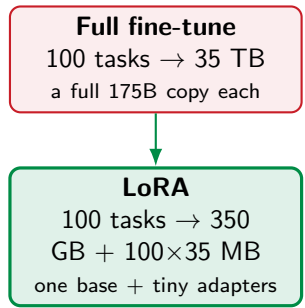
One base model, a hundred tasks

We now know *how* to adapt an LLM – SFT, DPO, GRPO. But full fine-tuning has a brutal cost at scale:

- Fine-tuning GPT-3 175B needs **1.2 TB** of VRAM.
- Each task's checkpoint is **350 GB**.
- 100 fine-tuned models $\approx$ **35 TB** to store.

LoRA: freeze the pretrained weights, and learn a tiny *low-rank* update instead. Same quality, **10,000**$\times$ fewer trainable parameters, **35 MB** per task, and *zero* extra inference latency.



Outline

Why full fine-tuning does not scale

And why existing efficient methods were not good enough.

Full fine-tuning: as big as the original

Fine-tuning learns a task-specific increment $\Delta\Phi$ with $|\Delta\Phi| = |\Phi_0|$ – every parameter moves:

$$\max_{\Phi} \sum_{(x,y)} \sum_{t} \log P_{\Phi}\left(y_t \mid x, y_{<t}\right)$$

- For GPT-2 / BERT this is an inconvenience. For GPT-3 175B it is a **deployment blocker**: a separate 175B model per task.
- You must store and *serve* a full-size copy for every downstream use.

Goal: encode the update with a *tiny* parameter set Θ, $|\Theta| \ll |\Phi_0|$ – as small as **0.01%** of the model – without giving up quality or adding latency.

Existing PEFT methods: not good enough

Adapter layers

- Insert small trainable layers *in sequence* inside each block.
- Few parameters, but the extra depth is on the critical path.
- **Adds inference latency** – worst for online, batch-1 serving (up to **+30%**).

Prefix / prompt tuning

- Prepend trainable "virtual tokens" to the input.
- **Hard to optimize**; performance is non-monotonic in # params.
- Prefix tokens **eat the sequence length** available for the actual task.

Both families often *fail to match* full fine-tuning – a real efficiency-vs-quality trade-off. LoRA aims to remove that trade-off entirely.

The key hypothesis

The *update* during adaptation has a low intrinsic rank.

From intrinsic dimension to intrinsic rank

Prior finding: over-parameterized pretrained models live on a low *intrinsic dimension* – they can be fine-tuned in a tiny random subspace and still learn.

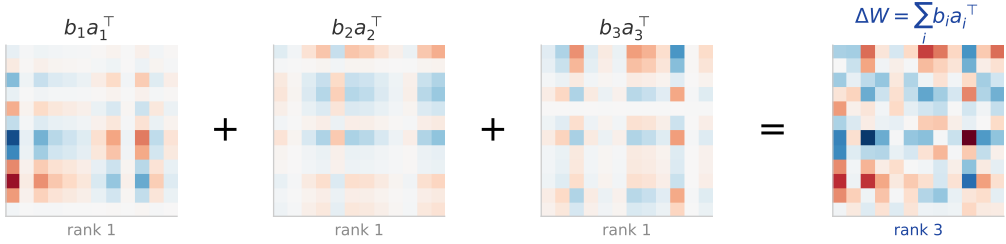
LoRA's hypothesis: the *change in weights* ΔW during adaptation also has a low *"intrinsic rank."* So even though W_0 is, say, $12{,}288 \times 12{,}288$, the update ΔW can be captured by a rank-r factorization with r as small as 1 or 2.

If ΔW is really low-rank, we should not need to learn a full $d \times k$ matrix – just its two thin factors.

What "low rank" actually means

A rank-r matrix is not exotic: it is a **sum of r rank-1 blocks**, $\Delta W = \sum_{i=1}^{r} b_i a_i^\top$ (each $b_i a_i^\top$ one outer product).

A rank-r update = a sum of r rank-1 blocks $\Rightarrow$ store the thin factors $B\,(d \times r)$, $A\,(r \times k)$, not the full grid



So storing ΔW costs the two thin factors $B\,(d \times r)$ and $A\,(r \times k)$ – $r(d+k)$ numbers – not the full $d \times k$ grid. Small r, big saving.

The method

Freeze W_0; learn $\Delta W = BA$ in parallel.

The reparameterization

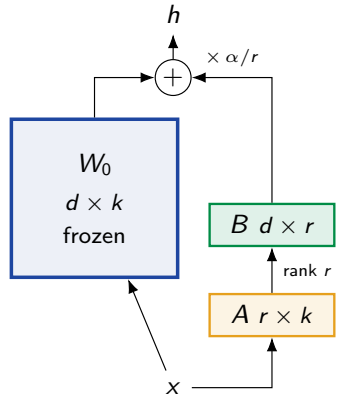
Replace the update with a low-rank product:

$$W_0 + \Delta W = W_0 + BA$$

with $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$ and $r \ll \min(d, k)$. Forward pass (with scaling α/r):

$$\boxed{h = W_0 x + \frac{\alpha}{r} BAx}$$

- W_0 **frozen** (no gradients); train only A, B.
- Init $A \sim \mathcal{N}$, $B = 0$ so $\Delta W = 0$ at the start – training begins exactly at the pretrained model.
- α acts like a learning-rate knob; set once, not tuned.



frozen W_0 and trainable BA run *in parallel*.

Why it is (almost) free

Parameter savings. One matrix full update $= d \times k$. LoRA $= r(d + k)$, i.e. $2rd \ll d^2$. Total budget $|\Theta| = 2 \cdot \hat{L} \cdot d_{\text{model}} \cdot r$ over the $\hat{L}$ adapted matrices.

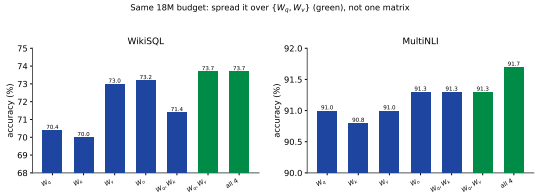
No inference latency. Because BA is added *in parallel* (not in sequence), at deploy time you simply **merge**:

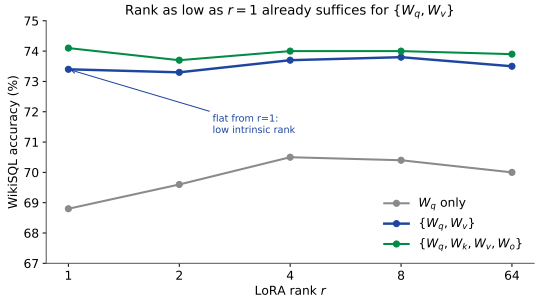
$$W = W_0 + BA$$

and run an ordinary layer – *zero* extra FLOPs. To switch tasks, subtract BA and add a different $B'A'$.

Merging removes latency but you lose **swap-ability**: a merged model can no longer batch *different* tasks in one forward pass. Keep adapters un-merged when you need per-request task selection.

Which weights, and which rank?



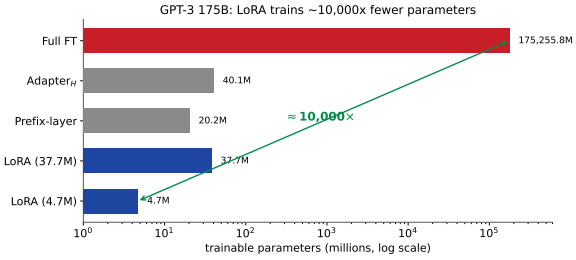


Left: at a fixed budget, spreading over $\{W_q, W_v\}$ beats pouring it all into one matrix. **Right:** accuracy is flat from $r = 1$ for $\{W_q, W_v\}$ – direct evidence of low intrinsic rank. (Only attention W_q, W_k, W_v, W_o were adapted; MLP left frozen.)

Does it match full fine-tuning?

10,000$\times$ fewer parameters, same or better quality.

The headline numbers



GPT-3 175B with LoRA:

- Trainable params **10,000**$\times$ smaller (175B $\rightarrow$ 4.7M).
- VRAM **3**$\times$ smaller (1.2 TB $\rightarrow$ 350 GB).
- Checkpoint **350 GB** $\rightarrow$ **35 MB**.
- **25%** training speedup.

And quality *matches or beats* full fine-tuning.

Quality: on par or better, everywhere

GPT-3 175B	# Trainable	WikiSQL	MNLI-m	SAMSum (R1/R2/RL)
Full fine-tune	175,255.8M	73.8	89.5	52.0 / 28.0 / 44.5
Prefix-layer	20.2M	70.1	89.5	50.8 / 27.3 / 43.5
Adapter$_H$	40.1M	73.2	91.5	53.2 / 29.0 / 45.1
LoRA	**4.7M**	73.4	**91.7**	**53.8 / 29.8 / 45.9**
LoRA	37.7M	**74.0**	91.6	53.4 / 29.2 / 45.1

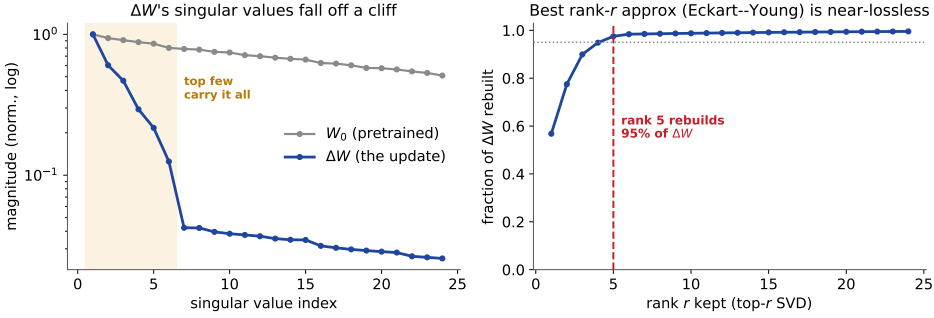
Same story on GLUE (RoBERTa, DeBERTa) and GPT-2 E2E: LoRA reaches full-FT quality with a fraction of a percent of the parameters, and **no added latency**.

Why does such a low rank work?

A look inside ΔW.

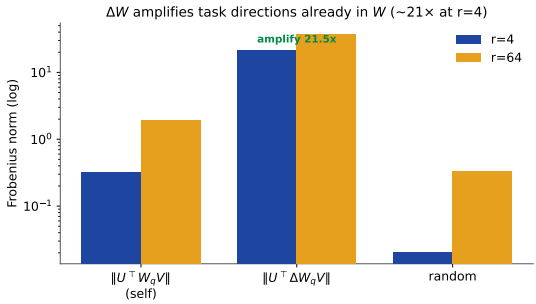
How do we know the update is low-rank?

Any matrix's best rank-r approximation is its top-r SVD (Eckart–Young). So the only question is: how fast do ΔW's singular values decay?



ΔW's energy sits in a **handful** of directions – a tiny r rebuilds almost all of it, while W_0 stays full-rank. *(Illustrative of the SVD mechanism; the paper's evidence is the flat accuracy-vs-rank curve and the amplification analysis, next.)*

ΔW amplifies directions already in W



Project the update ΔW onto the top singular directions of W and compare magnitudes:

- ΔW overlaps directions that are present in W but *not emphasized* during pretraining.
- It **amplifies** them $\sim$**21**$\times$ at r=4 (only $\sim 2\times$ at r=64).
- So adaptation surfaces a few task-specific features already latent in the model – not new ones. That is why a tiny rank is enough.

Recap and practical notes

- LoRA freezes W_0 and learns $\Delta W = BA$ with rank $r \ll \min(d, k)$ – $h = W_0 x + \frac{\alpha}{r} BAx$.
- Init $B = 0$: training starts exactly at the pretrained model.
- **10,000**$\times$ fewer trainable params, **35 MB** checkpoints, **no** inference latency (merge $W = W_0 + BA$).
- Adapt $\{W_q, W_v\}$; rank as small as 1–8 usually suffices.
- It works because ΔW has low intrinsic rank – it amplifies latent task directions rather than inventing new ones.
- **Not** distillation / pruning / quantization: the base model is untouched and full-size; LoRA *adds* a small trainable branch. Rank r is a hyperparameter.

Next: *LIMA* – if fine-tuning is this cheap, how **much** data do we actually need? The answer is surprisingly little.