VLMINTUNE: A UNIFIED INTERFACE FOR DEVELOPING VISUAL INSTRUCTION TUNING METHODS

Anonymous authors
Paper under double-blind review

ABSTRACT

Developing a visual instruction-tuning method often requires changes beyond model parameters, including inserted modules, forward hooks, token masks, supervision, objectives, and inference reconstruction. When these changes are scattered across paper-specific code, reusing a method requires substantial integration effort. We present **VLMintune**, a research library that treats the complete tuning method as the reusable unit. Each tuning recipe specifies how a method modifies a supported VLM, what it trains, any method-specific inputs, supervision, or loss, and how its learned behavior is saved and reconstructed. The surrounding data processing, backbone-native formatting, training, inference, and evaluation remain shared. VLMintune supports weight adaptation, inserted modules, representation interventions, and alternative token supervision. We implement seven representative methods and evaluate them on Qwen2.5-VL-3B-Instruct across TextVQA, VizWiz, and ScienceQA, yielding 21 full-data runs that demonstrate consistent within-platform comparison of structurally different tuning methods.
Anonymous repository: `https://anonymous.4open.science/r/VLMintune-7407/`
Anonymous documentation: `https://anonymous.4open.science/r/VLMintune-7407/docs/index.html`

1 INTRODUCTION

Visual instruction tuning has become a standard approach for adapting pretrained vision language models (VLMs) to multimodal interaction (Liu et al., 2023; Dai et al., 2023; Liu et al., 2024a; Xu et al., 2024; Jiang et al., 2024). As the field develops, tuning methods increasingly differ not only in what parameters they update, but also in where they modify the learning process. Some methods adapt model weights through low rank parameterizations (Hu et al., 2022; Dettmers et al., 2023; Liu et al., 2024b), others insert trainable modules (Sung et al., 2022), intervene on intermediate representations (Wu et al., 2024; Bi et al., 2025), or modify token supervision and training objectives (Zhou et al., 2025). These mechanisms may therefore require different model preparation, runtime inputs, forward behavior, supervision, optimization, and reconstruction logic.

Modern libraries already make much of VLM post-training reusable. Transformers, PEFT, and TRL provide models, adaptation components, and training utilities (Wolf et al., 2020; Mangrulkar et al., 2022; von Werra et al., 2020); LAVIS and general post training frameworks connect models, datasets, and training pipelines (Li et al., 2023; Zheng et al., 2024; Zhao et al., 2025; Axolotl maintainers and contributors, 2023; Oumi Community, 2025); and LMMs-Eval and VLMEvalKit standardize multimodal evaluation (Zhang et al., 2025; Duan et al., 2024). However, when the tuning mechanism itself is the object of study, a new method may cross several of these reusable boundaries simultaneously. A representation intervention, for example, can affect model construction, runtime masks, the forward pass, trainable state, checkpoint reconstruction, and generation. Paper-specific implementations often integrate these requirements independently with their own code. Consequently, reusing or comparing a method may require rebuilding substantial surrounding infrastructure that is not intrinsic to the method itself. The missing abstraction is therefore not another individual adaptation component, but a reusable boundary around the complete tuning method.

We introduce **VLMintune**, a research library built around this boundary. VLMintune treats a tuning recipe as the reusable unit of implementation. A recipe specifies how a method modifies a supported

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

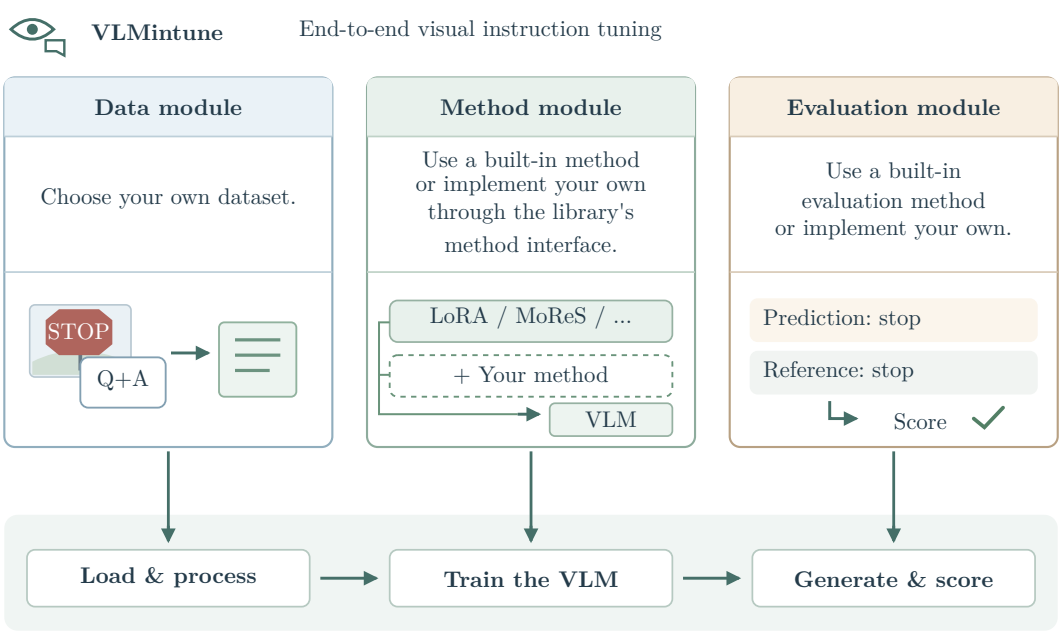


Figure 1: VLMintune provides an easy-to-use and end-to-end workflow for visual instruction tuning. Users choose a dataset, select a supplied tuning method or implement their own, and use a built-in or custom evaluation method. The data, method, and evaluation modules connect to a shared processing, training, and evaluation pipeline.

backbone, which state is optimized, any method specific inputs, supervision, or loss, and how the learned behavior is saved and reconstructed. The surrounding workflow remains shared across methods, including multimodal data processing, backbone native formatting, training, generation, and task evaluation. This design accommodates weight adaptation, inserted modules, representation interventions, and supervision modification without forcing them into a single parameter adaptation abstraction. As illustrated in Fig. 1, different tuning mechanisms can therefore use the same experimental workflow while retaining their distinct training behavior.

VLMintune is intentionally focused on method development rather than broad post-training infrastructure. The current release provides seven tuning methods, two VLM backbone adapters, and five VQA dataset adapters, with method availability varying by backbone. We evaluate the seven standalone recipes in 21 full data runs, where every method follows the same train, save, fresh reconstruction, generation, and task evaluation path. These experiments test whether structurally different tuning mechanisms can be implemented and compared within a shared workflow rather than establishing parameter-matched rankings of the underlying algorithms. Our contributions are four-fold:

(1) We introduce VLMintune, which treats the complete visual instruction tuning method as the reusable implementation unit and separates method-specific behavior from the surrounding experimental workflow.

(2) We provide a common interface for heterogeneous tuning mechanisms, including weight adaptation, inserted modules, representation interventions, method-specific runtime inputs, alternative supervision, and custom objectives.

(3) We integrate saving and fresh model reconstruction into the same tuning recipe, enabling structurally different methods to reuse a common inference and task evaluation pipeline.

(4) We provide seven methods, two backbone adapters, and five dataset adapters, and validate the workflow through 21 full-data runs on one backbone across three datasets.

2 Related Work

Instruction tuning. Instruction tuning is supervised post-training on examples written as instructions and desired responses. Datasets from different tasks can be cast in this form, allowing one generative model to learn several tasks without a separate output head for each. The instruction, optional context,

and earlier conversation turns are given as input, and the model learns to generate the target response. Loss is commonly applied only to response tokens. Instruction tuning improves how models respond to natural-language requests and can improve generalization to task formulations absent from the training set (Wei et al., 2022; Chung et al., 2022). The format can be used with full fine-tuning, selective parameter updates, or parameter-efficient methods.

Visual instruction tuning. Visual instruction tuning adds images to the instruction–response examples. Most systems join a pretrained vision encoder and language model with a projection layer or a query-based connector. LLaVA first aligns visual features with the language model and then trains on image-grounded conversations; InstructBLIP uses instruction-aware query representations to select visual information (Liu et al., 2023; Dai et al., 2023). Later work improves the connector and image resolution, expands the instruction-task mixture, or supports interleaved multi-image conversations (Liu et al., 2024a; Xu et al., 2024; Jiang et al., 2024). A typical pipeline turns each dataset example into a conversation with image placeholders and text turns, processes it with the model's tokenizer and image processor, and trains the response with an autoregressive loss. Implementations differ in their data mixtures, chat templates, frozen components, supervision, and schedules, even when they follow this same two-stage recipe.

3 Positioning among Multimodal Tuning Libraries

Existing ecosystem. Existing libraries support different parts of the multimodal post-training ecosystem. Libraries centered on tuning components or post-training algorithms, such as PEFT and TRL, provide reusable adaptation mechanisms and optimization procedures that can be integrated into larger systems (Mangrulkar et al., 2022; von Werra et al., 2020). General training frameworks, including LAVIS, LLaMA Factory, ms swift, Axolotl, and Oumi, connect models, datasets, configurations, and training infrastructure (Li et al., 2023; Zheng et al., 2024; Zhao et al., 2025; Axolotl maintainers and contributors, 2023; Oumi Community, 2025). Evaluation toolkits such as LMMs Eval and VLMEvalKit instead standardize model execution and scoring across multimodal benchmarks (Zhang et al., 2025; Duan et al., 2024). These categories are not mutually exclusive, but they represent different primary objects of reuse: tuning components, configured training workflows, and benchmark evaluation.

VLMintune's position. VLMintune focuses on a different reusable unit: the complete implementation of a visual instruction tuning method. Method specific behavior, including backbone modification, trainable state, runtime inputs, supervision, loss, and reconstruction, is defined together, while the surrounding data processing, training, generation, and evaluation pipeline is shared across methods. This design targets tuning mechanisms that may operate on model weights, inserted modules, intermediate representations, or training supervision. VLMintune therefore complements general post training frameworks rather than attempting to match their breadth in model coverage, training infrastructure, or deployment support.

Comparison dimensions. Table 1 compares tuning support and research workflow. Alongside backbone coverage, instruction formatting, weight adaptation, and adapter module insertion, it examines two narrower capabilities: visual-token-selective steering, which identifies visual-token positions and applies targeted representation interventions, and dataset-defined instruction-span supervision, which aligns dataset-specified instruction fragments to supervised token positions. These rows do not assess representation intervention or supervision modification in general. Workflow support covers configuration-based experiments, declarative field mapping, and a shared VQA evaluation pipeline. Capabilities provided through built-in APIs or documented integrations count for their supported models and methods; generic hooks that require a new implementation do not. VLMintune brings these capabilities together within a common experimental pipeline.

4 The VLMintune Library

VLMintune organizes visual instruction tuning around a common method interface. Its design goal is to keep method-specific changes local while allowing heterogeneous tuning mechanisms to reuse the surrounding experimental workflow. We describe the interface, illustrate it through the supported methods, and explain how it connects to shared data processing, training, and evaluation.

Table 1: Comparison of representative libraries for visual instruction tuning: (a) tuning support and (b) research workflow. ✓: directly supported; ×: no direct implementation identified in the inspected version.

Dimension	LAVIS	PEFT	LLaMA-F	ms-swift	Axolotl	VLMintune
(a) Tuning support						
Multi-backbone support	✓	✓	✓	✓	✓	✓
Unified instruction format	✓	×	✓	✓	✓	✓
Weight adaptation	✓	✓	✓	✓	✓	✓
Adapter module insertion	×	✓	×	✓	✓	✓
Visual-token-selective steering	×	×	×	×	×	✓
Dataset-defined instruction-span supervision	×	×	×	×	×	✓
(b) Research workflow						
Config-based experiments	✓	×	✓	✓	✓	✓
Declarative field mapping	×	×	✓	✓	✓	✓
Shared VQA evaluation pipeline	✓	×	×	✓	×	✓

Notes. Support is assessed through built-in APIs or documented integrations for supported models and methods; generic extension hooks alone do not qualify. Shared VQA evaluation covers generation, answer processing, and scoring. LLaMA-F: LLaMA-Factory.

4.1 A Unified Interface for Tuning Methods

VLMintune separates method-specific behavior from the surrounding experimental workflow through a common `TrainingMethod` interface. We use *recipe* to denote a concrete implementation of this interface together with its fixed method-specific choices. A recipe brings together the behavior needed to prepare, train, and reconstruct an adapted model, while shared components provide dataset processing, backbone-native formatting, the optimization loop, and task evaluation. This boundary allows methods with different adaptation mechanisms to use the same experimental workflow.

The interface covers model preparation, training behavior, and checkpoint reconstruction. During preparation, the recipe specifies how a supported backbone is modified and which parameters are optimized. During training, it defines any method-specific runtime inputs, supervision changes, or custom loss. It also specifies how learned state is saved and how the adapted model is reconstructed for inference. This last responsibility matters when weights alone do not determine the method's behavior: a representation intervention, for example, also depends on where and how it is applied. Defining reconstruction alongside training behavior keeps these dependencies within the method implementation.

Figure 2 illustrates how a method implementation enters the shared workflow. A developer defines a subclass of `TrainingMethod` and adds it to the method registry, after which the recipe can be selected by name. The code sketches this extension entry point; the subclass must still implement the method-specific behavior described above. Its registry entry must remain available during both training and checkpoint loading. Dataset adapters supply sample and scoring conventions to the shared workflow, while model adapters expose the backbone structure required by compatible recipes. Reuse therefore applies to supported model–recipe pairs, and a new backbone may require explicit adaptation. General training settings, such as learning rate, batch size, and epoch count, are supplied through configuration, while method-specific structural choices remain in recipe code. Changing an intervention rank, for example, may require corresponding updates to module construction and restoration.

4.2 Different Tuning Mechanisms, a Shared Workflow

The seven standalone methods illustrate how different tuning mechanisms use the common interface. LoRA and DoRA define weight adaptations and their trainable parameters, while QLoRA additionally requires quantized backbone preparation (Hu et al., 2022; Liu et al., 2024b; Dettmers et al., 2023). VL-Adapter introduces trainable modules whose placement and learned state must be reconstructed

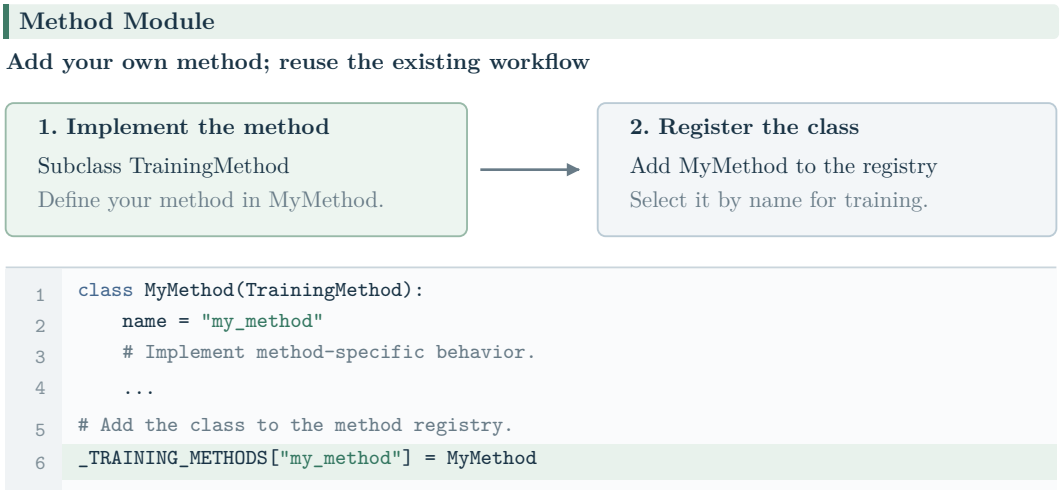


Figure 2: Method extension through the common interface. A new method implements `TrainingMethod` and is added to the method registry, making it selectable by name within the shared training and evaluation workflow. The example shows the subclass and registration structure; method-specific implementation details are omitted.

for inference (Sung et al., 2022). ReFT and MoReS require representation interventions, including the structures and positions involved in their application (Wu et al., 2024; Bi et al., 2025), whereas L2T introduces method-specific supervision during training (Zhou et al., 2025). These differences motivate an interface that covers model preparation, runtime behavior, supervision, and reconstruction rather than only the selection of trainable parameters.

MoReS and ReFT provide a concrete example of reuse across methods. Both intervene on hidden representations, but comparing their original implementations on a common VLM task requires aligning the backbone, data processing, training setup, and evaluation. For the supported Qwen2.5-VL and TextVQA setup, VLMintune provides this shared execution path: changing `method: mores` to `method: reft` retains the dataset adapter, native chat template, training loop, and answer scorer. Each recipe supplies its own intervention behavior and checkpoint reconstruction. Researchers can therefore compare the implemented recipes without migrating the surrounding data and evaluation code, while still selecting and reporting the optimization settings appropriate to each method.

The same separation provides a starting point for modifying an existing method. A change to an intervention is expressed in the recipe's model preparation or runtime behavior, together with any corresponding reconstruction changes; a change to supervision is expressed through the method's training behavior. When these changes remain within the supported interface, the dataset processing and evaluation pipeline can be reused. The interface thus gives structurally different methods a common execution path while keeping their adaptation-specific behavior explicit.

4.3 Shared Execution across Training and Evaluation

Figure 3 illustrates the separation between dataset conventions and backbone formatting. A dataset adapter converts a raw TextVQA record into a `CanonicalSample`, distinguishing the answer used for training from the reference answers retained for evaluation. The selected backbone then applies its native chat template and multimodal processor. The illustrated response-only supervision masks prompt labels; recipes with alternative supervision define the corresponding changes through the method interface. The code example connects another Hugging Face VQA dataset by mapping its columns, allowing compatible methods to reuse the same dataset connection. Less regular records require custom parsing, split definitions, and a scoring rule. The built-in adapters cover TextVQA, VQAv2, VizWiz, GQA, and image-bearing ScienceQA (Singh et al., 2019; Goyal et al., 2017; Gurari et al., 2018; Hudson & Manning, 2019; Lu et al., 2022).

Training connects the selected recipe to this data-processing path and the shared optimization loop. After training, the registered method implementation reconstructs the adapted model from its

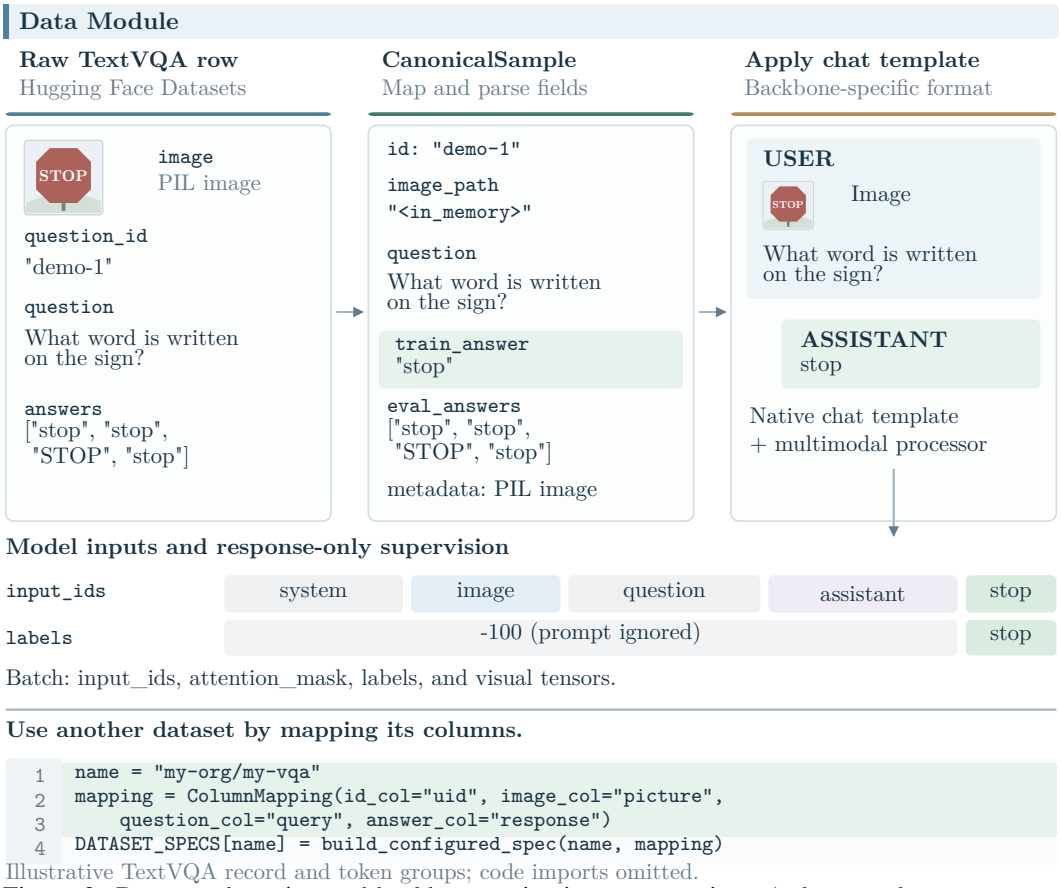


Figure 3: Dataset adaptation and backbone-native input processing. A dataset adapter maps an illustrative TextVQA record to a `CanonicalSample`, separating the training answer from the evaluation references. The backbone's native chat template and multimodal processor then produce model inputs. The illustrated response-only supervision masks prompt labels with -100. The code shows how another Hugging Face VQA dataset is connected through column mapping; green highlights indicate user-supplied choices.

checkpoint using the required method definition and structural choices. The restored model then enters the common generation and evaluation workflow, allowing methods with different adaptation structures to use the same task evaluator.

Figure 4 shows how built-in and custom evaluation implementations share this workflow. The built-in `LocalMethod` prepares multimodal inputs and generates answers, while the illustrated `MyEvaluation` subclass customizes these operations through `prepare_eval_input` and `generate`. Either implementation is passed to the same `evaluate_dataset` runner without a separate registration step. The dataset adapter specifies the scoring rule, such as consensus VQA accuracy for TextVQA or normalized exact match for ScienceQA, and the runner collects predictions, applies the scorer, and writes aggregate and per-example results. Input preparation and generation can therefore change while dataset loading, scoring, and reporting remain shared. Custom metrics are introduced separately through the scorer and dataset adapter.

5 Experiments

We use VLMintune as a common experimental platform to compare representative visual instruction-tuning methods and derive practical guidance for selecting tuning recipes. Comparisons across separate frameworks can conflate method differences with differences in data processing, training infrastructure, prompting, and scoring. Building on the recipe abstraction in Section 4, we standardize these surrounding components while preserving the adaptation and supervision choices that define each recipe. Our study comprises seven standalone methods evaluated on three tasks with Qwen2.5-

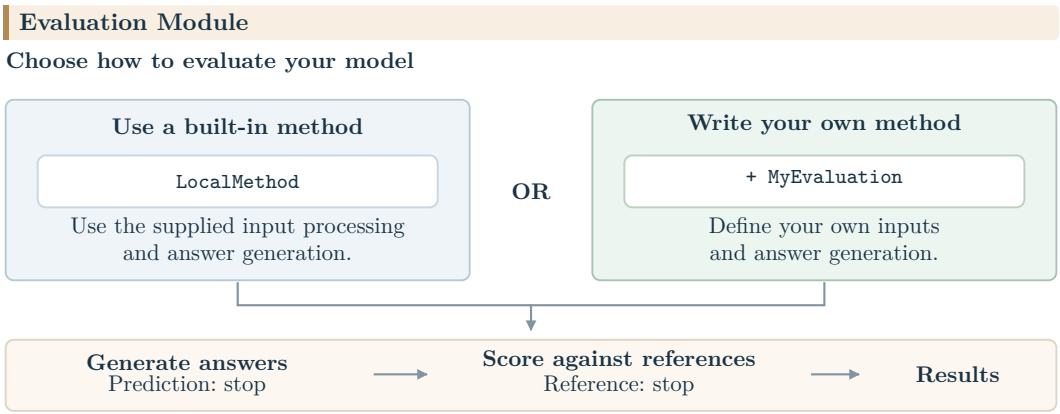


Figure 4: Customizable generation with shared evaluation. Users can select the built-in `LocalMethod` or supply a custom implementation. The illustrated `MyEvaluation` subclass customizes input preparation and answer generation through `prepare_eval_input` and `generate`, then passes the instance directly to `evaluate_dataset` without registration. Both paths reuse dataset loading, scoring, and result reporting. Method bodies and imports are omitted.

Table 2: The three datasets in the unified comparison. All seven methods use the same full training and validation splits within each dataset. ScienceQA includes only examples containing an image.

Dataset	Train	Validation	Metric
TextVQA	34,602	5,000	VQA accuracy
VizWiz	20,523	4,319	VQA accuracy
ScienceQA (image only)	6,218	2,097	Normalized exact match

VL-3B-Instruct, yielding 21 full-data training runs. We examine their task performance, analyze the relationship between trainable parameter counts and accuracy, and discuss the implications for practical method selection.

5.1 Unified Comparison Protocol

Backbone, methods, and datasets. We evaluate LoRA, QLoRA, DoRA, VL-Adapter, MoReS, ReFT, and L2T using Qwen2.5-VL-3B-Instruct (Bai et al., 2025). Within each dataset, all methods share the same dataset adapter, question and answer construction, backbone-native chat template, and multimodal processor. Each method uses the full training and validation splits of TextVQA (Singh et al., 2019) and VizWiz (Gurari et al., 2018), and the image-bearing training and validation subsets of ScienceQA (Lu et al., 2022). Table 2 summarizes the data coverage and evaluation metrics.

Shared training workflow and recipe configurations. All methods use the VLMintune trainer for one epoch with seed 42, batch size 1, four gradient-accumulation steps, and a maximum sequence length of 1,536. The fixed learning rates are 2×10^{-4} for LoRA, DoRA, ReFT, and MoReS; 5×10^{-5} for QLoRA; 10^{-4} for VL-Adapter; and 2×10^{-5} for L2T. No additional hyperparameter sweep is performed. Trainable modules, parameter counts, quantization, intervention masks, and supervision remain specific to each recipe. For example, L2T changes the supervised token set, whereas QLoRA uses a quantized backbone. The protocol therefore matches data exposure and the

surrounding workflow while making recipe differences explicit; it does not match parameter counts or computational budgets.

Evaluation and coverage checks. Each checkpoint is reconstructed on a fresh backbone in a separate process and evaluated through the same runner. Within each dataset, all methods share evaluation prompting, greedy decoding with at most 16 new tokens, answer normalization, and scoring. TextVQA and VizWiz use consensus-based VQA accuracy, and ScienceQA uses normalized exact match. All 21 runs reach their configured final training step with zero skipped samples and produce predictions for the full validation set. Validation identifiers and their order match across methods within each dataset. Only complete runs enter the comparison, including the complete TextVQA reruns. These checks ensure that the reported scores refer to saved recipes evaluated on the same examples through a common pipeline.

5.2 Comparative Results

Table 3 reports the performance and trainable parameter counts of the seven recipes. All scores are obtained within VLMintune. Comparisons are made within each dataset because the tasks and scoring metrics differ. LoRA obtains the highest observed scores on TextVQA and ScienceQA, at 82.12 and 93.18, while VL-Adapter obtains the highest score on VizWiz, at 75.93. DoRA closely tracks LoRA, with differences of 0.22, 0.14, and 0.09 points across the three tasks. L2T achieves the second-highest VizWiz score despite having the lowest TextVQA score. The results thus provide different candidate choices across tasks, with LoRA offering a competitive reference throughout the tested setting.

Task dependence extends beyond the top-ranked recipe. The variation across tasks is particularly apparent for MoReS and ReFT. On TextVQA, their scores of 81.24 and 81.25 are within 0.88 and 0.87 points of LoRA. Their gaps to the best observed score increase to 2.66 and 1.54 points on VizWiz, and to 7.87 and 8.06 points on ScienceQA. Near-matching performance on one task therefore does not imply the same relationship on another. Because prompting, evaluation examples, and scoring are shared within each task, these differences can be examined without combining results from separate evaluation pipelines. The present comparison does not identify which task properties or recipe components account for this variation.

Small updates can be competitive, but their accuracy cost varies. MoReS and ReFT train 0.147M and 0.590M parameters, approximately $102\times$ and $25\times$ fewer than LoRA's 14.967M. Their TextVQA results illustrate a substantial reduction in trainable parameter count with a gap of less than one point to LoRA, whereas their ScienceQA results show a larger accuracy cost. Conversely, increasing the number of trainable parameters does not consistently correspond to higher scores: L2T trains 3.123B parameters without leading any task, while VL-Adapter trains 112.521M parameters and leads only on VizWiz. These observations characterize the tested recipes jointly, since parameter count, adaptation mechanism, and optimization settings vary together.

Quantization and update size describe different resource choices. The QLoRA recipe trains 119.734M adapter parameters, approximately eight times the LoRA count, and scores 0.56, 1.44, and 2.15 points below LoRA on TextVQA, VizWiz, and ScienceQA, respectively. However, its frozen backbone is stored in four-bit form, a resource difference that the trainable-parameter column does not capture. The comparison therefore cannot attribute these score differences to quantization alone or determine which recipe uses less GPU memory. It distinguishes the size of the trainable update from the representation of the frozen backbone, both of which matter when configuring a tuning method.

5.3 Practical Recommendations

The observed results suggest three practical recommendations for selecting visual instruction-tuning recipes within the evaluated setting.

Use LoRA as an initial reference, then validate alternatives on the target task. LoRA is a reasonable starting recipe for the tested backbone and training budget: it obtains the highest observed score on two tasks and remains within 0.45 points of the best result on VizWiz. Alternatives should then be assessed on the intended task under the same validation protocol. For example, VL-Adapter and L2T are candidates worth checking on VizWiz, whereas their results on the other tasks do not

Table 3: Comparison of recipes on Qwen2.5-VL-3B-Instruct under the unified protocol. Each result uses one seed-42 training run followed by independent checkpoint reload. Scores are percentages. Best and second-best observed scores are bold and underlined; these markings do not indicate statistical significance. Trainable parameter counts describe update size, not total memory consumption.

Method	Trainable params. (M)	TextVQA ↑	VizWiz ↑	ScienceQA ↑
LoRA	14.967	**82.12**	75.48	**93.18**
QLoRA	119.734	81.56	74.04	91.03
DoRA	15.999	<u>81.90</u>	75.34	<u>93.09</u>
VL-Adapter	112.521	81.38	**75.93**	90.51
MoReS	0.147	81.24	73.27	85.31
ReFT	0.590	81.25	74.39	85.12
L2T	3,122.647	81.20	<u>75.74</u>	89.51

show the same advantage. The small differences between LoRA and DoRA should motivate repeated validation rather than a categorical preference based on a single run.

When limiting trainable parameters, measure the accuracy cost on each task. MoReS and ReFT are useful candidates when the objective is to reduce the number of trainable parameters. Their TextVQA results show that such reductions can retain performance close to LoRA, but their ScienceQA results caution against transferring that expectation across tasks. Practitioners should specify an acceptable accuracy gap and select among recipes that satisfy it on the target validation set. A small trainable update should not by itself be interpreted as lower peak memory or faster training.

Choose recipes using the resource constraint that matters in deployment. When comparing LoRA and QLoRA, inspect adapter capacity and quantization separately, since both differ in the recipes evaluated here. More generally, a requirement on trainable parameter count can be assessed from Table 3, whereas a GPU-memory or training-time constraint requires direct resource measurements under the intended hardware and configuration. VLMintune provides a common workflow for comparing these explicit recipe choices, allowing method selection to be grounded in the target task and the relevant constraint.

Scope of the recommendations. These recommendations concern the tested implementations on one backbone, with one run per configuration and fixed learning rates. Small score differences do not establish statistical superiority, and the results do not isolate the causal effect of any single method component. The current comparison also does not measure gains over the untuned backbone, resource consumption, or differences from the original method-specific frameworks. The recommendations should therefore be treated as starting points for target-task validation, rather than universal rankings of the underlying algorithms.

6 Conclusion

We presented VLMintune, a research library for developing, reusing, and comparing visual instruction-tuning methods through a common tuning recipe abstraction. Each recipe encapsulates method-specific adaptation, supervision, and checkpoint reconstruction while sharing data, training, and evaluation pipelines. This separation supports structurally different methods, including weight adaptation and representation intervention, throughout training and checkpoint-based evaluation. We evaluated seven methods on Qwen2.5-VL-3B-Instruct across TextVQA, VizWiz, and ScienceQA, revealing task-dependent performance and trade-offs between accuracy and trainable parameter count. LoRA provides a competitive starting point across the evaluated tasks, while compact intervention recipes approach its performance on TextVQA but exhibit larger gaps on ScienceQA. These findings suggest establishing a reference recipe, validating alternatives on the target task, and distinguishing trainable parameter count from actual resource consumption. VLMintune thus provides a shared experimental basis for method selection while keeping recipe-specific configurations explicit. Our evidence remains limited to one backbone, single-seed runs, and single-image question answering, making these recommendations conditional on the evaluated setting. Broader backbone coverage, repeated trials, and direct resource measurements are natural next steps toward more general guidance for visual instruction tuning.

AI USE STATEMENT

Generative AI tools assisted literature discovery, manuscript organization, language editing, and LaTeX formatting. The authors reviewed all cited sources, technical claims, and AI-assisted text. Generative AI did not produce experimental measurements or determine the reported results. The authors take responsibility for the final paper.

REPRODUCIBILITY STATEMENT

Our source code, method implementations, and example configurations are available in the anonymous repository below, together with anonymous documentation. Experimental settings and evaluation procedures are described in Section 5, and installation and execution instructions are provided in Appendix B.
Anonymous repository: `https://anonymous.4open.science/r/VLMintune-7407/`
Anonymous documentation: `https://anonymous.4open.science/r/VLMintune-7407/docs/index.html`

REFERENCES

Axolotl maintainers and contributors. Axolotl: Open source LLM post-training. `https://github.com/axolotl-ai-cloud/axolotl`, 2023. Software repository.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-VL technical report, 2025. URL `https://arxiv.org/abs/2502.13923`.

Jinhe Bi, Yujun Wang, Haokun Chen, Xun Xiao, Artur Hecker, Volker Tresp, and Yunpu Ma. LLaVA steering: Visual instruction tuning with 500x fewer parameters through modality linear representation-steering. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15230–15250. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.739. URL `https://aclanthology.org/2025.acl-long.739/`.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022. URL `https://arxiv.org/abs/2210.11416`.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven C. Hoi. InstructBLIP: Towards general-purpose vision-language models with instruction tuning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 49250–49267, 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/9a6a435e75419a836fe47ab6793623e6-Abstract-Conference.html`.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115, 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/1feb87871436031bdc0f2beaa62a049b-Abstract-Conference.html`.

Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, Dahua Lin, and Kai Chen. VLMEvalKit: An open-source toolkit for evaluating large multi-modality models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 11198–11201. Association for Computing Machinery, 2024. doi: 10.1145/3664647.3685520. URL `https://doi.org/10.1145/3664647.3685520`.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the V in VQA matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6904–6913, 2017. doi: 10.1109/CVPR.2017.670. URL https://openaccess.thecvf.com/content_cvpr_2017/html/Goyal_Making_the_v_CVPR_2017_paper.html.

Danna Gurari, Qing Li, Abigale J. Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P. Bigham. VizWiz grand challenge: Answering visual questions from blind people. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3608–3617, 2018. doi: 10.1109/CVPR.2018.00380. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Gurari_VizWiz_Grand_Challenge_CVPR_2018_paper.html.

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. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Drew A. Hudson and Christopher D. Manning. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6700–6709, 2019. doi: 10.1109/CVPR.2019.00686. URL https://openaccess.thecvf.com/content_CVPR_2019/html/Hudson_GQA_A_New_Dataset_for_Real-World_Visual_Reasoning_and_Compositional_CVPR_2019_paper.html.

Dongfu Jiang, Xuan He, Huaye Zeng, Cong Wei, Max W. F. Ku, Qian Liu, and Wenhu Chen. MANTIS: Interleaved multi-image instruction tuning. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=skLtdUVaJa.

Dongxu Li, Junnan Li, Hung Le, Guangsen Wang, Silvio Savarese, and Steven C. H. Hoi. LAVIS: A one-stop library for language-vision intelligence. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 31–41. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-demo.3. URL https://aclanthology.org/2023.acl-demo.3/.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34892–34916, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0e-Abstract-Conference.html.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 26296–26306, 2024a. URL https://openaccess.thecvf.com/content/CVPR2024/html/Liu_Improved_Baselines_with_Visual_Instruction_Tuning_CVPR_2024_paper.html.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 2024b. URL https://proceedings.mlr.press/v235/liu24bn.html.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *Advances in Neural Information Processing Systems*, volume 35, pp. 2507–2521, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/11332b6b6cf4485b84afadb1352d3a9a-Abstract-Conference.html.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, Benjamin Bossan, and Marian Tietz. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022. Software repository.

Oumi Community. Oumi: An open, end-to-end platform for building large foundation models. https://github.com/oumi-ai/oumi, 2025. Software repository.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards VQA models that can read. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8317–8326, 2019. doi: 10.1109/CVPR.2019.00851. URL https://openaccess.thecvf.com/content_CVPR_2019/html/Singh_Towards_VQA_Models_That_Can_Read_CVPR_2019_paper.html.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. VL-Adapter: Parameter-efficient transfer learning for vision-and-language tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5227–5237, 2022. URL https://openaccess.thecvf.com/content/CVPR2022/html/Sung_VL-Adapter_Parameter-Efficient_Transfer_Learning_for_Vision-and-Language_Tasks_CVPR_2022_paper.html.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers reinforcement learning. https://github.com/huggingface/trl, 2020. Software repository.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gEZrGCozdqR.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6/.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. ReFT: Representation finetuning for language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 63908–63962, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/75008a0fba53bf13b0bb3b7bff986e0e-Abstract-Conference.html.

Zhiyang Xu, Chao Feng, Rulin Shao, Trevor Ashby, Ying Shen, Di Jin, Yu Cheng, Qifan Wang, and Lifu Huang. Vision-Flan: Scaling human-labeled tasks in visual instruction tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 15271–15342. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.905. URL https://aclanthology.org/2024.findings-acl.905/.

Kaichen Zhang, Bo Li, Peiyuan Zhang, Fanyi Pu, Joshua Adrian Cahyono, Kairui Hu, Shuai Liu, Yuanhan Zhang, Jingkang Yang, Chunyuan Li, and Ziwei Liu. LMMs-Eval: Reality check on the evaluation of large multimodal models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 881–916. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.51. URL https://aclanthology.org/2025.findings-naacl.51/.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. SWIFT: A scalable lightweight infrastructure for fine-tuning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(28):29733–29735, 2025. doi: 10.1609/aaai.v39i28.35383. URL https://ojs.aaai.org/index.php/AAAI/article/view/35383.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 400–410.

Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-demos.38. URL https://aclanthology.org/2024.acl-demos.38/.

Zhihan Zhou, Feng Hong, Jiaan Luo, Yushi Ye, Jiangchao Yao, Dongsheng Li, Bo Han, Ya Zhang, and Yanfeng Wang. Learning to instruct for visual instruction tuning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 47382–47409, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/43c18853329c7504996b255252b6cb1f-Abstract-Conference.html.

APPENDIX

A BUILT-IN METHODS AND EXPERIMENTAL COVERAGE

Table 4 lists the seven standalone methods evaluated on Qwen2.5-VL-3B-Instruct. Other backbones and method combinations are outside the reported experiments.

Table 4: Current built-in methods and their experimental coverage. A check mark indicates complete training and evaluation on TextVQA, VizWiz, and ScienceQA (image only), using Qwen2.5-VL-3B-Instruct.

Method	**Primary mechanism**	**Qwen2.5-VL-3B**
LoRA	Weight adaptation	✓
QLoRA	Quantized weight adaptation	✓
DoRA	Weight decomposition	✓
VL-Adapter	Inserted bottleneck modules	✓
ReFT	Representation intervention	✓
MoReS	Representation intervention	✓
L2T	Alternative token supervision	✓

B QUICK START: MORES TRAINING AND EVALUATION

This example trains MoReS on eight TextVQA samples and evaluates eight validation samples with Qwen2.5-VL-3B-Instruct. It checks the workflow, not benchmark performance. Run from the source repository after installing with `python -m pip install -e .`; a compatible CUDA GPU and access to the model and dataset are required.

Run the supplied configurations. The following files are included under `examples/`. The built-in MoReS recipe intervenes on the first four and last five visual tokens in every language layer.

Training: `train.yaml`

```
model: qwen25vl_3b_instruct
dataset: lmms-lab/textvqa
method: mores
batch_size: 1
max_length: 1536
max_samples: 8
output_dir: experiments/demo/checkpoint
```

Evaluation: `eval.yaml`

```
experiment:
  name: demo
  base_dir: experiments
eval:
  source: trained
  dataset_name: lmms-lab/textvqa
  max_samples: 8
  sample_seed: 42
  max_new_tokens: 16
```

```
python -m vlmintune.training --config examples/train.yaml
python -m vlmintune.eval --config examples/eval.yaml
```

Outputs and configuration. Under `experiments/demo/`, training writes `checkpoint/` and evaluation writes scores, predictions, sample IDs, and a log to `eval_trained/`. Select another built-in method by changing `method`; use a separate output directory and matching evaluation experiment name for each run. Set `max_samples: 0` for full data. Method-specific settings, such as intervention positions, are changed in method code (Appendix D).

C Data Processing: From a Record to a Training Batch

Dataset selection. Each built-in adapter supplies parsing, default splits, and a scoring rule (Table 5). Custom dataset imports are not supported by the public configuration. Adapter availability does not imply experimental coverage in this paper.

Table 5: Built-in dataset identifiers and scoring rules.

Dataset	Configuration name	Scoring rule
TextVQA	`lmms-lab/textvqa`	VQA accuracy
VQAv2	`pingzhili/vqa_v2`	VQA accuracy
VizWiz	`ebrukilic/vizwiz_vqa_dataset`	VQA accuracy
GQA	`Mineru/GQA`	Normalized exact match
ScienceQA (image)	`scienceqa_image`	Normalized exact match

A shared sample. The adapter converts each record into a `CanonicalSample`. Below is an illustrative TextVQA conversion; `pil_image` is a loaded image, and only relevant sample fields are shown.

```
row = {
    "question_id": "demo-1", "image": pil_image,
    "question": "What word is written on the sign?",
    "answers": ["stop", "stop", "STOP"],
}
# The built-in adapter produces:
CanonicalSample(
    id="demo-1", image_path="<in_memory>",
    question="What word is written on the sign?",
    train_answer="stop", eval_answers=["stop", "stop", "STOP"],
    metadata={"_pil_image": pil_image},
)
```

Formatting and supervision. The backbone's chat template formats the image, question, and answer. The processor builds text and visual inputs; prompt labels are masked so that the default loss uses response tokens. MoReS additionally uses an `intervention_mask` to select visual positions. Evaluation uses the prompt alone and retains reference answers for scoring.

D MODIFYING MORES

A new method implements `TrainingMethod` or subclasses an existing method. The interface defines backbone modifications, trainable parameters, loss, and saving/loading; optional overrides change runtime inputs and supervision. The `examples/my_mores.py` variant below inherits MoReS and selects only the first visual token during training and inference.

```
from vlmintune.training.methods.mores import MoReSMethod

class MyMoReS(MoReSMethod):
    name = "my_mores"
    display_name = "MoReS first visual token"

    def prepare_model_impl(self, model, processor, model_spec):
        self.image_token_id = int(model.config.image_token_id)
        return super().prepare_model_impl(model, processor, model_spec)

    def token_mask(self, input_ids):
        visual = input_ids == self.image_token_id
        return visual & (visual.long().cumsum(dim=-1) == 1)

    def build_forward_batch(self, batch):
        mask = self.token_mask(batch["input_ids"])
        batch = {**batch, "intervention_mask": mask}
        return super().build_forward_batch(batch)

    def forward_pre_hook(self, module, args, kwargs):
        super().forward_pre_hook(module, args, kwargs)
        if not module.training and self.current_intervention_mask is not None:
            self.current_intervention_mask = self.token_mask(kwargs["input_ids"])
```

Register and run. Copy the file into `src/vlmintune/training/methods/`. Add the following after the existing mapping in `training/methods/registry.py`:

```
from vlmintune.training.methods.my_mores import MyMoReS
_TRAINING_METHODS["my_mores"] = MyMoReS
```

The supplied configurations set `method: my_mores`. Results are saved separately under `experiments/my_mores_demo/`:

```
python -m vlmintune.training --config examples/train_my_mores.yaml
python -m vlmintune.eval --config examples/eval_my_mores.yaml
```

The variant reuses MoReS's modules, loss, and saving/loading. It constructs its own mask because a new method name does not activate the built-in MoReS preprocessing path. Keep the class and registry entry available for evaluation. This is a usage example, not a reported experimental result.

E CUSTOM EVALUATION

The YAML command uses the built-in evaluator. To customize input processing or generation, pass an object with `prepare_eval_input()` and `generate()` to the Python runner. This example adds a short-answer instruction and inherits generation from `LocalMethod`, using the MoReS model trained in Appendix B.

```
from dataclasses import replace
from vlmintune.eval.method import LocalMethod
from vlmintune.eval.run import evaluate_dataset, parse_eval_target

class ShortAnswerEvaluation(LocalMethod):
    def prepare_eval_input(self, sample):
        modified = replace(
            sample,
            question=sample.question + " Answer with a short phrase.",
        )
        return super().prepare_eval_input(modified)

checkpoint = "experiments/demo/checkpoint"
method = ShortAnswerEvaluation.from_checkpoint(
    "qwen25vl_3b_instruct",
    checkpoint_path=checkpoint,
)
target = parse_eval_target({
    "source": "trained",
    "dataset_name": "lmms-lab/textvqa",
    "max_samples": 8,
    "sample_seed": 42,
    "max_new_tokens": 16,
})
summary = evaluate_dataset(
    method, target, "experiments/demo/eval_custom"
)
print(summary["metrics"])
```

The runnable version is `examples/custom_evaluation.py`. The runner writes predictions and sample IDs and returns aggregate metrics. This changes the prompt, not the scoring rule: scoring is dataset-defined, and adding a metric requires source changes rather than API registration. For method comparisons, use matching validation IDs and generation settings, and check completion, skipped samples, and independent model reloading.