When to Tune, When to Scale: Full FT vs. LoRA vs. Base Size Under Latency and Accuracy Constraints

Alberto Rodero* Pablo Lobato*

September 2025

1 Motivation & Context

Large Language Models (LLMs) are increasingly deployed under tight *latency*, *cost*, and *governance* constraints. Relying solely on closed API models creates operational risks (variable token pricing, rate limits, model deprecations/renames, slower release cycles, and vendor lock-in), while on-prem or edge deployments demand predictable throughput and VRAM-aware footprints. In this context, teams face a practical choice: *scale up* to a larger base model, or *tune* a smaller one to the task.

Why these techniques. **Full fine-tuning** (NoPeft) offers maximal capacity to adapt behaviors but requires more compute and careful lifecycle management. **Parameter-Efficient Fine-Tuning** (PEFT), notably **LoRA**[1], promises much lower VRAM and faster iteration by training small adapter matrices. However, open questions persist about the efficacy gap vs. full FT, the end-to-end *latency/throughput* implications at inference, and how LoRA *rank* impacts results in practice.

Why these models. We study **Qwen3-0.6B** and **Qwen3-1.7B**, two open-weight checkpoints from the same family. This pairing controls architectural confounds, represents a realistic size jump (0.6B $\rightarrow$ 1.7B) seen in production trade-offs, fits commodity GPUs, and is widely used in on-prem/edge scenarios. It also reflects a common decision boundary: *"Should we tune the small model we can host cheaply, or trade latency to a larger untuned model?"*

Why now. As API prices and policies fluctuate and new frontier releases arrive on slower cadences, organizations increasingly need *controllable*, *auditable*, and *cost-stable* stacks. Evaluating when to *tune* vs. when to *scale*—and whether PEFT is "good enough" under real latency budgets—is therefore critical.

Our aim. We provide actionable guidance on accuracy–latency trade-offs across **full FT** vs. **LoRA (various ranks)** vs. **base size**, using three complementary task families (ARC, OpenMath, SQuAD v2). We also surface the impact of uncontrolled *reasoning* behavior (a realistic deployment confound we later discuss in Sec. 2) to inform **production decisions**.

Key questions.

- **Q1:** Is fine-tuning worth it in efficacy and efficiency?
- **Q2:** How does a fine-tuned 0.6B compare to a larger 1.7B base?
- **Q3:** If VRAM is abundant, is LoRA still worth using?
- **Q4:** Does LoRA rank materially affect outcomes?

*Equal contribution

- **Q5:** Partial vs. full tuning: LoRA vs. NoPeft?
- **Q6:** Cross-task transfer effects?
- **Q7:** If VRAM forces LoRA, is fine-tuning still worth it?

2 Experimental Setup

Models. We compare **Qwen3-0.6B** under two tuning regimes—**full fine-tuning (NoPeft)** and **LoRA** with ranks $r \in \{32, 64, 256, 512, 1024\}$—against **Qwen3-0.6B base** and a larger **Qwen3-1.7B base**. We **do not use quantization** in any reported run; this isolates the effects of the fine-tuning method, rank, and configuration from quantization-related confounds.

Datasets. We evaluate on three task families:

- **ARC[2] (AI2 ARC, 4-way MCQ).** Multiple-choice science questions; we compute macro-F1 across options A/B/C/D.
- **OpenMathInstruct-2[3] (numeric).** Free-form numeric answers; we report average absolute difference (lower is better).
- **SQuAD v2[4] (extractive QA).** Span extraction with unanswerable cases; we report F1 and EM in the evaluator.

Frameworks. We use **HuggingFace Transformers**[5] (`AutoModelForCausalLM`, `AutoTokenizer`), **TRL** (`SFTTrainer`, `SFTConfig`) for supervised fine-tuning, and **PEFT** (`LoraConfig`, `get_peft_model`). Execution is PyTorch (`bfloat16` on CUDA when available).

Supervised fine-tuning (SFT). Each dataset is preprocessed into a simple chat-style format (single user turn with optional context; single assistant completion). Tokenizer uses EOS as pad if unset and a minimal chat template. Global training hyperparameters:

- **Optimizer/schedule (via TRL defaults):** `learning_rate` 1×10^{-5}, `warmup_ratio=0.06`, `weight_decay=0.0`, `max_grad_norm=1.0`.
- **Batching:** `per_device_train_batch_size=1`, `gradient_accumulation_steps=8` (effective batch = 8), `num_train_epochs=1`.
- **Sequence length:** `max_length=1024`, `packing=False`, `assistant_only_loss=False`.
- **Memory:** `gradient_checkpointing=True`.

NoPeft updates all weights. **LoRA** targets attention projections {`q_proj`, `k_proj`, `v_proj`, `o_proj`} with `lora_alpha=16`, `lora_dropout=0.1`; ranks r as above. After SFT, LoRA adapters are merged for evaluation.

Evaluation protocol. We run a deterministic, non-sampling decode (`do_sample=False`; `temperature=None`, `top_p=None`, `top_k=None`; `max_new_tokens=1000`). Each dataset is associated with a concise *system prompt* that standardizes the expected output format using `\boxed{...}` (e.g., `\boxed{C}` for ARC; exact span for SQuAD; numeric for OpenMath). Latency is measured as wall-clock time of a full `generate()` call per sample. We also record output token counts via the tokenizer.

Metrics.

- **ARC:** macro-F1 across A/B/C/D from a 4-class confusion matrix; we additionally track accuracy.
- **OpenMathInstruct-2:** average absolute difference between predicted and gold numeric answers (tolerance 10^{-4}); we also report accuracy (within tolerance).
- **SQuAD v2:** EM and F1 with standard normalization (lowercasing, punctuation/article removal), plus AvNA and No-Answer accuracy; we report F1 in the main table.
- **Latency:** per-sample seconds and aggregates (mean/median/std).

Runtime, decoding & confounds. We place models via `device_map` (GPU when available) and use `bfloat16` on CUDA. Decoding is deterministic (no sampling). *Reasoning-like behavior is intentionally left unlocked at inference time*: we do not enforce a specific "reasoning mode" so that differences reflect the **effect of fine-tuning itself** on *when* the model chooses to reason and *how much* it reasons (versus responding directly). This design helps isolate tuning-induced shifts in reasoning propensity, but it also introduces variance in latency and accuracy. Additional confounds: single-seed/shot artifacts may exist; measured latency depends on this pipeline/hardware; and we did not run calibration/temperature sweeps.

Scope. This study focuses strictly on **full SFT (NoPeft)** vs. **LoRA**. We do not report quantized-training variants (e.g., QLoRA) or alternative adapters (DoRA/VeRA), nor do we sweep decoding temperatures or instruction-following prompts. These choices are deliberate to isolate the *tuning method vs. base size* question.

3 Full Results Table

Notes: "Math AbsDiff↓" is lower=better. Latencies are seconds.

Table 1: Full results across tasks. Lower is better for Math AbsDiff. Latency in seconds. Best *efficacy* and *efficiency* (latency) within each training block are **bolded**.

			ARC		OpenMath		SQuAD v2	
Base	**PEFT**	r	**F1 ↑**	**Lat ↓**	**AbsDiff ↓**	**Lat ↓**	**F1 ↑**	**Lat ↓**
Trained on ARC								
Qwen3-0.6B	LoRA	1024	0.4921	0.1557	**22,871**	0.4374	8.59	**0.1939**
Qwen3-0.6B	LoRA	512	0.4861	0.1570	**22,871**	0.4312	8.59	0.1955
Qwen3-0.6B	LoRA	256	0.4921	0.1549	**22,871**	0.4191	8.59	0.1948
Qwen3-0.6B	LoRA	64	**0.4937**	0.1601	**22,871**	**0.1811**	8.09	0.1952
Qwen3-0.6B	LoRA	32	0.4880	0.1595	**22,871**	0.4439	8.59	0.1958
Qwen3-0.6B	NoPeft	–	0.4905	**0.0803**	23,108	0.2137	**18.89**	0.2003
Trained on OpenMath								
Qwen3-0.6B	LoRA	1024	0.4990	0.9257	23,919	1.5384	**8.48**	0.2091
Qwen3-0.6B	LoRA	256	0.5031	0.8982	23,655	1.5016	**8.48**	**0.1934**
Qwen3-0.6B	LoRA	32	0.5095	0.8702	23,919	1.5776	**8.48**	0.1963
Qwen3-0.6B	NoPeft	–	**0.5171**	**0.0605**	**16,540**	**0.0482**	7.40	0.2439
Trained on SQuAD v2								
Qwen3-0.6B	LoRA	1024	**0.5024**	0.3178	23,647	2.1285	9.59	0.1951
Qwen3-0.6B	LoRA	256	**0.5024**	0.3116	23,647	2.0291	9.59	**0.1908**
Qwen3-0.6B	LoRA	32	**0.5024**	0.3095	23,523	1.9676	9.59	0.1950
Qwen3-0.6B	NoPeft	–	0.4542	**0.1903**	**22,997**	**0.2203**	**27.95**	0.2202
Base models (no fine-tuning)								
Qwen3-0.6B	Base	–	0.4932	**1.1397**	24,834	**6.0139**	10.07	**0.2274**
Qwen3-1.7B	Base	–	**0.7986**	3.4025	**742**	12.5197	**30.36**	0.2837

4 Results by Question & Interpretation

Q1. Is fine-tuning worth it (efficacy & efficiency)?

Yes, when aligned to the target task, fine-tuning yields large quality gains and often lower latency.

- **OpenMath SFT (NoPeft)**: Math abs diff improves by **33.4%** vs 0.6B_base; ARC macro-F1 rises **+4.8%**; SQuAD F1 drops **–26.5%**. Latency massively drops on ARC (**–94.7%**) and Math (**–99.2%**), small increase on SQuAD (**+7.3%**).
- **SQuAD SFT (NoPeft)**: SQuAD F1 jumps **+177.6%**; ARC macro-F1 dips **–7.9%**; Math improves mildly (**+7.4%** abs-diff reduction). Latency generally improves (ARC **–83.3%**, Math **–96.3%**, SQuAD **–3.2%**).

Why? Full-task alignment amplifies the relevant capabilities and stabilizes decoding behavior; our pipeline also appears to run tuned checkpoints more efficiently.

Q2. Fine-tuned 0.6B vs. larger 1.7B base?

The 1.7B base dominates on quality but is slower. ARC macro-F1 +61.9%, Math abs diff ∼97% better, and SQuAD F1 +201.5% vs 0.6B_base. Latency worsens markedly (ARC +198.5%, Math +108.2%, SQuAD +24.8%). **If latency/throughput matters, 0.6B + targeted SFT is the speed/price sweet spot.**

Q3. If VRAM is not a problem, should we still use LoRA?

No. Prefer full fine-tuning. In these runs, **NoPeft is both more accurate and often faster** than LoRA (Table 2). The only place where NoPeft is slightly slower is SQuAD (+0.029 s), but it delivers a **+18.36** absolute F1 jump over the best LoRA.

Table 2: Best LoRA vs. NoPeft by task. For OpenMath, lower is better (AbsDiff↓). Latency in seconds. Within each task, best *score* and best *latency* are **bolded**.

			LoRA (best)		**NoPeft (best)**	
Task	**Base**	**LoRA r**	**Score**	**Lat ↓**	**Score**	**Lat ↓**
ARC (F1 ↑)	Qwen3-0.6B	64	**0.4937**	0.1601	0.4905	**0.0803**
OpenMath (AbsDiff ↓)	Qwen3-0.6B	256	23,655	1.5016	**16,540**	**0.0482**
SQuAD v2 (F1 ↑)	Qwen3-0.6B	1024	9.59	**0.1951**	**27.95**	0.2202

Q4. Does LoRA rank matter?

Little to no consistent improvement from using higher ranks. Correlation between rank and score (LoRA-only subsets): **ARC F1:** $\rho = 0.021$; **OpenMath abs diff:** $\rho = 0.301$ (higher rank slightly worse); **SQuAD F1:** constant across ranks (no signal). Rank vs. latency shows weak-to-moderate links (e.g., ARC $\rho = -0.646$ hints at slightly lower latency at higher ranks), but the absolute differences are tiny.

Table 3: Correlation between LoRA rank and performance/latency (LoRA-only).

Task	Corr(rank, score)	Corr(rank, latency)
ARC (F1)	0.021	-0.646
OpenMath (AbsDiff↓)	0.301	-0.233
SQuAD (F1)	N/A	0.321

Why might rank not matter much? With limited data or strongly structured tasks, the low-rank subspace often captures the critical adaptations; beyond a point, extra capacity (higher rank) faces diminishing returns and optimization noise. Also, our decoding and "reasoning" variability likely masks small rank effects.

Q5. LoRA (partial) vs. NoPeft (full) training?

Full fine-tuning wins decisively.

- **OpenMath:** NoPeft improves error by **7,115** absolute over the best LoRA and is **∼1.45 s** faster.
- **SQuAD v2:** NoPeft improves F1 by **+18.36** with only a **+0.029 s** latency penalty.
- **ARC:** NoPeft is **∼2× faster** than the best LoRA, with essentially tied F1.

Q6. Cross-task transfer?

Catastrophic forgetting (CF). CF is the degradation of previously acquired capabilities when fine-tuning on a new task. It is distinct from benign *negative transfer* (where a skill conflicts but is still partly retained). In our single-task SFT runs, we observe patterns consistent with CF on some capabilities:

- **OpenMath SFT (NoPeft) →** *SQuAD:* **−26.5%** F1 indicates a substantial loss of extractive-span behavior, beyond mild interference. In contrast, *ARC* improves **+4.8%**, suggesting numeric SFT does not erase MCQ.

- **SQuAD SFT (NoPeft)** → *ARC:* **−7.9%** (moderate), while *Math* slightly improves **+7.4%**. This looks more like cross-task interference than severe CF.
- **ARC SFT (NoPeft)** → *SQuAD:* **+87.6%** (surprisingly positive), and *Math:* **+7.0%**; *ARC* self-performance is **−0.5%** (negligible). Here ARC seems to prime general reading/selection skills that transfer to extractive QA.

PEFT vs. full FT. LoRA variants show *smaller cross-task shifts* than NoPeft in several cases (e.g., OpenMath SFT → SQuAD: LoRA ≈ 8.48 F1 vs. NoPeft 7.40; ARC SFT → SQuAD: LoRA ≈ 8–8.59 vs. NoPeft 18.89). This suggests LoRA *can* mitigate forgetting by restricting updates, but it may also *limit positive transfer* when full FT would have helped. In short: **NoPeft magnifies both upside and downside**; LoRA dampens both.

Interpretation. Extractive QA (SQuAD) and numeric reasoning (OpenMath) compete for representational capacity in different subskills (span extraction vs. procedural computation). Tuning toward one can suppress the other. ARC SFT (full) appears to reinforce token-level selection and short-answer discipline, which transfers to SQuAD.

Takeaway. In our setting, **OpenMath→SQuAD degradation under NoPeft** is the clearest CF signal; **SQuAD→ARC** looks like moderate interference; and **ARC→SQuAD** shows strong positive transfer. Choosing *when to use full FT vs. LoRA* is therefore also a choice about *risk/reward on forgetting vs. transfer*.

Q7. If VRAM forces LoRA, is it still worth it?

Yes, but temper expectations. OpenMath LoRA-256 reduces error by **4.8%** vs 0.6B_base (useful), but SQuAD LoRA variants lose **∼4.8%** F1 vs 0.6B_base, and ARC LoRA gains are marginal (**+0.1%**). **Recommendation:** When constrained, **LoRA ≈ 256** is a good default; otherwise prefer NoPeft.

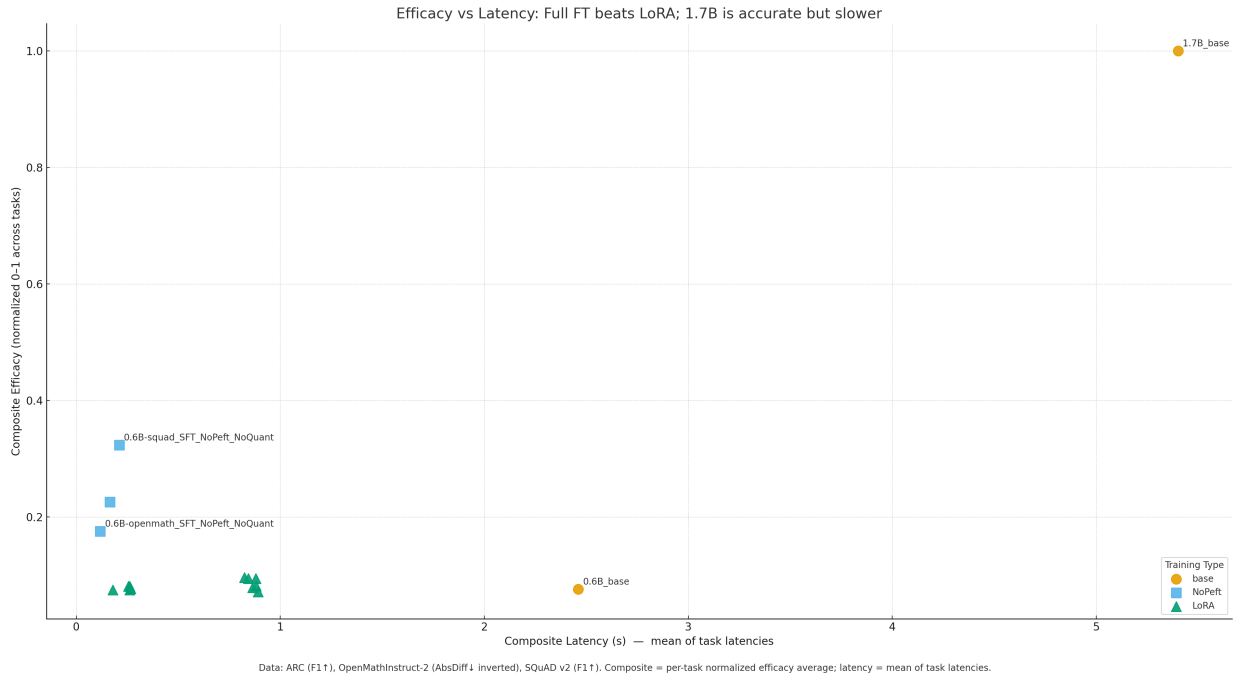

Figure 1: Efficacy vs Latency

5 Conclusions & Takeaways

We summarize two complementary views: (i) technical takeaways that guide modeling choices, and (ii) practical conclusions for solution design.

- **(1) Full fine-tuning beats LoRA on both accuracy and, in these runs, speed.**
- **(2) A larger base model is best on accuracy but slower; 0.6B+SFT is the speed/value sweet spot.**
- **(3) LoRA rank has weak, non-predictable effects; use ∼256 by default when constrained.**
- **(4) Practical decision rule.** *Start small and tune.* If the tuned 0.6B misses the quality bar, try task-specific SFT refinements (data curation, longer context, decoding hygiene). *Only then* consider scaling to 1.7B if accuracy remains the blocker and latency/cost allow.
- **(5) When to prefer LoRA.** Use LoRA if VRAM is tight, you need *multiple domain adapters* with hot-swapping, or you want safer edits that reduce catastrophic forgetting. Keep $r \approx 256$ as a strong default; raise r only with evidence.
- **(6) Forgetting vs. transfer.** Full SFT amplifies both *upside* (positive transfer) and *downside* (catastrophic forgetting). If cross-task robustness matters, use multi-task/rehearsal SFT, per-task adapters with routing/fusion, or light regularization/distillation.
- **(7) Cost/ops note.** Tuned small models reduce serving cost and throughput variance; larger bases raise accuracy ceilings but increase latency and hardware requirements. Choose based on the product's *primary constraint* (SLO latency vs. target quality).

Practical conclusions

What do these results mean at a high level? **Context-anchored tasks (e.g., RAG/extractive QA) are easier to lift with SFT**, because fine-tuning efficiently aligns span selection, formatting, and instruction adherence when relevant context is provided. **Procedural/logic tasks (e.g., math) benefit more from full FT with richer supervision or from scaling the base model (often with tool use)**, as they require deeper representational shifts. In practice, a *small fine-tuned model* can often meet quality targets for RAG-style workloads while lowering latency and cost versus a larger hosted API; reserve **LoRA** for VRAM-constrained setups or rapid domain switching.

References

[1] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang and Weizhu Chen; "LoRA: Low-Rank Adaptation of Large Language Models", *arXiv*, 2021.

[2] Ai2_arc. Disponible en *https://huggingface.co/datasets/allenai/ai2_arc*. Consultado el 1 de septiembre de 2025.

[3] Open Math Instruct-2. Disponible en *https://huggingface.co/datasets/nvidia/OpenMathInstruct-2*. Consultado el 1 de septiembre de 2025.

[4] Squad_v2. Disponible en *https://huggingface.co/datasets/rajpurkar/squad_v2*. Consultado el 1 de septiembre de 2025.

[5] Hugging Face Transformers. Disponible en *https://huggingface.co/docs/transformers/en/index*. Consultado el 4 de septiembre de 2025.