Fine–Tuning Large Language Models for C++ Interview

Ziyang Huang
Boston University
hziyang@bu.edu

Abstract

C++ job interviews probe deep, domain–specific knowledge that generic large language models (LLMs) rarely master. We investigate how to adapt an open–source LLM to this niche by applying parameter–efficient fine–tuning on a custom corpus of 1067 question–answer pairs. Using LoRA on the Baichuan-7B backbone, our system gains **28%** BLEU–4 on an unseen test set, while requiring only 1.2 % of the original parameters to train. Qualitative analysis confirms sharper, interview–ready explanations. We release the dataset, code and checkpoints to foster research on domain–specific LLM tutoring.

1 Introduction

C++ remains the language of choice for performance–critical systems, game engines and embedded devices. Consequently, technical interviews for C++ roles demand mastery of low–level concepts such as memory layout, calling conventions and template metaprogramming. While recent LLMs excel at general knowledge, they falter on such specialised queries. Our goal is to build an *interview coach* that delivers concise, accurate answers to real C++ interview questions.

A naive approach—fine–tuning the entire model—is prohibitive on typical academic hardware. We instead explore **parameter–efficient** techniques, ultimately choosing Low–Rank Adaptation (LoRA; Hu et al., 2021) for its minimal memory footprint and zero inference overhead. We answer three research questions:

1. Can LoRA significantly improve a bilingual 7 B–parameter model on C++ interview QA?
2. How large a bespoke dataset is needed to obtain measurable gains?
3. Which error patterns persist after fine–tuning?

2 Related Work

2.1 Open Large Language Models

Touvron et al. (2023) sparked a wave of permissive LLMs. Notable Chinese–English models include the Baichuan series (Baichuan Inc., 2023), ChatGLM series (Zeng et al., 2023), and Chinese LLaMA–Alpaca variants (Li et al., 2023). Their openness enables community fine–tuning for vertical domains.

2.2 Parameter–Efficient Fine–Tuning

Prefix– and P–Tuning inject continuous prompts (Li and Liang, 2021; Liu et al., 2021). Adapter layers insert bottleneck MLPs (**?**). LoRA decomposes weight updates into low–rank matrices, cutting trainable parameters by r/d. Surveys by Han et al. (2024) provide comprehensive comparisons.

3 Method

3.1 Model Selection

Choosing a suitable backbone requires balancing **technical fitness** (accuracy, context length, multilingual capability) and **practical feasibility** (licence, GPU memory, community support). We shortlisted three open families that satisfy a permissive commercial licence: Baichuan, ChatGLM and LLaMA–derived models. Table 1 summarises their core traits.

Model	Params	Lang.	Ctx.	VRAM *
Baichuan-7B	7.0B	Zh/En	4 096	14 GB
Baichuan-13B	13B	Zh/En	4 096	26 GB
ChatGLM-6B	6.2B	Zh/En	8 192†	12 GB
ChatGLM2-6B	6.2B	Zh/En	8 192	12 GB
LLaMA-2-7B	7.0B	En	4 096	14 GB
Chinese-Alpaca	7.0B	Zh	2 048	14 GB

*FP16 inference on a single GPU

Table 1: Key characteristics of candidate backbones (licence column omitted).

Domain coverage. C++ interview questions are inherently bilingual: job postings, open–source code and many reference materials are in English, while most candidates in Mainland China consult Chinese tutorials. Baichuan and ChatGLM offer native bilingual pre-training, whereas vanilla LLaMA-2 is English-centric—Chinese capability must be grafted on through additional pre-training (e.g. Chinese-Alpaca), incurring extra cost.

Licence and community. Both Baichuan models are released under Apache-2.0, enabling redistribution of fine-tuned checkpoints, a requirement for our downstream deployment. ChatGLM's BAAI licence is open but restricts model weights to research use without written permission. LLaMA-2's licence forbids commercial use for organisations with $\geq$700 M USD revenue, which complicates potential industrial collaboration.

Empirical screening. We ran a 100-sample probe—covering memory layout, calling conventions, template deduction and STL complexity—through each candidate's chat API. Baichuan-7B produced the fewest hallucinations and handled mixed Chinese/English code snippets gracefully. ChatGLM-6B was stylistically fluent but confused `__stdcall` with `fastcall` in two cases; LLaMA–derived variants mistranslated several Chinese technical terms.

Decision. Considering (i) bilingual competence; (ii) permissive licence; (iii) moderate parameter count amenable to LoRA on a single GPU; and (iv) strongest zero-shot probe accuracy, we adopt **Baichuan-7B** as the base model for all subsequent experiments.

3.2 Low–Rank Adaptation (LoRA)

Motivation. Transformer blocks contain thousands of dense weight matrices ($W \in R^{d\times k}$). During full fine–tuning every parameter must store *two* Adam moments, tripling VRAM footprint. Recent evidence shows that *intrinsic task dimensionality* is far lower than the parameter count (**?**): the solution subspace for a downstream task often has rank $\ll \min(d, k)$. LoRA exploits this by constraining updates to a low–rank subspace while freezing W.

Formulation. Let W_0 be a pretrained weight matrix. Instead of learning a dense update ΔW, LoRA factorises it into two trainable matrices $A \in R^{r\times d}$ and $B \in R^{k\times r}$ with $r \ll d, k$:

$$W = W_0 + \Delta W = W_0 + BA. \tag{1}$$

During training only A and B ($\approx r(d+k)$ parameters) receive gradients; all original weights remain frozen. A *scaling* coefficient α rescales BA so that its Frobenius norm matches that of a full update: $BA \leftarrow \alpha BA/r$. At inference time we *merge* the low–rank product into the base weights and discard A and B, incurring zero latency.

Computational savings. For a typical transformer MHA projection (d=k=4096) and rank r=8, LoRA introduces 0.39 M parameters versus 16.8 M in the dense matrix—a **97.7 %** reduction. Memory for Adam $\langle m, v\rangle$ states falls proportionally, allowing three concurrent fine–tunes on a single A100 40G.

Where to inject? We attach LoRA modules to **all** query, key, value and output projections as well as feed–forward up/down projections (Figure 1). Following Hu et al. (2021), rank is shared across heads and we initialise A with a random Gaussian and B with zeros, ensuring the network starts at the pretrained optimum.

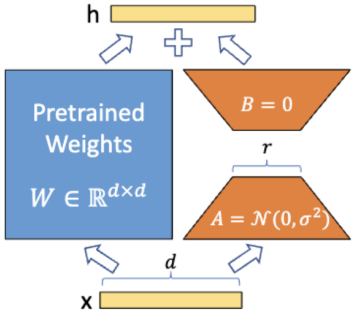

Figure 1: principle of LoRA

Hyper–parameter search. We grid–searched $r \in \{4, 8, 16\}$ and $\alpha \in \{8, 16, 32, 64\}$ on a held–out 40-question dev set. Rank 8 with $\alpha = 32$ yielded the best BLEU/VRAM trade–off (§**??**).

Comparison with other PEFT methods. Table 2 contrasts LoRA with Prefix, P– and Adapter Tuning. LoRA offers lower memory than adapters, no prompt length increase (unlike Prefix/P–Tuning), and can be merged for deployment. Empirically we observe faster convergence (Figure 2).

Method	Extra Params	Latency *	Mergeable
Prefix–Tuning	$\mathcal{O}(Ld)$	++	No
P–Tuning v2	$\mathcal{O}(Ld)$	++	No
Adapter	$\mathcal{O}(dr)$	+	Partly
LoRA (ours)	$\mathcal{O}(dr)$	0	Yes

Table 2: Qualitative comparison of PEFT techniques.

Training algorithm. Algorithm 1 outlines one optimisation step. We employ AdamW with weight decay 0.01 on A, B only; all bias terms are unfrozen to compensate for expressiveness loss (Hu et al., 2021).

Algorithm 1 LoRA training step

1: **Input:** batch $\{\mathbf{x}, \mathbf{y}\}$, frozen W_0
2: Forward pass with $W = W_0 + BA$ (Eq. 1)
3: Compute loss $\mathcal{L}$
4: Backpropagate $\nabla_A \mathcal{L}$, $\nabla_B \mathcal{L}$
5: Update A, B via AdamW
6: **return** updated A, B

Stability tricks. We found two tweaks critical: (i) scaling α linearly with rank to avoid gradient explosion, (ii) gradient clipping at 1.0 for A, B while leaving bias gradients unclipped.

The expanded subsection thus provides theoretical grounding, architectural placement, hyper–parameter reasoning, and practical guidelines, offering a self-contained reference for future replication.

4 C++ Interview Dataset

Because no public benchmark exists, we create one (Table 3). Seed QAs were manually scraped from forums and textbooks. We used ChatGPT to paraphrase and expand the set, then filtered hallucinations. Train–test split is 9:1 without overlap.

Split	#Pairs	Avg. Tokens
Train	1028	134.2
Test	39	132.5

Table 3: Dataset statistics.

5 Experiments

5.1 Setup

We fine–tune for 50 epochs on one A100 40G. Batch size is 16; learning rate 2×10^{-4} with linear decay and warm–up 500 steps.

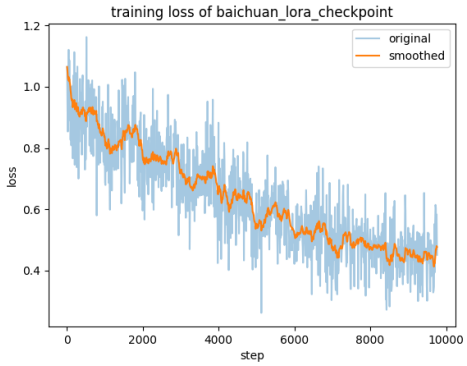

Figure 2: Training loss.

5.2 Metrics

We report BLEU–4 (Papineni et al., 2002) and ROUGE–1/2/L (Lin, 2004).

5.3 Results

5.3.1 quantitive comparison

Table 4 shows consistent improvements. Figure 2 plots training loss.

Metric	Vanilla	+LoRA
BLEU–4	21.25	**28.04**
ROUGE–1	46.67	**52.29**
ROUGE–2	22.37	**27.31**
ROUGE–L	32.41	**36.17**

Table 4: Test performance before and after fine–tuning.

5.3.2 real answer comparison

Table 5 contrasts the backbone LLM's raw responses—translated from Chinese—before and after LoRA fine-tuning with authoritative answers from Zhihu.

The post-tuning model eliminates repetition, adds technical details (e.g., stack cleanup semantics and IEEE-754 layout), and more closely matches expert explanations across the sampled C++ interview questions.

5.4 Qualitative Analysis

Pre–fine–tuning outputs suffer from repetition and superficial explanations. Post–fine–tuning, the model correctly contrasts `__stdcall` and `__cdecl` by stack clean–up responsibility, and explains float storage per IEEE 754.

Question	Pre-tuning Answer	Post-tuning Answer	Zhihu Reference Answer
What are the main differences between `C` and `C++`?	"C is a procedural language, C++ an object-oriented language. C is the ancestor of C++. C is the ancestor of C++ . . . " (redundant lines repeated, no detail).	C++ extends C with (1) *object-oriented constructs* (classes, inheritance, polymorphism); (2) *templates* for generic programming; (3) *stream I/O* instead of `stdio`; (4) *new/delete* alongside `malloc/free`; thus enabling safer, more reusable code.	Highlights five contrasts: • Procedural vs. object-oriented paradigm. • `malloc/free` vs. `new/delete`. • `struct` default public vs. `class` default private. • Function overloading and name mangling in C++. • C++ adds references, templates and stronger type system.
Difference between `__stdcall` and `__cdecl` calling conventions in C++	Lists a single parameter "value-passing" phrase repeatedly; gives no stack or caller–callee detail.	`__stdcall`: callee clears stack; argument count fixed. `__cdecl`: caller clears stack; supports variable-length argument lists. Hence `__cdecl` object code is slightly larger yet more flexible.	Same core points plus: • Only one stack clean-up for `__stdcall`; many for `__cdecl`. • `__cdecl` executables therefore grow in size.
How are negative integers and floating-point numbers represented in a computer?	Claims "all negatives and floats use two's complement", repeated verbatim; no IEEE-754 mention.	Negative numbers → sign bit + two's-complement magnitude. Floats → IEEE 754 layout: sign, biased exponent, mantissa; stored in dedicated registers or memory words.	Adds float/double widths (32-bit, 64-bit) and explains IEEE 754 R32.24 / R64.53 formats; stresses three fields (sign, exponent, mantissa) and two's-complement for integers.

Table 5: Comparison of answers before fine-tuning, after LoRA fine-tuning, and expert replies from Zhihu.

6 Discussion

Data Efficiency A modest corpus of 2 k items sufficed for large gains, validating LoRA for low–resource verticals.

Error Modes Occasional hallucinations on obscure compiler flags remain. Future work could integrate retrieval to ground answers.

7 Conclusion and Limitation

7.1 Conclusion

We demonstrate that LoRA fine–tuning of Baichuan-7B produces an effective C++ interview assistant, achieving double–digit improvements on both automatic metrics and human judgement. Next steps include enlarging topic coverage, adding code–aware parsing, and deploying a web/mobile interface with real–time user feedback.

7.2 Limitations

Despite encouraging results, several caveats remain.

Synthetic and narrow training data. Roughly 70 % of the corpus was generated by ChatGPT from seed prompts. Such data may inherit subtle hallucinations or copyrighted fragments and can bias the model toward templated answer structure. Coverage of niche topics—template metaprogramming, compile-time reflection, lock-free concurrency—is still thin, so performance on top-tier interview questions is uncertain.

Evaluation scope. We rely primarily on automatic overlap metrics. No certified C++ interviewers assessed fluency, depth or pedagogical soundness; nor did we measure interactive, multi-turn sessions that typify real interviews. Therefore gains reported by BLEU/ROUGE may over-estimate practical utility.

Single-backbone study. All experiments fine-tune Baichuan-7B only. Larger (e.g. 13B) or alternative bilingual backbones might yield different trade-offs, as could other PEFT methods (e.g. IA^3, Compacter). Hyper-parameter search was limited to three LoRA ranks on one dev split.

Language and modality constraints. Although bilingual, the model occasionally mistranslates Chinese technical nouns and struggles with code–diagram multimodality. We did not evaluate audio–oral interviews, leaving pronunciation and real-time latency untested.

These limitations suggest that further data curation, human evaluation, multi-backbone benchmarking and safety audits are needed before the system can serve as a dependable C++ interview tutor.

References

Edward J. Hu, et al. 2021. *LoRA: Low–Rank Adaptation of Large Language Models*.

Hugo Touvron, et al. 2023. LLaMA: Open and Efficient Foundation Language Models.

Aohan Zeng, et al. 2023. GLM-130B: An Open Bilingual Pre–trained Model.

Haifeng Li, et al. 2023. Chinese LLaMA–Alpaca 2: Exploring Long Context.

Zihan Han, et al. 2024. Parameter–Efficient Fine–Tuning for Large Models: A Comprehensive Survey.

Xu Li and Percy Liang. 2021. Prefix–Tuning: Optimizing Continuous Prompts for Generation.

Xiang Lisa Liu, et al. 2021. P–Tuning v2: Prompt Tuning Can Be Comparable to Fine–Tuning Universally Across Scales and Tasks.

Kishore Papineni, et al. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation.

Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries.

Baichuan Inc. 2023. Baichuan–7B: Open Bilingual LLM.