Visual Multiple-Choice Science QA via QLoRA Fine-Tuning of SmolVLM-500M

Sarvar Khamidov
NYU Global Public Health
sk10583@nyu.edu

Hivansh Dhakne
NYU Tandon School of Engineering
hd2296@nyu.edu

Abstract

We fine-tune `SmolVLM-500M-Instruct` with QLoRA for visual scientific multiple-choice question answering (MCQ) under a strict 5 M trainable-parameter budget on free-tier T4 GPU hardware. Starting from a 0.819 public leaderboard baseline, we reach **0.87525** through four staged, controlled ablations covering rank, learning rate schedule, LoRA target modules, LoRA scaling factor, gradient accumulation, regularization, and DoRA. A central finding is a cross-over interaction: raising LoRA alpha from 16 to 32 *hurts* by 2.7 pp with attention-only adapters but *helps* by 1.6 pp when MLP projections are also adapted — an effect that would be missed without testing both module conditions. In parallel, an independent 22-experiment inference ablation improves validation accuracy from 77.3% (naive scoring, no context) to 86.0% (per-letter log-probability scoring, full context).

Reproducibility. Code: https://github.com/khsarvar/Pixels-to-Prediction. Adapter weights: https://drive.google.com/drive/folders/1MPp68QvTl3fzrtRiz0Nt4ztb10EYMc9s. Seed 42 throughout; all library versions pinned in `requirements.txt`.

1 Introduction

Visual scientific question answering requires models to integrate heterogeneous image content — charts, maps, organism photographs, labeled diagrams, and data tables — with natural language questions and select the correct answer from 2–5 candidates. This competition formalizes that problem over K–8 science content, scoring predictions by 0-indexed answer accuracy.

The constraints are deliberately restrictive. The base model is fixed as `HuggingFaceTB/SmolVLM-500M-Instruct`; trainable parameters are capped at 5 M; compute is limited to free-tier Kaggle T4 or Google Colab; inference must run offline without internet access. These restrictions rule out larger models, external pretraining data, and multi-model ensembles, forcing careful allocation of the parameter budget.

We adopt QLoRA: lightweight adapters are attached to selected projections of a 4-bit quantized backbone, enabling meaningful adaptation within the parameter cap. At inference, candidates are scored by the log-probability assigned to each answer letter token at the `Answer:` position—a method we selected empirically through the inference study in §4.

The remainder of the report is structured as follows. §2 presents the dataset and the EDA findings that directly shaped our preprocessing and prompt design decisions. §3 describes the complete final configuration. §4 develops the inference pipeline through a staged ablation before any training hyperparameter search. §5 walks through four training stages, each changing exactly one variable at a time. §6 reports final results, per-subject breakdown, calibration, and all leaderboard scores.

2 Dataset and EDA

2.1 Structure and Statistics

The Kaggle Free-tier dataset contains 3,109 training, 1,048 validation, and 1,008 test examples. Each example consists of an RGB image, a natural-language question, 2–5 labeled answer choices (indexed 0–4), and two optional text fields: `lecture` (background knowledge) and `hint` (problem-specific guidance). A `solution` explanation appears in train and val but is completely withheld from test, ruling out chain-of-thought methods that rely on it.

Subject distribution is skewed: natural science accounts for 74% of all examples, social science for 23%, and language science for only 3% (approximately 80 training examples). Grades span 1–12 with concentration in grades 4–8. Tasks are 97%

closed-choice and 3% true-or-false. Distribution shift across splits is negligible (Jensen-Shannon divergence <0.05 on all categorical columns: subject, grade, topic, task, num_choices), confirming that test-time generalization does not require handling unseen covariate shift.

2.2 EDA Findings and Modeling Consequences

We performed exploratory analysis before any modeling. Four concrete decisions followed directly from what we found.

Both context fields must be included. The `lecture` field is present in 86% of examples and `hint` in 77%. Lectures supply topic-level background; hints often contain partial reasoning scaffolding. Our inference ablation (§4) confirms that removing both costs 8.7 pp — the largest perturbation in the entire inference study — validating the decision to always include them.

Long context requires careful truncation. Lecture character length has a median of ∼370 and a 95th-percentile tail exceeding 1,500 characters. SmolVLM's processor expands the `<image>` token into roughly 1,088 interleaved placeholder tokens at processing time. If long text is passed with `truncation=True`, the processor cuts into those placeholders and throws a shape-mismatch error at the language model head. The correct fix is to truncate lecture and hint to 400 characters *inside* `build_prompt`, before the processor ever sees the input. This ensures the image token structure is never at risk.

Language science will underperform. With only ∼80 training examples (3% of the total), language science is severely underrepresented relative to natural science (∼2,300). We recorded this as a prediction before training; §6.2 confirms it: language science is the lowest-performing subject at 75.0% val accuracy.

Images are visually diverse. All images are RGB, predominantly wide-aspect. Content spans photographs of organisms, US state maps with highlighted regions, bar charts and data tables, food chain diagrams, solution beakers, and labeled advertisement images. This diversity means no single visual modality dominates, and general-purpose image embeddings from SmolVLM's vision encoder must carry the load.

3 Model Description

3.1 Base Model and Quantization

We use `HuggingFaceTB/SmolVLM-500M-Instruct` loaded in 4-bit NF4 quantization: `bnb_4bit_quant_type=nf4`, `bnb_4bit_compute_dtype=float16`, `bnb_4bit_use_double_quant=True`. The frozen, unmodified backbone scores 44.5% on val under our inference method, establishing a concrete floor for the fine-tuning contribution.

3.2 Complete QLoRA Configuration

Parameter	Value
Base model	SmolVLM-500M-Instruct
Quantization	4-bit NF4, double-quant
LoRA rank (r)	8
LoRA alpha (α)	32
Scaling factor (α/r)	4.0
LoRA dropout	0.05
Bias mode	none
Task type	CAUSAL_LM
Target modules	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj
Trainable parameters	≈4.6 M
Learning rate	2×10^{-4} (flat/constant)
Optimizer	AdamW, β_1=0.9, β_2=0.999
Weight decay	0.01
Epochs	3
Per-device batch size	1
Gradient accumulation	8 steps (effective batch = 8)
Gradient clipping	1.0
Mixed precision	fp16 AMP
Random seed	42

Table 1: Complete model and training configuration. Every hyperparameter is listed explicitly; justifications are given in §5.

All seven target modules cover every attention projection (q, k, v, o) and every MLP projection (gate, up, down) in the language model backbone. Including MLP layers was validated experimentally in Stage 2 (§5.2). We list every target module name and the dropout value here because omitting them is the most common reproducibility gap in LoRA work.

3.3 Prompt Format and Training Objective

Each training example is formatted as:

```
<image>
Context:
{lecture[:400]}  {hint[:400]}
Question: {question}
Choices:
A. {choice_0} B. {choice_1} . . .
Answer: A
```

The correct letter is appended at training time. The loss mask is set so only this single token contributes to the gradient (`labels[:, :-1] = -100`), focusing all learning capacity on the answer prediction rather than fitting the prompt text.

4 Inference Pipeline Development

Before any training hyperparameter search, we built and validated the inference pipeline independently, running 22 experiments across seven axes. All experiments use the Run 5 adapter on the validation set. Because Run 5 trains on the combined train + val set, these results are in-distribution; absolute numbers are optimistic, but *relative rankings among methods* are reliable. The final inference design was locked before touching training ablations, so it cannot confound them.

4.1 Scoring Method

The most natural approach for MCQ scoring is *length-normalized log-probability*: compute the model's log-probability for each full answer choice and divide by the number of tokens, correcting for shorter answers being favored. However, this assumes correct answers are not systematically shorter than wrong ones—an assumption that fails here, since many questions pair a long correct answer against short distractors.

A better approach is *per-letter log-probability*: instead of scoring the full text of each choice, we read off the log-softmax at the `Answer:` position for the single letter token ("` A`", "` B`", etc.). This is entirely independent of choice text length and measures exactly what the model is trained to predict.

Scoring method	Val acc.	Δ
Length-normalized log-prob	80.25%	—
Full-choice log-prob	84.06%	+3.81 pp
Per-letter log-prob	**85.97%**	+1.91 pp

Table 2: **Effect of scoring method** on val accuracy. Per-letter scoring, the final method, outperforms both alternatives.

4.2 Context Fields

Starting from a minimal prompt with no context, we added each field incrementally.

Hint contributes more when used alone (+7.1 pp) than lecture (+3.8 pp), likely because hints often directly scaffold the reasoning step needed to select the right letter. Their combined gain (+8.7 pp) modestly exceeds the sum of individual gains, suggesting complementary information.

Context configuration	Val acc.	Δ
No context	77.29%	—
Lecture only	81.11%	+3.82 pp
Hint only	84.35%	+7.06 pp
Both fields	**85.97%**	+8.68 pp

Table 3: **Effect of context fields** on val accuracy. Both fields contribute independently; together they add 8.7 pp over no context, the largest gain in the entire inference study.

4.3 Image Resolution

We tested inference at 336px and 448px to see whether finer image detail helps.

Resolution	Val acc.	Δ
224px (training resolution)	**85.97%**	—
336px	83.59%	−2.38 pp
448px	83.11%	−2.86 pp

Table 4: **Effect of inference resolution** on val accuracy. Larger resolution hurts because the model was trained exclusively at 224px; the distribution shift outweighs any detail gain.

4.4 Other Experiments

Seven further axes were tested; all underperformed the baseline design. Injecting metadata (subject, grade, topic) into the prompt cost 0.8 pp, likely adding irrelevant tokens that dilute attention. Changing the trigger phrase from `Answer:` to `The correct answer is:` cost 1.1 pp. Permutation averaging over four choice orderings cost 2.6 pp — the additional passes may introduce inconsistency. Self-caption image augmentation cost 1.9 pp. Log-probability bias correction cost 0.9 pp. Running with a blank (black) image but otherwise identical prompt scored 74.6%, establishing that the image contributes +11.4 pp over text-only inference (relative to Run 5's 86.0 val accuracy). The frozen base model (no fine-tuning) scored 44.5%, showing that fine-tuning contributes a further +41.5 pp.

5 Training Experimentation

We ran 13 training configurations across four stages. **In every comparison, exactly one variable changes from the previous best.** All runs use exactly 3 epochs and constant LR 2×10^{-4}, eliminating epoch count and schedule as confounds. We

report public leaderboard accuracy as the primary metric.

5.1 Stage 1: Establishing the Baseline (Runs 1–3)

We begin with the most conservative configuration: $r = 8$, attention-only adapters, flat learning rate, trained on the train split. Before building up, we test two common defaults that might seem beneficial.

Does a cosine LR schedule help?

LR schedule	Rank	Pub. LB	Δ
Flat 2×10^{-4}	16	**0.81287**	—
Cosine (3 epochs)	16	0.79476	−0.018

Table 5: **Effect of LR schedule.** Both rows use $r = 16$ (Runs 2 and 3); no r=8 cosine run was performed. Run 1 ($r = 8$, flat LR, LB 0.81891) confirms the flat-LR advantage holds at lower rank. Cosine decays to near-zero by the final epoch, losing most of epoch 3's learning. Flat LR is fixed for all subsequent runs.

Does higher rank ($r = 16$) help over $r = 8$?

LoRA rank	Trainable params	Pub. LB	Δ
$r = 8$	2.08 M	**0.81891**	—
$r = 16$	4.20 M	0.81287	−0.006

Table 6: **Effect of LoRA rank.** $r = 16$ underperforms despite providing twice the adapter capacity: 3,109 training examples are too few to benefit from additional rank. We fix $r = 8$.

After Stage 1, the clean baseline is Run 1 at 0.819. Both tested defaults hurt, giving us a reliable starting point.

5.2 Stage 2: Modules and Data — A Full Factorial

Two natural improvements are (1) expanding LoRA targets to include MLP layers alongside attention, and (2) training on all available labeled data. Rather than apply them sequentially—which would confound their individual contributions—we run all four combinations as a 2×2 factorial.

MLP modules encode task-specific semantic associations that attention projections alone cannot recapture at low rank. Including all 7 projection types (q, k, v, o, gate, up, down) captures both the routing behavior in attention and the feature transformation in the MLP, giving the adapter more surface area to adapt. Adding train + val data increases training examples by 34% (3,109 → 4,157), providing more diverse examples across subjects and skills. Both changes together bring the score from 0.819 to 0.859 (+4.0 pp).

	Train only	Train + val
Attn only (Runs 1 / 3A)	0.81891	0.84909
Attn + MLP (Runs 4 / 5)	0.82293	**0.85915**
MLP gain	+0.004	+0.010
Data gain	+0.030	+0.037

Table 7: **Effect of LoRA target modules and training data** (2×2 factorial, public LB). Both factors improve performance independently, with roughly additive effects. MLP modules add a reliable secondary gain in both data conditions.

5.3 Stage 3: LoRA Alpha and the Module Interaction (Runs 3B, 6)

The LoRA scaling factor α controls the magnitude of adapter updates: larger α means larger steps per gradient update. Because Stage 2 established that module scope matters, we test the $\alpha = 16 \rightarrow 32$ change under *both* module conditions to check for interaction.

Target modules	$\alpha = 16$	$\alpha = 32$	Δ
Attention only	0.84909	0.82293	−0.027
Attention + MLP	0.85915	**0.87525**	+0.016

Table 8: **Interaction between LoRA alpha and target modules.** The effect of raising alpha *reverses sign* depending on which modules are adapted. This interaction would be entirely invisible if alpha were tested in only one module condition.

This is the central finding of the training study. With attention-only targets, stronger updates ($\alpha = 32$) cause overfitting on the limited data (−2.7 pp). With MLP targets included, the additional adapter capacity absorbs the stronger updates and generalizes better (+1.6 pp). We verify this is not just better training-set fit: Run 6 ($\alpha = 32$) has a *higher* epoch-3 training loss than Run 5 ($\alpha = 16$), confirming the gain comes from reduced overfitting rather than improved optimization. Run 6 (LB 0.875) is our best result.

5.4 Stage 4: Attempts to Push Further (Runs 7–10, DoRA)

Five additional experiments attempt to improve beyond Run 6.

Gradient accumulation (Run 7)

Halving accumulation (4 vs. 8 steps) doubles optimizer updates per epoch, acting as additional stochastic regularization. Tested at $\alpha = 16$ to isolate the effect: LB 0.861. Useful as a data point, but below Run 6's 0.875.

Stacking $\alpha = 32$ and accum=4 (Run 8)

α	Accum	Pub. LB	Δ vs. Run 6
32	8	0.87525	—
16	4	0.86116	-0.014
32	**4**	**0.79**	$\mathbf{-0.085}$

Table 9: **Interaction between LoRA alpha and gradient accumulation.** $\alpha = 32$ doubles the LoRA scaling; accum=4 doubles optimizer steps. Combining them amplifies LoRA updates $\approx 4\times$ relative to Run 6, causing catastrophic instability. These two hyperparameters interact multiplicatively and cannot be stacked without compensating by reducing the learning rate.

Regularization (Runs 9–10)

Regularization change	Pub. LB	Δ
None (Run 6 baseline)	0.87525	—
Dropout $0.05 \rightarrow 0.10$ (Run 9)	0.87525	0.000
Weight decay $0.01 \rightarrow 0.05$ (Run 10)	0.87323	-0.002

Table 10: **Effect of standard regularization.** Neither stronger dropout nor higher weight decay improves over Run 6, indicating a performance ceiling for standard methods given the parameter cap and dataset size.

DoRA with image augmentation

DoRA decomposes LoRA updates into separate magnitude and direction components, enabling richer parameter-efficient adaptation. However, DoRA adds one scalar per output dimension per target module, pushing trainable parameters above 5 M at $r = 8$. We reduced rank to $r = 7$ to stay within the cap. We also applied mild image augmentation (RandomResizedCrop scale 0.85–1.0, ColorJitter brightness/contrast 0.15; no flips, to preserve spatial layout).

Stage 4 produced no improvement over Run 6. The performance ceiling appears to be a function of dataset size and the parameter cap rather than choice of regularizer or adapter architecture.

Architecture	Rank	Pub. LB	Δ vs. Run 6
LoRA (Run 6)	8	0.87525	—
DoRA + aug	7	0.78269	-0.093

Table 11: **Effect of DoRA architecture and image augmentation.** DoRA required $r = 7$ to meet the 5 M cap. The rank reduction from 8 to 7 likely sacrifices more than the architectural improvement gains.

6 Results

6.1 Baseline Comparisons

System	Pub. LB
Majority-class guess	0.326
Frozen SmolVLM (no fine-tuning)	0.445
Text-only inference (black image)	0.746
Run 6 (best, fine-tuned)	**0.875**

Table 12: **Baseline comparisons** (public LB scores). Fine-tuning contributes $+43.0$ pp; the image modality contributes a further $+12.9$ pp over text-only inference.

6.2 Per-Subject Analysis

Subject	n	Val acc.
Language science	28	75.0%
Natural science	777	83.4%
Social science	243	95.5%
Overall	1,048	86.0%

Table 13: **Per-subject validation accuracy.** Language science is lowest, as predicted from its underrepresentation. Social science benefits from the model's strong ability to read maps and structured tables, which dominate social science content. Natural science spans many topics (biology, physics, chemistry, earth science), so no single skill dominates.

6.3 Calibration Analysis

Confidence bin	n	Accuracy
[0.0, 0.4)	19	26.3%
[0.4, 0.6)	75	49.3%
[0.6, 0.75)	53	60.4%
[0.75, 0.9)	65	66.2%
[0.9, 1.0]	836	93.8%

Table 14: **Model calibration on the validation set.** 79.8% of predictions fall at $\geq$0.9 letter-probability with 93.8% accuracy, indicating the model is well-calibrated at high confidence. In the 0.6–0.9 range the model is overconfident (reported 60–90%, true accuracy 60–66%). Predictions below 0.4 confidence are essentially random at 26.3%.

6.4 All Training Runs

For runs trained on train-only (Runs 1, 3, 4), the LB$-$val gap is 0.027–0.030 (mean 0.028, $\sigma = 0.002$). For runs trained on train + val, validation is in-distribution so we report only the public LB as the independent held-out measure.

Run	Key change	Modules	r	α	Data	LB
Stage 1 — Baseline						
1	Baseline ($r = 8$, flat LR)	attn	8	16	tr	0.819
2	Cosine LR	attn	16	16	tr	0.795
3	$r = 16$, flat LR	attn	16	16	tr	0.813
Stage 2 — Modules × Data factorial						
4	+MLP, train only	a+M	8	16	tr	0.823
3A	+train + val, attn	attn	8	16	t+v	0.849
5	+MLP + train + val	a+M	8	16	t+v	0.859
Stage 3 — Alpha interaction						
3B	$\alpha = 32$, attn-only (control)	attn	8	32	t+v	0.823
6	$\alpha = 32$**, attn+MLP**	**a+M**	**8**	**32**	**t+v**	**0.875**
Stage 4 — Further attempts						
7	Accum=4, $\alpha = 16$	a+M	8	16	t+v	0.861
8	Stack: $\alpha = 32$+accum=4	a+M	8	32	t+v	0.790
9	Dropout 0.10	a+M	8	32	t+v	0.875
10	WD 0.05	a+M	8	32	t+v	0.873
DoRA	DoRA+aug, $r = 7$	a+M	7	32	t+v	0.783

Table 15: **All 13 training runs** (public leaderboard accuracy). *tr* = train split; *t+v* = train + val. All runs use 3 epochs and constant LR 2×10^{-4}. Runs 3A and 3B are necessary factorial controls.

7 Conclusion

Starting from a 0.819 baseline, we reached 0.875 through four sequential stages of controlled ablation. The progression was: eliminate harmful defaults (cosine LR, high rank) $\rightarrow$ establish both contributions via factorial (MLP modules, more data) $\rightarrow$ discover the alpha$\times$modules interaction $\rightarrow$ confirm the performance ceiling through regularization and DoRA.

Three findings stand out. **The alpha$\times$modules interaction** (Table 8) is the most practically important: testing alpha in only one module condition gives the wrong conclusion about whether to raise it. **The stacking failure** (Table 9) shows that LoRA scaling factor and gradient accumulation interact multiplicatively; combining them without re-tuning the learning rate causes collapse. **The inference pipeline** contributes as much as training choices (Table 2, 3): the naive default loses 8.7 pp to the final design.

What worked. $r = 8$, $\alpha = 32$, attn + MLP modules, flat LR, 3 epochs, per-letter log-prob scoring, full context at 400-char cap, 224px.

What did not work. $r = 16$; cosine LR; $\alpha = 32$ on attention-only; stacking $\alpha = 32$ with accum=4; dropout increase; weight decay increase; DoRA at $r = 7$; every tested inference perturbation.

Limitations

All experiments use a single seed (42); variance across seeds is uncharacterized. The Kaggle Free tier provides roughly half the full dataset. Inference ablations are in-distribution on Run 5. The

alpha×modules interaction was discovered post-hoc rather than pre-registered. Per-subject results on language science use only 28 validation examples, making those estimates noisy.

Reproducibility

Code: https://github.com/khsarvar/Pixels-to-Prediction.
Adapter weights: https://drive.google.com/drive/folders/1MPp68QvTl3fzrtRiz0Nt4ztb10EYMc9s.
Steps: (1) On Kaggle, attach the competition dataset and the adapter dataset. (2) Run `training_run6.ipynb` (≈4 h on T4) to reproduce the Run 6 adapter. (3) Run `inference_final.ipynb` to evaluate on val then generate the test submission CSV. No hardcoded paths; all data paths are resolved via `/kaggle/input` autodetection with a local fallback. Library versions: `transformers==4.57.6`, `peft==0.18.1`, `bitsandbytes`, `accelerate`, `pillow`; full list in `requirements.txt`. Seed 42 is set for Python `random`, NumPy, PyTorch, and CUDA before any stochastic operation.

AI Tooling Disclosure

Claude (Anthropic) was used as a coding assistant for debugging, notebook construction. All experimental design, training runs, empirical results, and analysis were produced by the authors.