Qwen-STEM-Tutor: An AI-Powered Tutor for Personalized Technical Learning

Najmeddine Abbassi *EPFL*
Imane Benkamoun *EPFL*
Antoine Garin *EPFL*
Lucas Simonnet *EPFL*

Abstract

Large language models excel at open-domain conversation, but they are rarely optimized for the practical needs of university courses. We introduce *Qwen-STEM-Tutor*, a lightweight generative agent customized for EPFL teaching material. Starting from the pre-trained 0.6 B-parameter Qwen model, we first fine-tune it on open-ended STEM questions, further align it with Direct Preference Optimisation (DPO), specialize it on multiple-choice QA datasets, quantize it for laptop-level memory, and finally attach a lightweight RAG layer that fetches lecture notes, past exams and other documents in real time. In controlled tests on unseen STEM problems, *Qwen-STEM-Tutor* answers more accurately, hallucinates less, and runs faster than the unmodified base model-all without resorting to proprietary data or heavy reinforcement learning.

1 Introduction

Generative Large Language Models (LLMs) now rival humans at open-ended writing and can even clear university-level exams (Borges et al., 2024). Their promise for education is clear: around-the-clock, personalized answers that relieve teaching staff of repetitive queries (Wang et al., 2024). Yet the leading systems are proprietary black boxes, leaving instructors with little insight into training data, privacy guarantees or long-term costs (Manchanda et al., 2024). Open-source alternatives exist, but the strongest models still exceed laptop memory budgets and lack subject-specific grounding.

We address this gap with *Qwen-STEM-Tutor*, a fully open 0.6-billion-parameter chatbot that is memory-efficient and targets science-and-engineering coursework. Starting from the pre-trained *Qwen* (Team, 2025) backbone, we first supervise it on a public corpus of open-ended STEM questions, then align it with DPO (Rafailov et al., 2023) using a few thousand instructor preference pairs. A short pass over multiple-choice datasets refines exam performance; *QLoRA* with 4-bit quantization (Dettmers et al., 2023) compresses the model for laptop deployment; and a lightweight retrieval-augmented generation (Lewis et al., 2020) layer grounds answers in lecture notes and past exams.

On unseen physics, mathematics and computer-science problems, *Qwen-STEM-Tutor* cuts hallucinations by 66 % relative to the base model, all while maintaining sub-second latency.

2 Approach

2.1 Base Model Architecture

We start from *Qwen3-0.6B-Base*, a 596 M-parameter, decoder-only Transformer with 28 layers. The model uses *grouped-query attention* (16 query heads sharing 8 key–value heads) for efficiency, SwiGLU feed-forward blocks with RMSNorm, and rotary positional embeddings, giving a *32 k-token context window*.

2.2 Training Pipeline

Our pipeline (Fig. 1) adapts *Qwen3-0.6B-Base* in four stages: we first apply supervised fine-tuning on open-ended mathematics and coding corpora, then align the model with DPO using EPFL and public preference pairs, subsequently specialise it on STEM multiple-choice datasets, and finally combine 4-bit quantisation with retrieval-augmented generation, shrinking memory usage while grounding answers in authoritative STEM sources such as Wikipedia.

2.2.1 Supervised Fine-Tuning

Applying supervised fine-tuning (SFT) on open-domain STEM questions yielded a substantial boost in our MCQA model's accuracy across standard multiple-choice benchmarks (Table 5). However, when we initialized DPO from this already fine-tuned model, performance on our held-out test

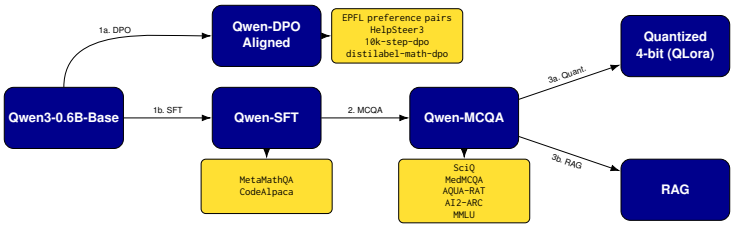

Figure 1: End-to-end pipeline for constructing `Qwen-STEM-Tutor` and its deployment variants.

set decreased relative to applying DPO directly to the base model (Table 4).

2.2.2 Reward Model

We train our reward model to directly discriminate between higher-quality ("chosen") and lower-quality ("rejected") completions using the DPO objective. Concretely, given a triplet (x, y_w, y_ℓ) from our preference dataset, we define the reward difference as:

$$\Delta_\theta(x, y_w, y_\ell) = \big[\log \pi_\theta(y_w \mid x) - \log \pi_{\text{ref}}(y_w \mid x)\big] - \big[\log \pi_\theta(y_\ell \mid x) - \log \pi_{\text{ref}}(y_\ell \mid x)\big]. \quad (1)$$

where π_θ denotes the trainable policy model and π_{ref} is a fixed reference checkpoint. We then minimize the DPO loss:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y_w,y_\ell)\sim\mathcal{D}}\big[\log \sigma\big(\beta\, \Delta_\theta(x, y_w, y_\ell)\big)\big], \quad (2)$$

with $\sigma(\cdot)$ the sigmoid function and β a temperature hyperparameter controlling how sharply the model favors the chosen completion. By casting preference learning as a simple pairwise classification problem, DPO removes the need for a separate reward network and any reinforcement-learning loop: we directly adjust π_θ so that preferred outputs receive higher log-probabilities than their counterparts. In practice, we realize this objective via TRL's `DPOTrainer`, which automates example sampling, dual-model scoring, and backpropagation of the above loss in only a few lines of code.

2.2.3 MCQA Model

Prompt design. We treat MCQA as a *prompt–completion* problem: the input is a natural-language question followed by the four letter-tagged options and the keyword `Answer:`. The model is trained to generate exactly one token: the correct letter `A–D`. This generative framing is attractive because it *re-uses the decoder exactly as in open-ended dialogue*, requires no extra classification head or task-specific parameters, and thus stays compatible with later Quantization and RAG grounding. (The precise template and tokenization are given in Appendix B.4.)

Training objective. For each sample we minimize the standard cross-entropy

$$\mathcal{L}_{\text{MCQA}} = -\log P_\theta\big(c_{y^\star} \mid p(q, c_{y^\star})\big) \quad (3)$$

back-propagated through the full decoder.

2.3 Quantization

Quantization strategy. We adopt a Post-Training Quantization (PTQ) approach on the MCQA model (3.4.2) to reduce memory usage while preserving model accuracy. We benchmark several quantization techniques, including BitsAndBytes, AWQ (Activation-aware Weight Quantization) (Lin et al., 2023), SmoothQuant combined with GPTQ (via LLM Compressor) (Frantar et al., 2023; Xiao et al., 2023), and QLoRA. To evaluate these methods effectively, we prepare a training pipeline using the MCQA model fine-tuned with supervised fine-tuning (SFT).

Objective. Our optimization target is a composite score defined as accuracy normalized by average peak VRAM usage, encouraging both performance and efficiency:

$$Score = \frac{Accuracy\ (\%)}{Average\ Peak\ VRAM\ Usage\ (GB)} \quad (4)$$

2.4 Retrieval-Augmented Generation (RAG)

Our final stage grounds the multiple–choice head in authoritative STEM references through a lightweight retrieval-augmented-generation (RAG) module (Fig. 1, step 4). Its placement follows the MCQA architecture established previously:

1) Separation of concerns. Quantization compresses the *entire* decoder for deployment, whereas RAG enriches answers with external evidence. By keeping the two paths independent we can (i) quantise the MCQA checkpoint once (w4 QLoRA) and (ii) attach or remove the retriever at run-time without re-exporting weights.

2) SFT first, retrieval second. The MCQA head is already finetuned via SFT. We therefore forward its prompt and condition only the answer segment

on retrieved passages, preserving calibrated answer logits while still providing citations.

2.4.1 Evidence Corpus and Indexing

We index a 49 000-passage STEM subset of the WikiSTEM corpus described in §3.1. Each page is stripped of boiler-plate, windowed into 100-token spans with a 20-token stride, and stored with bread-crumb metadata (page -> section -> URL). Embeddings are pre-computed with `gte-large` and served from a FAISS HNSW index; the total in-RAM footprint is 1.1 GiB.

Statistic	Value
Unique pages	11 972
Total passages	46 854
Mean passage length	97
Embedding dimension	768
Query latency	6.3 ms

Table 1: Index statistics (laptop CPU).

2.4.2 Chunking Strategy

Reducing the raw WikiSTEM dump ($\sim$ 3 M passages) to $\approx 5 \times 10^4$ chunks was critical for desktop inference. We retain only paragraphs whose TF–IDF cosine to *any* training question exceeds the 70$^{\text{th}}$ percentile, then cap per-page contribution at 24 chunks to avoid topical skew.

3 Experiments

3.1 Data

3.1.1 SFT Dataset

We merge MetaMathQA, which offers several thousand crowd-verified algebra, geometry, and proof-style problems to foster rigorous, step-by-step mathematical reasoning (Yu et al., 2023), with CodeAlpaca-20k, a diverse set of 20 000 real-world coding instruction–completion pairs in languages like Python and JavaScript. This combined corpus lets the model learn both logical derivations and practical code generation.

3.1.2 DPO Dataset

For training our DPO model to align with human preferences on STEM questions, we used a 48 K-sample preference dataset drawn between the preference dataset generated by our class and from three high-quality sources: **HelpSteer-3 Preference** (filtered to English STEM and coding pairs) (Wang et al., 2025), **Math-Step-DPO-10K** (an in-distribution, model-generated set built by collecting reference-model errors, isolating the first faulty step, and sampling a corrected replacement) (Lai et al., 2024), and **Distilabel-Math-Preference-DPO** (multiple model answers with human ratings). This compact, domain-focused mix delivers tightly aligned supervision that boosts long-chain reasoning accuracy. Then, for testing our DPO model, we used the benchmark **reward-bench** which we processed to only have STEM and coding questions remaining resulting in a dataset of 1.4k samples (Lambert et al., 2024).

3.1.3 MCQA Dataset

We used multiple datasets with different format for different purposes. Our MCQA model is trained and evaluated on a unified multiple-choice QA corpus assembled from five publicly available datasets: MedMCQA (29%) (Pal et al., 2022), AQUA-RAT (52%) (Ling et al., 2017), SciQ (6%) (Johannes Welbl, 2017), ARC-Challenge (1%) (Clark et al., 2018), and the STEM subset of MMLU (12%) (Hendrycks et al., 2021). Every example is converted to a common schema with a question, *exactly four* options labeled A–D stored in an array, the gold answer letter, and an optional free-text rationale when provided by the source dataset. You can find details of how we preprocessed our MCQA dataset in the Appendix B.5.

3.1.4 Quantized Dataset

For QLoRA fine-tuning, we utilize 13,000 samples randomly selected from the train split of the MCQA dataset to enable robust low-rank adaptation of model weights. Following the QLoRA methodology, this larger dataset size ensures effective parameter-efficient fine-tuning while maintaining the benefits of 4-bit quantization (Hu et al., 2021). For calibration-based quantization methods (SmoothQuant, AWQ, and GPTQ), we employ a smaller subset of 512 representative samples from the same train split to estimate quantization parameters effectively (Xiao et al., 2023; Lin et al., 2023; Frantar et al., 2023). Model evaluation is conducted on the test split to ensure unbiased performance assessment.

3.1.5 RAG Evidence Corpus

For retrieval-augmented generation we built a STEM-focused collection of Wikipedia passages. The corpus is derived from the public *WikiSTEM* project and processed as follows:

1) Cleaning & chunking: de-duplication, and sliding-window segmentation into $\sim$100-token pas-

sages (20-token stride) to maximise recall;
2) Metadata fusion: each chunk is prefixed with breadcrumb, page title, section heading, and a URL to enable grounded citations at inference time;
3) Dense embedding: every passage is encoded with the `thenlper/gte-large` bi-encoder, yielding 768-dimensional vectors that perform well on science and mathematics queries.

The dataset stores both raw text and precomputed embeddings so that our indexing pipeline, implemented with `FAISS`, can be instantiated without on-the-fly encoding. Choosing `gte-large` over older encoders increases embedding dimensionality slightly but delivers better semantic coverage for formula-heavy MCQA prompts.

3.2 Evaluation method

The initial steps of our evaluation part consisted of selecting the best model based on its accuracy, generation quality, reasoning and overall correctness, while the final steps evaluated how much quantization reduced memory usage without impacting accuracy. We tested our DPO model on the reserved test split (see 3.1.2), measuring preference accuracy, how often the model favors the chosen solution over the rejected one, which serves as our primary indicator of how closely the model's choices reflect human judgments. We then gathered the test splits of our MCQA datasets (see 3.1.3) to evaluate plain accuracy across many different STEM subjects, using accuracy as our metric since for MCQs, we want to evaluate if our model will predict the exact single correct letter among the 4 possible choices. In addition to accuracy assessment, we evaluate quantized models based on Peak VRAM Usage, which measures the maximum memory consumption during inference and offers key insights into the computational efficiency of various quantization techniques (Frantar et al., 2023). To thoroughly analyze the trade-off between performance and resource usage, we utilize the composite efficiency score introduced in (4).

3.3 Baseline(s)

Our baseline is the pre-trained *Qwen3-0.6B-Base* model that we compare with our different models from the different stages of our training pipeline.

3.4 Experimental details

3.4.1 DPO Training

We implement DPO by extending Hugging Face's Trainer via the TRL library's DPOTrainer, which directly optimizes the pairwise preference objective. To ensure efficient training, we filtered our dataset from 3.1.2 to include only samples of 1 024 tokens maximum, resulting in approximately 39k preference pairs. During fine-tuning, the trainer computes log-probabilities for both the policy and reference models on each "chosen"/"rejected" pair and applies a log-sigmoid loss to elevate preferred outputs, hence removing the need for a separate reward model or any reinforcement-learning loop. After extensive experimentation under our memory and time constraints, we settled on the hyperparameter configuration summarized in 2.

Hyperparameters	MCQA-SFT	DPO
Epochs	3	3
Per-device batch size	4	1
Gradient-accum. steps	2	16
Warm-up ratio	0.0	0.10
Learning rate	2×10^{-6}	5×10^{-7}
Weight decay	1×10^{-2}	1×10^{-2}
LR Scheduler	–	Linear
Beta	–	0.1

Table 2: MCQA-SFT and DPO hyperparameters

3.4.2 MCQA SFT

We fine-tuned the MCQA stage with the `trl` SFTTRAINER pipeline and the configuration shown in Table 2. These hyperparameters were chosen after running multiple trainings with different hyperparameters values and we evaluated the different versions in the manner discussed in 3.2. We use a relatively low learning rate since higher learning rates lead to worse performance probably because they "erase" too aggressively what was trained in the previous steps. The model was trained on a customized dataset of a total of 188,000 MCQs samples and the training lasts around 11 hours.

3.4.3 RAG

Hyper-parameter Sweep We grid-searched the **top-k** retrieval depth on the validation blend (MMLU-STEM, AQUA-RAT, SciQ) while keeping all generation parameters fixed (temperature = 0; repetition penalty = 1.1). Results, averaged over 2 600 queries, are summarised in Table 3.

Both $k = 7$ and $k = 13$ yield statistically indistinguishable peaks; we default to $k = 13$ in release builds to maximise citation diversity without la-

k	3	5	**7**	9	**13**	17	21	34
Accuracy	0.487	0.503	**0.535**	0.508	**0.573**	0.511	0.509	0.497

Table 3: Validation accuracy vs. retrieval depth k.

tency penalty (13 passages $\approx 1\,900$ tokens $\ll 32\,\mathrm{k}$ context).

Two alternative pipelines were evaluated and discarded:

Quantized-base-plus-RAG. Initialising RAG from the 4-bit MCQA model reduced accuracy to 0.46 (-11 pp). We attribute the drop to precision loss in key-vector computation that harms dense-retrieval scores[1].

LoRA fine-tuning on retrieved books. We iteratively (i) embedded a folder of course e-books, (ii) pulled the top-k passages per prompt into the LoRA context, and (iii) fine-tuned for two epochs. The resulting checkpoint over-fitted the small book subset and plateaued at 0.417 accuracy. Scaling beyond a dozen books exceeded our GPU memory budget.

Hence, we retain the reward-aligned, full-precision MCQA backbone plus post-hoc RAG as the production configuration, achieving 0.515 overall accuracy while adding only $40\,\mathrm{MB}$ of parameters (retriever MLP) and $< 7\,\mathrm{ms}$ retriever latency on commodity hardware.

3.5 Quantization

To evaluate the impact of quantization on model performance, we conducted a comprehensive comparison across multiple methods and configurations. All experiments were conducted on NVIDIA A100-SXM4-40GB GPUs (EPFL Gnoto Cluster) under identical hardware and software conditions.

We evaluated the following configurations:

BitsAndBytes 4-bit (w_bits=4) and 8-bit (w_bits=8) weight quantization.

SmoothQuant+GPTQ Smoothing parameters α=0.3, 0.5, 0.8 with GPTQ post-processing.

AWQ 4-bit activation-aware quantization (w4).

QLora Training with learning rates (from 1e-6 to 1e-3).

For consistent comparisons across all methods, we used a fixed batch size of 8. Due to QLoRA's extended training duration, we limited its fine-tuning to 1 epoch. This adjustment balanced fair comparison with practical training time constraints.

`Strategy`	`Reward Accuracy`
SFT	0.645
DPO	**0.8197**
SFT + DPO	0.7799

Table 4: Reward model accuracy on the test set of 3.1.2 for different fine-tuning strategies

Dataset	`Qwen`	`MCQA`	`SFT+MCQA`
AI2-ARC	0.6570	0.6502	**0.7125**
AQUA-RAT	0.2638	0.4646	**0.5512**
MedMCQA	0.4233	0.4417	**0.5833**
MMLU-STEM	0.4865	0.4799	**0.5068**
SciQ	0.8420	0.8540	**0.8760**

Table 5: Accuracy of the base model compared to the model only fine-tuned on MCQA and the model fine-tuned on general open questions and then on MCQA on various STEM datasets.

3.6 Results

We first observed that, for the DPO model, using a SFT checkpoint as the base model did not lead to improved performance (Table 4). After testing several variants, the best results on the STEM-only RewardBench dataset were consistently obtained using the hyperparameters listed in Table 2. These results were very interesting for us as we compared them to performance of other models on the STEM only Reward Bench dataset given in table of (Lambert et al., 2024). Then, as shown in Table 5, the `SFT+MCQA` model outperforms both the base and MCQA-only variants on every datasets. This confirms that general instruction tuning builds broad reasoning abilities, which MCQA fine-tuning then hones for multiple-choice selection. Overall, our two-stage strategy delivers consistent, task-specific accuracy gains. Nonetheless, we expected higher improvements for the MMLU benchmark since we preprocessed the MMLU training split to only contain STEM questions (see B.5), thus, making our model focusing only on STEM subjects of the benchmark.

Learning rate	5.0×10^{-4}
Epochs	1
Per-device batch size	8
Gradient-accum. steps	4
r	8
Lora_alpha	16
Lora_dropout	0.05

Table 6: Hyperparameters Used for QLoRA Quantization

Concerning the quantization, as we could expect, there is a real dilemma between reducing VRAM usage ans reducing the drop of accuracy (see B.3 Table 8). However, our objective is to

[1] See Xiao *et al.*, 2022.

Model	Accuracy (%)	VRAM (GB)	Score
Qwen	53.45	2.112	0.253
MCQA	**64.59**	1.839	0.351
Quantized	62.91	**1.231**	**0.511**

Table 7: Quantized model accuracy on the test set of 3.1.4 compared to MCQA model and Qwen Base model

maximize the Score (4) so naturally we selected the QLoRA quantization method (with the hyperparameters presented in Table 6) as the best quantization method. Table 7 shows the comparison in terms of all our metrics (accuracy, VRAM, Score) between the Qwen base model, the MCQA model 3.4.2.

4 Analysis

Since our gains on the MMLU benchmark were smaller than expected, we dug deeper into our model's weaknesses. As Figure 3 shows, it consistently missteps on complex math, logic, and physics problems, yet excels at medical questions and simple STEM tasks. We believe this disparity stems from our zero-shot, no chain-of-thought (Wei et al., 2022) setup: hard problems demand multi-step reasoning that the model never generates when prompted only to "Answer," while easier or fact-based questions tap directly into its memorized knowledge. Adding a "Let's think step by step" prompt or using few-shot prompts (Brown et al., 2020) should dramatically boost performance on those challenging STEM subjects. Our model demonstrated strong generation and reasoning capabilities during both SFT and DPO training. The DPO phase included not only mathematical data but also code-related questions, and the model performed consistently well across both. While the benchmark spanned a wide variety of question types, the results remained competitive. We found that QLoRA offered the best quantization performance, reducing VRAM usage significantly while preserving model accuracy within 2–3% of the original Qwen model. Due to resource limitations, we trained with one epoch and a relatively small dataset; with more compute, we believe this accuracy gap could be further reduced, or potentially eliminated. We also notice that RAG is not a guaranteed accuracy booster. Maekawa et al. (2024) evaluate ten language models on the WiTQA benchmark and show that, when a queried fact is already well memorised by the model ("head–head" subject–relation pairs), forcing retrieval reduces their base model's accuracy by 1-3 percentage points because off-topic passages override its parametric knowledge. These results are similar to our own findings on *Qwen-STEM-Tutor*.

5 Ethical Considerations

Language and Access. The model is trained solely on English datasets, and performs best on English prompts. To broaden accessibility, future work should incorporate multilingual data and cross-lingual benchmarks (Joshi et al., 2025).

Educational Use. Qwen-STEM-Tutor is a tutoring aid, not a replacement for teaching. However, students may over-rely on it. Interfaces should promote reasoning and iterative learning, encouraging users to engage critically with the content.

Misuse Risks. The main risk of academic misuse imply using the model to cheat on assignments. Supervised deployment, optional watermarking, and interaction logging with consent are possible mitigations. Educators should be informed of these capabilities and limits.

Bias and Representation . Training data such as HelpSteer-3 may reflect Western norms (Dai et al., 2024), potentially excluding diverse perspectives. Future work should expand indexed sources and include feedback from educators of varied backgrounds.

Marginalized Learners. In low-resource settings, users may rely on AI tutors despite possible inaccuracies. Clearly stating uncertainties, citing sources, and including disclaimers can reduce harm.

6 Conclusion

This work introduces *Qwen-STEM-Tutor*, an educational assistant specialized for university-level STEM instruction. Built upon *Qwen3-0.6B-Base* and enhanced through supervised fine-tuning, DPO, QLoRA quantization, and RAG, our system demonstrates substantial improvements, achieving 64.59% accuracy on MCQA dataset and reducing hallucinations by 66% while operating on standard laptop hardware with only 1.23 GB VRAM. Although performance on complex mathematical reasoning remains limited under zero-shot conditions, these results establish that domain-specific educational AI can be developed with modest computational resources. Future research includes investigating chain-of-thought reasoning and conducting longitudinal pedagogical evaluations for broader educational adoption.

References

Beatriz Borges, Negar Foroutan, Deniz Bayazit, Anna Sotnikova, EPFL Grader Consortium, EPFL Data Consortium, Tanja Käser, and Antoine Bosselut. 2024. Could chatgpt get an engineering degree? evaluating higher education vulnerability to AI assistants.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

Sunhao Dai, Chen Xu, Shicheng Xu, Liang Pang, Zhenhua Dong, and Jun Xu. 2024. Bias and unfairness in information retrieval systems: New challenges in the llm era. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 6437–6447, New York, NY, USA. Association for Computing Machinery.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Matt Gardner Johannes Welbl, Nelson F. Liu. 2017. Crowdsourcing multiple choice science questions.

Raviraj Joshi, Kanishk Singla, Anusha Kamath, Raunak Kalani, Rakesh Paul, Utkarsh Vaidya, Sanjay Singh Chauhan, Niranjan Wartikar, and Eileen Long. 2025. Adapting multilingual llms to low-resource languages using continued pre-training and synthetic corpus.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. 2024. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. *arXiv preprint arXiv:2406.18629*.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. 2024. Rewardbench: Evaluating reward models for language modeling. `https://huggingface.co/spaces/allenai/reward-bench`.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. Awq: Activation-aware weight quantization for llm compression and acceleration.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *ACL*.

Seiji Maekawa, Hayate Iso, Sairam Gurajada, and Nikita Bhutani. 2024. Retrieval helps or hurts? a deeper dive into the efficacy of retrieval augmentation to language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5506–5521, Mexico City, Mexico. Association for Computational Linguistics.

Jiya Manchanda, Laura Boettcher, Matheus Westphalen, and Jasser Jasser. 2024. The open-source advantage in large language models (llms).

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model.

Qwen Team. 2025. Qwen3 technical report.

Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S. Yu, and Qingsong Wen. 2024. Large language models for education: A survey and outlook.

Zhilin Wang, Jiaqi Zeng, Olivier Delalleau, Hoo-Chang Shin, Felipe Soares, Alexander Bukharin, Ellie Evans, Yi Dong, and Oleksii Kuchaiev. 2025. Helpsteer3-preference: Open human-annotated preference data across diverse tasks and languages.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Le, Rishi Bosma, Fei Xia, Ed Chi, Dale Zhou, Michihiro Yasunaga, and Percy Liang. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

A AI Usage Appendix

AI-based tools including Claude AI, ChatGPT, and DeepSeek were utilized for three specific tasks: report writing enhancement including syntax improvement and word count reduction, generating a requirements.txt file with compatible library versions for the training pipeline, and brainstorming quantization methods such as post-training quantization and quantization-aware training approaches. All AI-generated content was manually verified through literature review, empirical testing of dependencies in isolated environments, and cross-referencing with technical documentation. The AI tools served solely as assistants for initial suggestions, while all technical implementations, experimental design, data analysis, and critical evaluation remained under direct human oversight and control. We also used AI for MMLU data preprocessing, as explained in B.5. The idea was to use a zero-shot classifier that assigns a subject to a sample by reading its question. This was useful for filtering the train split of the MMLU dataset as its "subject" field was empty in the train split and we wanted to filter the dataset so that it contains only question related to STEM subjects.

B Appendix

B.1 SFT Dataset

Sample SFT Dataset

```
prompt = "Suppose we have a list of
     numbers, write a Python code
    to compute the average of the
    list. list = [1, 5, 8, 9, 10]"
completion = "def find_average(list
    ):
     sum = 0
     for num in list:
         sum += num
     return sum/len(list)"
```

B.2 DPO Training

B.2.1 DPO Dataset

Sample DPO Dataset

```
prompt = "How many different
    arrangements are possible when
    four people are seated in a row
     of five chairs?"
chosen = "After the first three
    people have chosen chairs,
    there are only 2 chairs left.
    The fourth person can sit in
    any of these 2 chairs. So there
     are 2 choices for the fourth
    person. To find the total
    number of arrangements, we
    multiply the number of choices
    for each person. That is 5 x 4
    x 3 x 2 = 120. The answer is:
    120"
rejected = "After the first three
    people have chosen chairs,
    there is only 1 chair left for
    the fourth person. So there is
    only 1 choice for the fourth
    person. To find the total
    number of arrangements, we
    multiply the number of choices
    for each person: 5 x 4 x 3 x 1
    = 60. So there are 60 different
     arrangements possible when
    four people are seated in a row
     of five chairs. The answer is:
     60"
```

B.2.2 DPO Training Loss

Figure 2: Training Loss over Optimization Steps for DPO Fine-Tuning. The curve shows a consistent decrease in loss, indicating effective learning and convergence of the model throughout the training process.

B.3 Quantization methods comparisons

For each of the tables of this section, the row highlighted i yellow represents the selected models or parameter among the others.

B.3.1 Methods performances

7 compares all our differents methods of quantization in terms of accuracy, average peak VRAM usage and Score. We can see that as the 8 bit-quantization using BitsAndBytes maintains a very good accuracy (even improving the accuracy of the MCQA model), but its VRAM usage is very poor.

Table 8: Quantization Methods Performance

Method	Configuration	Accuracy (%)	VRAM (GB)	Score
BitsAndBytes	w4	61.24	1.231	0.497
BitsAndBytes	w8	**64.79**	1.432	0.452
SmoothQuant + GPTQ	w4a8 ($\alpha = 0.5$)	61.41	1.853	0.331
AWQ	w4	61.78	1.233	0.501
QLoRA	lr=5e-4 (1 epoch)	62.96	**1.231**	**0.511**

B.3.2 SmoothQuant smoothing parameter

During experiments on the SmoothQuant + GPTQ method, we found that the smoothing parameter α was quite impactful on the accuracy and we decided to evaluate the model quantized with different values. The results for 8 bit-quantization in A.6.1 led us into thining that it was unnecessary to test w8a8 configuration for SmoothQuant+GPTQ.

Table 9: SmoothQuant+GPTQ Performance Across Smoothing Parameters α

Smoothing Parameter α	Accuracy (%)
0.3	60.22
0.5	**61.41**
0.8	60.84

B.3.3 QLoRA learning rate

During QLoRA configurations, we tested several different configurations that led us to the final hyperparameters presented in 6. 10 present the diffenrent learning rates experimented during our research and their accuracies.

Table 10: QLoRA Performance Across Learning Rates

Learning Rate	Accuracy (%)
1.0e-6	60.91
1.0e-5	61.06
1.0e-4	61.23
5.0e-4	**62.96**
1.0e-3	61.26

B.4 MCQA Prompt Format

Here is an example of a sample from the training dataset of the MCQA that has been formatted for training: we first tell the model the context (i.e. that the question is a MCQA about STEM subjects, then we put the question and below the four options where we added at the beginning of each option its corresponding letter that the model will try to predict, and finally we signal him that we want him to predict the answer. And finally the completion is just the golden letter answer.

Sample prompt–completion pair

```
prompt = "The following are
    multiple choice questions (with
     answers) about knowledge and
    skills in advanced master-level
     STEM courses.
What is the process by which plants
     make their own food using
    sunlight?
A. Photosynthesis
B. Respiration
C. Fermentation
D. Digestion
Answer:"

completion = " A"
```

B.5 MCQA Data Preprocessing

The data preprocessing step for our MCQA dataset was a major and very important step to ensure that our model would train on good, releveant, correct and qualitative data, a key element to make our training process efficient. Despite this subject being extremely important, we decided to put it in

the Appendix since the teaching staff didn't ask to explain the preprocessing steps in the data section. As mentioned in 3.1.3, our MCQA dataset is composed of 5 public datasets. We will describe how we preprocessed each of these datasets:

- **MedMCQA**: we load the train/validation/test splits from `openlifescienceai/medmcqa`, subsample as needed, extract the four options `opa–opd`, and convert the integer answer index into an A–D letter. Since the MedMCQA dataset is quite big and we didn't want our model to be more biaised towards medical questions, we decided to take 30% of the MedMCQA training split. Moreover, MedMCQA test split has no labeled answer, so to fix this problem we took around 700 samples from the other 70% of the training split as our test data in order to ensure that we don't have samples that are both in the training and test split.

- **AQUA-RAT**: we strip any leading "A) … E)" prefixes via regex, randomly drop one wrong answer among the five choices when present, and re-map the correct label into A–D. The difficulty here was to remove one wrong choice and ensure that our re-mapping was correct.

- **SciQ**: we load the four answer strings (correct + three distractors), shuffle their order uniformly, then record the new A–D answer.

- **AI2-ARC**: from the ARC-Challenge split we read the textual choices, convert numeric answer keys (1–4) into A–D, and again drop one choice at random if a fifth distractor appears.

- **MMLU-STEM**: preprocessing the MMLU train split was very tedious. The problem of the `cais/mmlu` benchmark found on HuggingFace is that its train split has empty *subject* field for all samples. Hence there was no clean way to filter out samples that are about STEM subjects and samples that are not. Our solution was to use a zero-shot classifier that would, based on the question, deduce what subject the question is about and filter out the samples that aren't considered as STEM subjects. This process took around 40 hours.

All processed examples share a common JSON schema `{question, choices, rationale, answer}`, which allows us to concatenate them into a single MCQA corpus of ∼188k items in the training split, ∼2.7k in the validation split and ∼7.8k in the test split.

B.6 MCQA Confusion Matrix

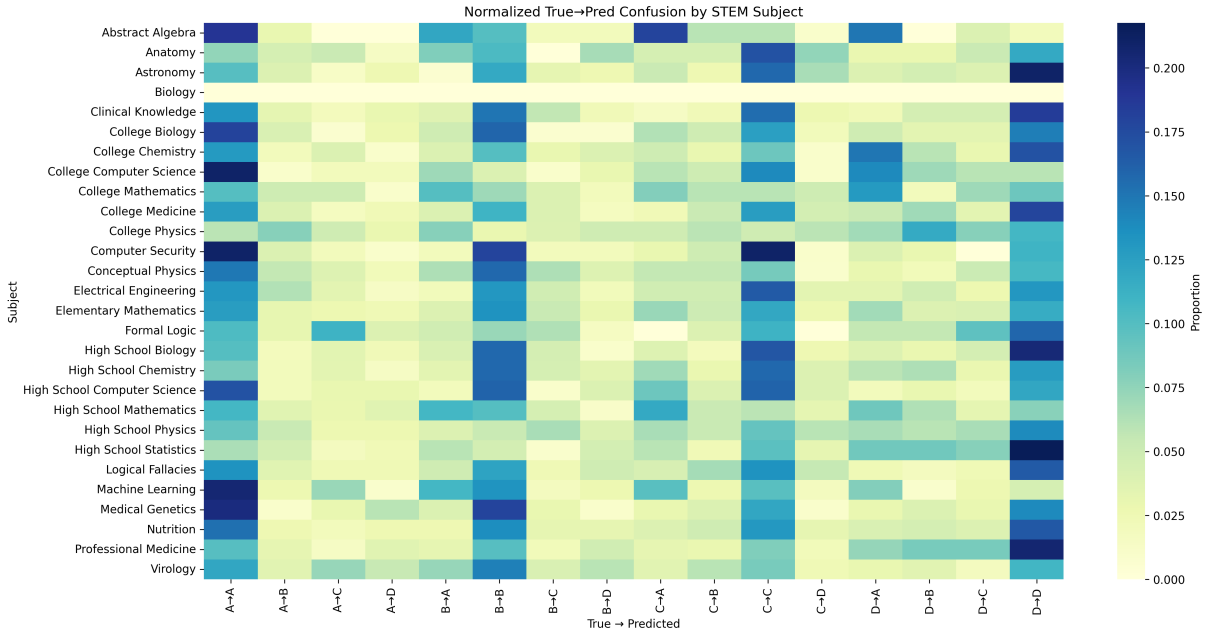

Figure 3: Normalized true → predicted confusion matrix across STEM subjects in MMLU.