

ASSEMBLY-R1: 3D ASSEMBLY REASONING VIA RL-BASED VISION LANGUAGE MODELS

Anonymous authors

Paper under double-blind review

1 APPENDIX

1.1 SUPERVISED FINE-TUNING (SFT) CONFIGURATIONS

The SFT training configurations are listed in Table 1. The fine-tuning was performed with the help of LlamaFactory Zheng et al. (2024). The dataset is structured in Alpaca format Taori et al. (2023) for training the model.

Parameter	Value
model_name_or_path	Qwen/Qwen2-VL-2B-Instruct
trust_remote_code	true
stage	sft
do_train	true
finetuning_type	full
freeze_vision_tower	false
freeze_multi_modal_projector	false
freeze_language_model	false
deepspeed	LLaMA-Factory/examples/deepspeed/ds_z3_config.json
dataset	FurniBench_train_shuffled_selected_15000
template	qwen2_vl
cutoff_len	20480
preprocessing_num_workers	16
dataloader_num_workers	4
output_dir	outputs/qwen2_vl-2b_512_15000/sft
logging_steps	25
save_steps	300
report_to	wandb
batch_size	8
learning_rate	5.0e-5
num_train_epochs	3
lr_scheduler_type	cosine
warmup_ratio	0.1
bf16	true
ddp_timeout	180000000
resume_from_checkpoint	null

Table 1: Supervised Fine-Tuning (SFT) & DeepSpeed training configurations.

1.2 GROUP RELATIVE POLICY OPTIMIZATION (GRPO) CONFIGURATIONS

The GRPO model fine-tuning configurations are listed in Table 2. The multi-GPU training benefits from DeepSpeed Rasley et al. (2020).

Parameter	Value
config_file	configs/zero2.yaml
model_name_or_path	Qwen/Qwen2-VL-2B-Instruct
dataset_name	FurniBench_train_shuffled_selected_15000
max_prompt_length	1024
max_completion_length	700
learning_rate	1.0e-6
batch_size	8
logging_steps	1
bf16	true
gradient_checkpointing	true
num_train_epochs	3
save_steps	300
save_only_model	true
report_to	wandb
compute_environment	LOCAL_MACHINE
distributed_type	DEEPSPEED
deepspeed_multinode_launcher	standard
zero_stage	2
zero3_init_flag	false
offload_optimizer_device	none
offload_param_device	none
mixed_precision	bf16
downcast_bf16	no
num_processes	8
num_machines	1
machine_rank	0
main_training_function	main
main_process_port	44326
rdzv_backend	static
same_network	true
use_cpu	false
tpu_use_cluster	false
tpu_use_sudo	false
tpu_env	[]

Table 2: GRPO & DeepSpeed training configuration.

1.3 DATASET - QUESTION REPRESENTATIVE EXPRESSIONS

FurniQA includes 15 distinct QA task types. To make the dataset more diverse, each task is associated with three representative question expressions, as illustrated in Table 3. When generating QA pairs for each assembly video frame, one of the three expressions for the corresponding question type is randomly selected.

Question Type	Representative Expressions
Single Part Recognition (MCQ)	What is the part labeled in {id}? Please identify the part labeled as {id}. Which part does the label {id} correspond to?
Part Set Completeness (YN)	Are the currently labeled parts sufficient to complete the assembly? Do the labeled parts cover everything needed for assembly? Are all necessary parts labeled for assembly?
Missing Part Recognition (MCQ)	What other parts are required to complete the assembly? Are there any parts not labeled that are needed? Which parts are still required to finish the assembly?
First Assemble Pair (MCQ)	Which two parts can be assembled first? Out of the listed pairs, which can be assembled at the beginning? Select the pair of parts that should be assembled first.
First Assemble Pair (YN)	Can I directly attach Part A to Part B? Are Part A and Part B ready to be connected now? Is it possible to assemble them together now?
Connection After Installation (MCQ)	What parts does Part A connect to after installation? After assembly, which parts will be connected to Part A? Select the parts that will be attached to Part A.
Disassemble First (MCQ)	Which parts can be disassembled first? Out of the listed parts, which can be removed first? Select the part(s) that should be taken apart first.
Object Recognition (MCQ)	What could be the type of furniture? What is the most likely furniture type? Which furniture category do these parts belong to?
Installation Completed (YN)	Is the installation completed? Has the assembly process finished? Are all parts fully assembled now?
Action Recognition (MCQ)	What is the user doing in this frame? Describe the action performed by the user. Which activity is the user engaged in now?
Action Recognition (YN)	Is the user manipulating a {part}? Is the user interacting with a {part}? Do you see the user handling a {part}?
Next Step Inference (MCQ)	What should the user do next to complete the installation? What is the next action required? Which step should be performed next?
Installation Preparation (MCQ)	What should the user do next to prepare? Which preparation is needed before continuing? What action should be taken before the next step?
Installation Assembly (MCQ)	What should the user do next to complete the assembly? Which assembly action comes next? What is the next step in the assembly process?
Ready for Installation (YN)	Are the {part} ready to be installed? Can the {part} be installed now? Is any step required before installing the {part}?

Table 3: Overview of all 15 question types in FurniQA with representative expressions. Each type has 3 variations to encourage language diversity. For easier evaluation, each question in the dataset comes with a list of options, either a list of different choices or Yes/No. To maintain clarity, the answer options are not shown in this table.

1.4 DATASET - STATISTICS OF FURNIQA

Table 4: Statistics of FurniQA, including the main category, sub-category, task type, and quantities of corresponding QA pairs.

Main Category	Sub Category	Type	Quantity
Part Recognition	Single Part Recognition	MCQ	176,903
	Part Set Completeness	YN	176,903
	Missing Part Recognition	MCQ	154,105
	Object Recognition	MCQ	154,105
Part Connectivity	First Assemble Pair	MCQ	3,050
	First Assemble Pair	YN	10,654
	Connection After Installation	MCQ	45,786
	First Dissemble Part	MCQ	22,798
General Assembly Understanding	Installation Completion	YN	176,903
	Action Recognition	MCQ	150,286
	Action Recognition	YN	176,903
	Next Step Inference	MCQ	176,903
	Installation Preparation	MCQ	107,972
	Installation Assembly	MCQ	45,655
	Ready For Installation	YN	35,969

1.5 MODEL PERFORMANCE COMPARISONS

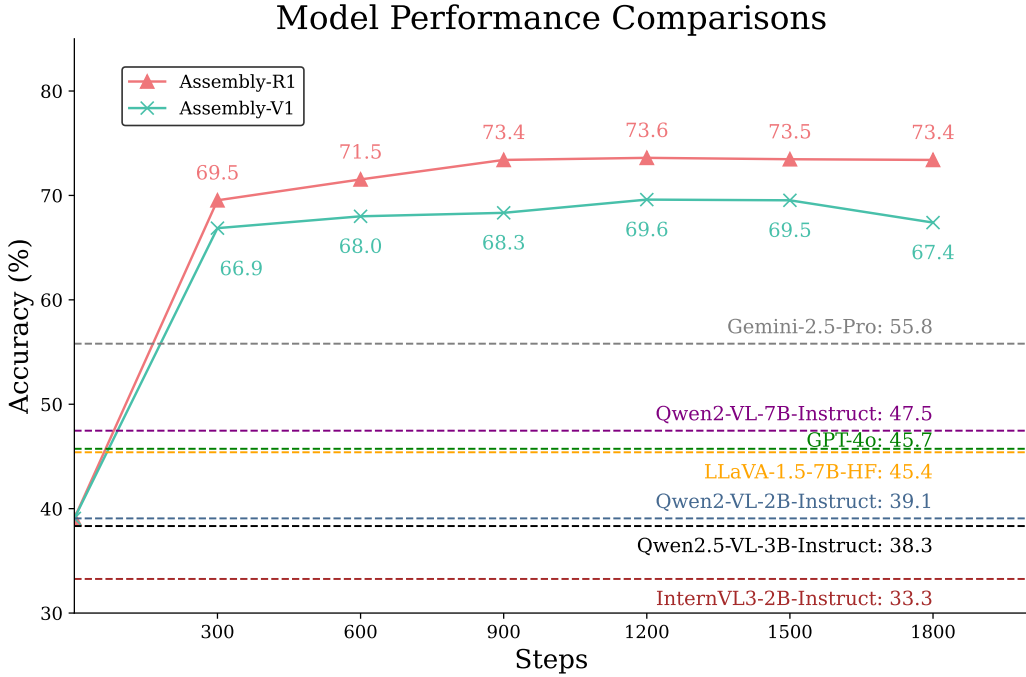


Figure 1: Performance comparison of various models on FurniBench. The green and red lines depict the progression of Assembly-V1 and Assembly-R1 performance throughout the training steps. Horizontal dashed lines indicate the benchmark performance of popular open-source vision-language models (VLMs).

1.6 ADDITIONAL ABLATION STUDIES

Reward Design Analysis. To validate our reward structure, we explored alternative designs intended to explicitly encourage reasoning via chain length and logical structure. We tested two configurations: (1) a *Length-Incentivized Reward* ($r_{len} \propto N_{tokens}$), and (2) a *Logic-Keyword Reward* combining clipped length incentives with bonuses for transition words (e.g., “therefore”, “next”). Both approaches led to severe reward hacking and training instability. The length-based reward caused a “verbosity explosion,” where completion lengths surged to over 400 tokens as the model learned to filibuster rather than reason. Similarly, the keyword incentive encouraged repetitive, long-winded generation to maximize keyword frequency, significantly increasing compute costs without improving accuracy. These findings confirm that heuristic-based rewards induce superficial verbosity. Consequently, we retained our final design—a simple format constraint combined with a strong outcome-based accuracy reward—which allows the model to self-discover optimal reasoning patterns without bias.

1.7 ADDITIONAL ANALYSIS - REWARD HACKING

The experiment is conducted on the classic Assembly-R1 design, where there was no Pure Coverage Reward (PCR).

Reward hacking occurs when an agent exploits flaws in the reward design to gain rewards through unintended behaviors Shen et al. (2025). Zhou et al. (2025) show that rewarding reasoning length can lead models to generate longer outputs without improving reasoning quality.

Although we don’t explicitly reward reasoning length, we still observe signs of reward hacking during training. We define the length reward hacking as a response that repeats with meaningless reasoning

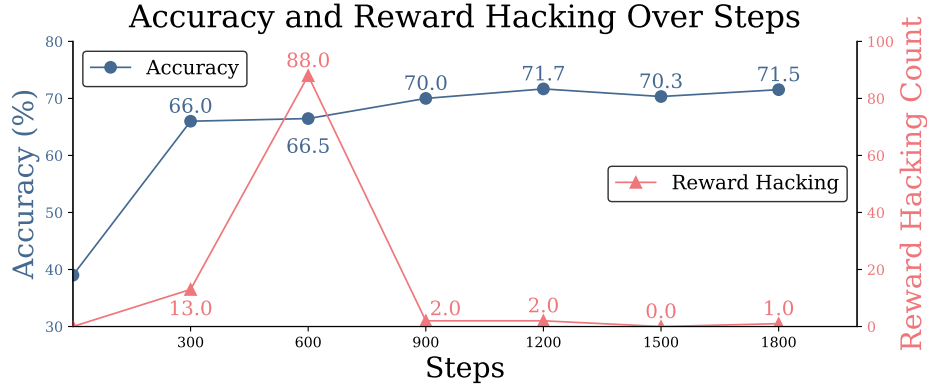


Figure 2: Visualization of model performance (blue line) and the number of reward hacking instances (red line) across training steps.

until reaching the output limit (1024) without a closing `</think>` tag. These incomplete responses suggest the model tries to exploit perceived reward signals without understanding the task.

As shown in Fig. 2, accuracy initially rises from 39.1% to 66.0% at 300 steps, with 13 reward hacking instances out of 1500 testing samples. This initial improvement in accuracy likely results from introducing the reasoning pattern, which the base model lacks. However, from 300 to 600 steps, hacking behavior increases while accuracy stagnates. In other words, the agent is optimizing for quantity over quality, generating longer but ineffective reasoning sequences. After 600 steps, rewards hacking diminishes and accuracy improves, reaching 71.7%. This is expected since our reward design does not explicitly favor long reasoning but rather meaningful thinking and accuracy. Eventually, the model shifts towards generating useful reasoning to gain more **Accuracy Reward**.

This observation highlights the importance of careful reward design in the RL-based fine-tuning framework for enhancing LLM/VLM reasoning capability.

1.8 ADDITIONAL ANALYSIS - AVERAGE RESPONSE LENGTH

The experiment is conducted on the classic Assembly-R1 design, where there was no Pure Coverage Reward (PCR).

Fig. 3 shows the relationship between the model’s performance and its average reasoning length over training steps. Importantly, we exclude samples flagged as reward hacking behavior when calculating the average reasoning length per response, so the statistics reflect only valid reasoning sequences.

Since our reward function does not explicitly encourage longer reasoning, the average length does not increase monotonically during training. Instead, it fluctuates between 17 and 35 words from step 300 onward. Notably, we can observe that the improvement in accuracy is usually accompanied by longer reasoning, while the periods of stable accuracy often show a decrease in reasoning response length.

In the early training phase, from the start to step 300, accuracy improves from 39.1% to 66.0%, with the average reasoning length reaching 39.1 tokens. Between steps 300 and 600, accuracy remains steady while the average reasoning length drops to 17.2 tokens. Then, from step 600 to step 1200, the accuracy climbs further to 71.1%, accompanied by an increment in average reasoning length to 35.1 tokens. Afterward, while the model keeps the accuracy around 71%, the average length decreases by over 10 tokens per response.

In summary, while the model is not directly rewarded for longer reasoning, it learns to use a more elaborate self-reflective reasoning chain to gain reward by improving answer accuracy. At the same time, it continues to refine its reasoning pattern to avoid unnecessary verbosity.

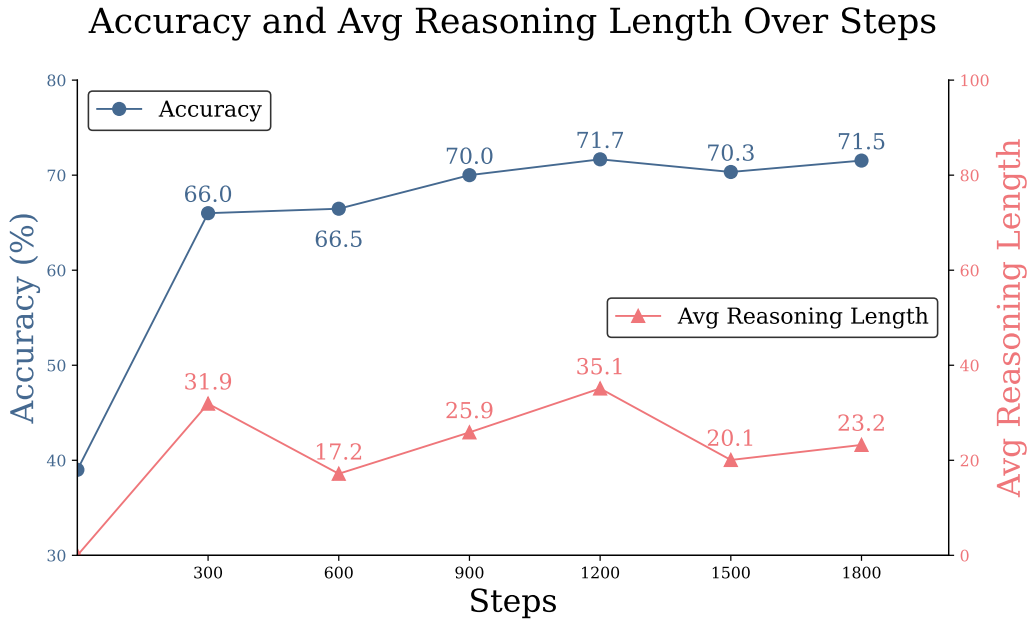


Figure 3: Visualization of model performance (blue line) and the average length of reasoning (red line) across training steps

REFERENCES

- Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3406703. URL <https://doi.org/10.1145/3394486.3406703>.
- Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, Ruochen Xu, and Tiancheng Zhao. Vlm-r1: A stable and generalizable r1-style large vision-language model, 2025. URL <https://arxiv.org/abs/2504.07615>.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models*. <https://crfm.stanford.edu/2023/03/13/alpaca.html>, 3(6):7, 2023.
- Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*, 2024.
- Hengguang Zhou, Xirui Li, Ruochen Wang, Minhao Cheng, Tianyi Zhou, and Cho-Jui Hsieh. R1-zero’s "aha moment" in visual reasoning on a 2b non-sft model, 2025. URL <https://arxiv.org/abs/2503.05132>.