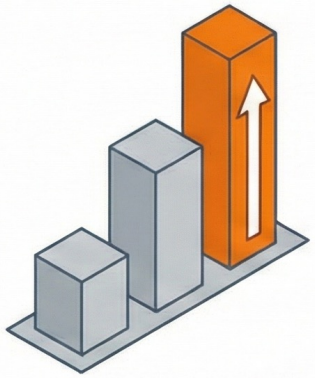
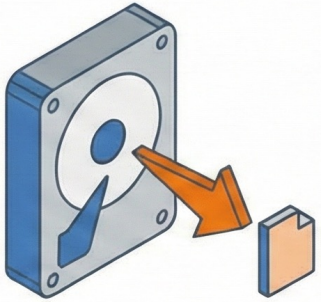
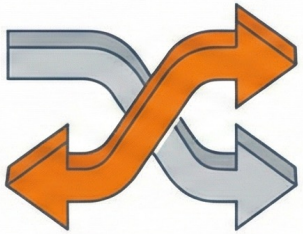
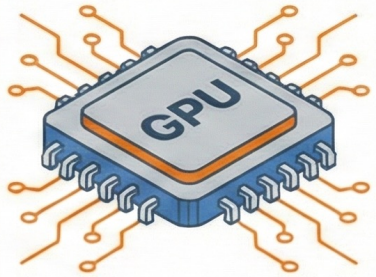
Democratizing LLM Adaptation



LoRA transforms fine-tuning from a privilege of the few to a tool for the many. By proving that model adaptation resides in a low-rank subspace, we solve the bottleneck for the Generative AI era.

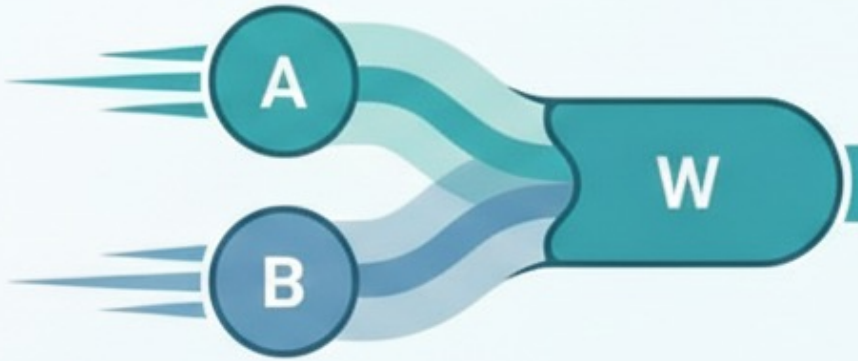
Resources & Implementation: `> pip install loralib  > github.com/microsoft/LoRA`
Available for: RoBERTa, DeBERTa, GPT-2, GPT-3

Summary of Strategic Advantages

Model Quality	Inference	Storage	Task Switching	Hardware
		**350GB** checkpoint  **35MB** file		
Matches or exceeds Full Fine-Tuning. Outperforms Adapters/Prefix.	Zero added latency via weight merging.	**35MB** file (10,000x reduction)	Instant hot-swapping of tasks in memory.	Training possible on consumer hardware (3x less VRAM).

The LoRA Weight Merging Trade-off: Latency vs. Multi-Tasking

Merged Weights (Low Latency)



Zero Inference Latency

Absorbing A and B into W eliminates extra processing time during inference.

Restricted Multi-Task Batching

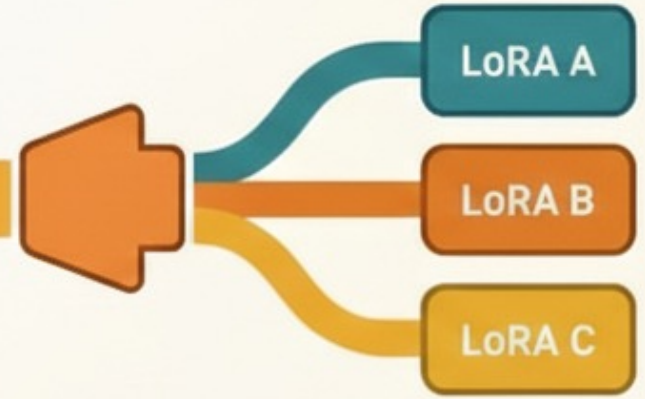
It is not straightforward to batch different tasks in a single forward pass.

Merged Weights Strategy Outcomes

Inference Speed
Optimized (Fastest)

Batching Capability
Single task focus

Non-Merged Weights (High Flexibility)



Dynamic Module Selection

Dynamically choose specific LoRA modules for different samples within one batch.

Non-Merged Weights Strategy Outcomes

Inference Speed
Higher Latency

Batching Capability
Multi-task flexibility

Latency-Tolerant Scenarios

Best suited for environments where immediate response speed is not critical.

Thank You