Can't Get Better Without Making Mistakes: Improving Finetuning with Self-Generated Wrong Answers

Anonymous ACL submission

Abstract

Supervised fine-tuning (SFT) is a standard paradigm for adapting large language models (LLMs) to downstream domains. However, conventional SFT relies exclusively on gold standard labels, neglecting the rich supervisory signals of incorrect predictions. In this paper, we propose **Terran** (**T**uning **Err**or-**A**wareness from **N**egative predictions), a simple yet effective framework that leverages self-generated errors to better finetune the model. Terran utilizes temperature-based sampling to elicit incorrect responses from the base LLM, which are subsequently compiled into multiple-choice training formulations. This design enables the model to learn to distinguish gold standard solutions from plausible negative distractions within a standard SFT pipeline. Extensive experiments across three benchmarks using three diverse LLMs demonstrate that Terran consistently boosts performance over vanilla SFT. Furthermore, Terran outperforms reinforcement learning baselines like GRPO and significantly improves the model's inference-time self-consistency metrics.

1 Introduction

Supervised fine-tuning (SFT) has established itself as the primary paradigm for adapting large language models (LLMs) to new tasks and target domains (Wu et al., 2025). Traditional SFT methods predominantly focus on maximizing the likelihood of gold standard references. However, inspired by recent advancements in reinforcement learning (Guo et al., 2025) and self-correction (Kamoi et al., 2024), there is a growing consensus that the negative predictions generated by LLMs themselves can improve model performance.

One prominent line of research incorporates negative feedback via teacher-guided correction. These approaches leverage a more capable model to critique or correct erroneous predictions, appending the revised trajectories to the training data (Wang et al., 2024). Nevertheless, such methods heavily depend on the availability of an advanced teacher model, often degenerating the process into vanilla knowledge distillation. Another major direction centers on reinforcement learning (RL), such as Reinforcement Learning from AI Feedback (RLAIF) (Lee et al., 2023), which optimizes models using LLM-generated reward signals. Despite its theoretical appeal, RL-based training remains notoriously unstable and resource-intensive due to sparse rewards and high variance in long-horizon reasoning trajectories.

To address these limitations, we propose **Terran** (**T**uning **Err**or-**A**wareness from **N**egative predictions). Terran leverages the self-generated errors of LLMs as instructional materials to train them to distinguish gold standard answers from negative samples. Concretely, we employ temperature-based sampling to gather a mixture of correct and incorrect predictions from the base model. The incorrect outputs are then concatenated with the correct response to construct multiple-choice questions for subsequent fine-tuning. Operating entirely without an external teacher model, Terran retains the algorithmic simplicity and efficiency of standard SFT. Terran does not train the model to **generate** errors; instead, it optimizes the model's capacity to **discriminate** correct trajectories from plausible negative alternatives.

We conduct extensive experiments across three benchmarks utilizing three large language models of varying scales. The empirical results demonstrate that Terran outperforms vanilla SFT by a substantial margin in almost all configurations—even in scenarios where vanilla SFT fails to surpass the base model, highlighting Terran's robust effectiveness and generalizability. Furthermore, we benchmark Terran against state-of-the-art methods, including GRPO (Guo et al., 2025) and NAT,

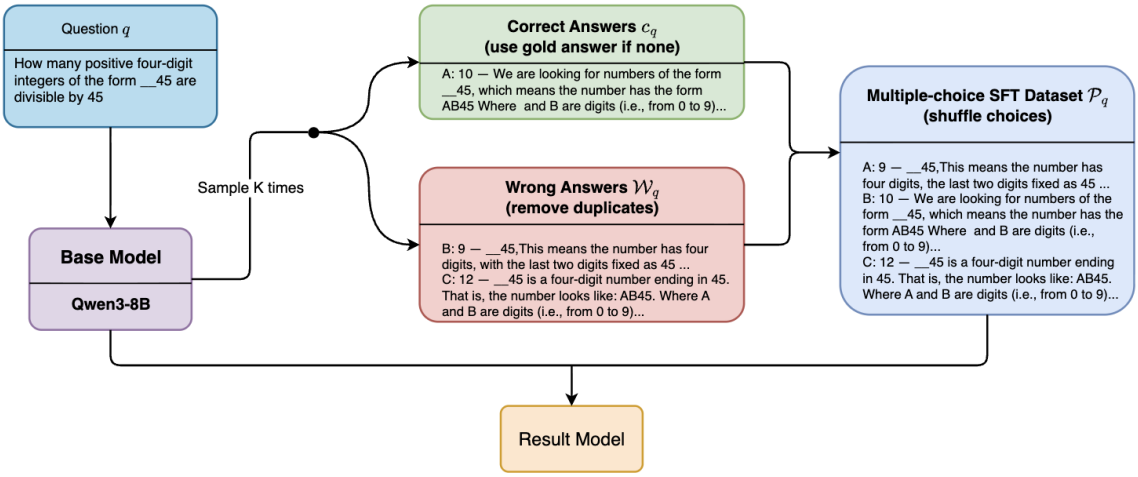

Figure 1: Overview of Terran. Given a question, the base model is sampled K times to produce correct and wrong answers. These answers are then converted into a multiple-choice SFT dataset, which is used to fine-tune the base model.

showcasing its superior performance and computational efficiency. Intriguingly, our analysis reveals that Terran also improves inference-time self-consistency metrics by a wide margin. By explicitly training the model to suppress negative distractors, Terran encourages correct reasoning paths to consistently stand out during generation.

2 Method

Overview. We propose Terran, a simple multiple-choice supervised fine-tuning framework built from the model's own sampled responses. For each question, Terran samples multiple responses, separates them into correct and incorrect answers, and trains the model to select the correct answer from a candidate set (Figure 1).

Given a question q, we first sample K responses from the base model. Each sampled response is evaluated via gold labels and split into two sets: the correct set $\mathcal{C}_q$ and the incorrect set $\mathcal{W}_q$. If $\mathcal{C}_q$ is non-empty, we select one response $c_q \in \mathcal{C}_q$ as the positive answer. Otherwise, we use the gold answer from the original dataset as c_q.

We construct a multiple-choice training example by pairing the positive answer with the sampled incorrect answers:

$$\mathcal{P}_q = \{c_q\} \cup \mathcal{W}_q.$$

The choices in $\mathcal{P}_q$ are randomly shuffled before being presented to the model. The model is then trained with supervised fine-tuning to select the correct answer c_q from the candidate set. Formally, for each training example, we optimize the negative log-likelihood of the correct choice:

$$\mathcal{L}_{\text{SFT}} = -\sum_{q \in \mathcal{D}} \log p_\theta(c_q \mid q, \mathcal{P}_q).$$

This procedure encourages the model to distinguish correct reasoning paths from plausible but incorrect sampled answers, rather than only imitating a single gold response.

3 Related Work

3.1 Learning from Errors

Recent work studies how LLMs can use mistakes or negative outputs as learning signals. One line of work uses errors at inference time without updating model parameters: Self-Refine (Madaan et al., 2023), Reflexion (Shinn et al., 2023), and ExpeL (Zhao et al., 2023) use self-feedback, external feedback, or comparisons between successful and failed trajectories to improve later responses. These methods show that failures can provide useful feedback, but they rely on the model or an evaluator to interpret the errors.

Another line of work incorporates negative data during training. Prior work uses negative reasoning data for arithmetic reasoning (Li et al., 2023), language-agent fine-tuning (Wang et al., 2024), theorem proving (An et al., 2024), and out-of-domain reasoning (Tian et al., 2026). Recent work also connects negative feedback with reinforcement-learning-style objectives (Chen et al., 2025). Terran is related to this direction, but uses a different formulation: instead of imitating negative trajectories

Model	Method	MATH		GPQA		LiveCodeBench	
		Train	Test	Train	Test	Train	Test
Qwen3-8B	Base model	80.96	82.48	15.38	20.00	41.93	19.16
	Vanilla SFT	80.61	81.57	16.14	17.43	–	–
	Terran	**82.86**	**83.20**	**16.59**	**21.13**	**42.63**	**19.76**
Llama-3.1-8B-Instruct	Base model	67.80	50.59	6.56	6.15	17.95	8.98
	Vanilla SFT	63.41	46.99	6.75	8.16	–	–
	Terran	**73.77**	**50.92**	**7.46**	**8.62**	**19.07**	**10.77**
Qwen3.5-27B	Base model	–	–	44.11	54.54	66.76	50.89
	Vanilla SFT	–	–	43.31	53.09	–	–
	Terran	–	–	**46.74**	**54.59**	**67.18**	49.70

Table 1: Training and test accuracy across models and datasets. Qwen3.5-27B is evaluated only on settings where sufficient incorrect samples are available for constructing the multiple-choice training data. No Vanilla SFT results for LiveCodeBench due to the absence of official gold answers.

or optimizing a separate negative-aware objective, we turn correct and incorrect sampled answers into a multiple-choice supervision signal.

3.2 Finetuning Methods for LLMs

Supervised fine-tuning is widely used to adapt LLMs to instruction following and reasoning tasks, often by training on gold answers or correct reasoning traces. However, positive-only fine-tuning discards information from plausible but incorrect generations. Parameter-efficient methods such as LoRA (Hu et al., 2021) reduce the cost of fine-tuning large models, while reinforcement-learning methods such as GRPO (Shao et al., 2024) optimize models with reward signals but are typically more expensive than SFT. Terran keeps the simplicity of supervised fine-tuning, but changes the training format from imitating a single target answer to selecting the correct answer from a candidate set containing both correct and incorrect model-generated responses.

4 Experiments

In this section, we describe the experimental setup, including the models, datasets, training details, evaluation protocol, and main results.

4.1 Experimental Setup

We evaluate Terran on several instruction-tuned language models from different families and scales. Our main models are Qwen3-8B (Yang et al., 2025) and Llama-3.1-8B-Instruct (Dubey et al., 2024). We also include Qwen3.5-27B (Qwen Team, 2026) on a subset of tasks to test whether Terran remains effective for a stronger model.

We conduct experiments on three reasoning and coding benchmarks: MATH (Hendrycks et al., 2021), GPQA (Rein et al., 2023), and LiveCodeBench (Jain et al., 2024). These benchmarks cover mathematical reasoning, scientific reasoning, and code generation. Details about train/test splits, training hyperparameters, and evaluation protocols are provided in Appendix A.

4.2 Overall Results

Table 1 shows the main results across models and datasets. Terran generally improves over the base model on both training and test sets. For Qwen3-8B, it improves performance on MATH, GPQA, and LiveCodeBench. For Llama-3.1-8B-Instruct, it also yields consistent gains, especially on MATH and LiveCodeBench.

Vanilla SFT does not show the same trend. On MATH, it slightly reduces accuracy for both Qwen3-8B and Llama-3.1-8B-Instruct, suggesting that simply fine-tuning on gold answers is less effective than training the model to distinguish correct answers from sampled wrong answers. Terran also improves Qwen3.5-27B on GPQA, indicating that the proposed training signal remains useful for stronger models.

4.3 Comparison with Other Works

GROP. Since Terran samples 10 responses per question, we can also apply GRPO (Shao et al., 2024) on questions with both correct and incorrect samples. For Qwen3-8B on MATH, GRPO reaches 82.46 training accuracy and 82.88 test accuracy, giving only small gains over the base model while still underperforming Terran. It is also substantially more expensive to train.

NAT. We also compare Terran with Negative-Aware Training (NAT) (Wang et al., 2024), a closely related fine-tuning method that uses both positive and negative trajectories. Following NAT, we construct a baseline from the same sampled responses used by Terran and split the responses

Metric	Base (Train)	Terran (Train)	Base (Test)	Terran (Test)
sample-level accuracy	81.53	**83.34 (+1.81)**	82.05	**83.35 (+1.30)**
majority-vote accuracy	85.68	**87.23 (+1.55)**	85.94	**87.13 (+1.19)**
any-correct rate	90.78	**91.54 (+0.76)**	91.50	**91.76 (+0.26)**
all-correct rate	66.83	**69.43 (+2.60)**	68.16	**68.85 (+0.69)**
all-wrong rate	9.22	**8.46 (-0.76)**	8.50	**8.24 (-0.26)**
average consistency	89.62	**90.92 (+1.30)**	90.28	**91.20 (+0.92)**

Table 2: Self-consistency metrics for Qwen3-8B on the MATH dataset, comparing the base model and Terran. Values are reported as percentages without the percent sign, and numbers in parentheses indicate the change over the base model. Definitions of the metrics are provided in Appendix A.5.

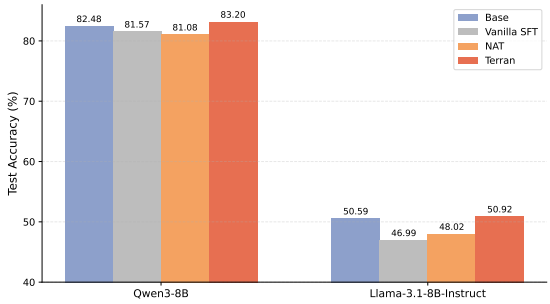

Figure 2: Test accuracy on MATH for the base model, vanilla SFT, NAT, and Terran. NAT underperforms the base model on both backbones, while Terran improves over the base model.

into correct and incorrect sets according to the reference answers. We then convert them into NAT-style training instances, where the model is conditioned to generate either a correct or an incorrect trajectory. Figure 2 compares NAT with the base model and Terran on MATH. NAT underperforms the base model on both backbones: it obtains 81.08 test accuracy for Qwen3-8B and 48.02 for Llama-3.1-8B-Instruct, compared with 82.48 and 50.59 for the corresponding base models. In contrast, Terran improves over the base model for both Qwen3-8B and Llama-3.1-8B-Instruct, reaching 83.20 and 50.92 test accuracy, respectively.

This gap may come from the different training objectives. NAT was originally designed for language-agent settings, where negative trajectories can teach the model about failed actions and observations. In our answer-level reasoning setting, however, negative trajectories are substantially longer than positive ones, so a large fraction of the token-level loss comes from incorrect reasoning traces. Terran avoids training the model to generate wrong answers; instead, it directly trains the model to select the correct answer from plausible incorrect candidates.

4.4 Analysis on Error-Correction

We analyze 573 difficult questions that the base model answers incorrectly in all 10 samples. Terran solves 19 of them with a single deterministic sample, and also reduces repeated wrong answers from 737 to 608. This suggests that Terran can correct some persistent base-model errors and avoid previously observed mistakes. A representative case study is provided in Appendix A.4.

4.5 Terran Improves Self-Consistency

Terran's sampled responses also enable self-consistency analysis (Wang et al., 2022). As shown in Table 2, Terran improves majority-vote accuracy, any-correct rate, all-correct rate, and average consistency, while reducing the all-wrong rate. These gains appear on both training and test data, suggesting that Terran makes model outputs more stable and less likely to repeat incorrect answers.

5 Conclusion

We propose Terran, a multiple-choice supervised fine-tuning framework that trains models to distinguish correct responses from their own sampled errors. By constructing candidate sets from correct and incorrect model-generated answers, Terran provides a simple way to use negative outputs without training the model to imitate them. Experiments on MATH, GPQA, and LiveCodeBench show that Terran generally improves over base models and vanilla SFT, and our analysis shows gains in self-consistency and error correction.

Future work can extend Terran in two directions. First, Terran can be used as an initialization before reinforcement learning to examine whether better candidate-level supervision improves later RL training. Second, although we focus on answer-level reasoning tasks, the same idea may be applied to agent tasks by constructing multiple-choice supervision over generated trajectories.

Limitations

Terran relies on the availability of informative incorrect samples. When the base model already performs very well on a dataset, it can be difficult to collect enough wrong answers for constructing useful multiple-choice training examples. For example, we do not apply Terran to Qwen3.5-27B on MATH because the model produces too few incorrect answers for this dataset. This suggests that Terran is most applicable when the base model has a moderate error rate, leaving enough room for learning from its sampled mistakes.

Another limitation is that our current experiments focus on datasets with gold answers or reliable automatic evaluation. The construction of Terran requires separating sampled responses into correct and incorrect sets, which is straightforward for benchmarks such as MATH and GPQA. However, for open-ended tasks or agent settings without gold answers, this separation may require additional verifiers, human annotation, or environment-based feedback. The effectiveness of Terran in such settings remains an important direction for future work.

Acknowledgments

This document has been adapted by Steven Bethard, Ryan Cotterell and Rui Yan from the instructions for earlier ACL and NAACL proceedings, including those for ACL 2019 by Douwe Kiela and Ivan Vulić, NAACL 2019 by Stephanie Lukin and Alla Roskovskaya, ACL 2018 by Shay Cohen, Kevin Gimpel, and Wei Lu, NAACL 2018 by Margaret Mitchell and Stephanie Lukin, BibTEX suggestions for (NA)ACL 2017/2018 from Jason Eisner, ACL 2017 by Dan Gildea and Min-Yen Kan, NAACL 2017 by Margaret Mitchell, ACL 2012 by Maggie Li and Michael White, ACL 2010 by Jing-Shin Chang and Philipp Koehn, ACL 2008 by Johanna D. Moore, Simone Teufel, James Allan, and Sadaoki Furui, ACL 2005 by Hwee Tou Ng and Kemal Oflazer, ACL 2002 by Eugene Charniak and Dekang Lin, and earlier ACL and EACL formats written by several people, including John Chen, Henry S. Thompson and Donald Walker. Additional elements were taken from the formatting instructions of the *International Joint Conference on Artificial Intelligence* and the *Conference on Computer Vision and Pattern Recognition*.

References

Chenyang An, Zhibo Chen, Qihao Ye, Emily First, Letian Peng, Jiayun Zhang, Zihan Wang, Sorin Lerner, and Jingbo Shang. 2024. Learn from failure: Fine-tuning llms with trial-and-error data for intuitionistic propositional logic proving. *arXiv preprint arXiv:2404.07382*.

Huayu Chen, Kaiwen Zheng, Qinsheng Zhang, Ganqu Cui, Yin Cui, Haotian Ye, Tsung-Yi Lin, Ming-Yu Liu, Jun Zhu, and Haoxiang Wang. 2025. Bridging supervised learning and reinforcement learning in math reasoning. *arXiv preprint arXiv:2505.18116*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

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. *arXiv preprint arXiv:2106.09685*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Kimi Team. 2026. Kimi k2.5: Visual agentic intelligence.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and 1 others. 2023. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Bin Sun, Xinglin Wang, Heda Wang, and Kan Li.

2023. Turning dust into gold: Distilling complex reasoning capabilities from llms by leveraging negative data. *arXiv preprint arXiv:2312.12832*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

Xueyun Tian, Minghua Ma, Bingbing Xu, Nuoyan Lyu, Wei Li, Heng Dong, Zheng Chu, Yuanzhuo Wang, and Huawei Shen. 2026. Learning from mistakes: Negative reasoning samples enhance out-of-domain generalization. *arXiv preprint arXiv:2601.04992*.

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. 2024. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Xiao-Kun Wu, Min Chen, Wanyi Li, Rui Wang, Limeng Lu, Jia Liu, Kai Hwang, Yixue Hao, Yanru Pan, Qingguo Meng, and 1 others. 2025. Llm fine-tuning: Concepts, opportunities, and challenges. *Big Data and Cognitive Computing*, 9(4):87.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. *arXiv preprint arXiv:2308.10144*.

A Appendix

A.1 Dataset Details

We conduct experiments on three reasoning and coding benchmarks. MATH (Hendrycks et al., 2021) is a mathematical reasoning benchmark containing competition-style math problems. GPQA (Rein et al., 2023) is a challenging question-answering benchmark designed to evaluate expert-level reasoning in science domains. LiveCodeBench (Jain et al., 2024) evaluates code generation and programming problem-solving ability.

For MATH, we use the official training and test splits. For GPQA, we use GPQA-main as the training set and GPQA-diamond as the test set. For LiveCodeBench, we use release-v4 as the training set and use the problems in release-v5 that are not included in release-v4 as the test set.

A.2 Training Details

We train all models using supervised fine-tuning with cross-entropy loss. For optimization, we use AdamW (Loshchilov and Hutter, 2017) with a warmup ratio of 0.03 and a weight decay of 0.01. To reduce training cost, we apply LoRA (Hu et al., 2021) during fine-tuning, with the LoRA rank set to 16. Unless otherwise specified, all hyperparameters are kept the same across models and datasets. The detailed sampling and fine-tuning configurations are shown in Tables 3 and 4.

A.3 Evaluation Details

For MATH, we require the model to output the final answer in the same format as the gold answer. We extract the final answer from the model output and compare it with the gold answer using exact matching after basic normalization, which removes minor formatting differences.

For GPQA, since answers may be expressed in different but semantically equivalent forms, we use Kimi2.5-Thinking (Kimi Team, 2026) as the evaluator model. Given the question, the gold answer, and the model output, the evaluator judges whether the model output is consistent with the gold answer. We report accuracy based on the evaluator's binary judgment.

For LiveCodeBench, we follow the official evaluation protocol (Jain et al., 2024) and use the LCB

evaluator. The generated code is executed against the provided test cases, and a solution is considered correct only if it passes all required tests.

A.4 Case Study

We perform a case study on question 4278: *"How many positive four-digit integers of the form __45 are divisible by 45?"*

> **Question 4278.** How many positive four-digit integers of the form __45 are divisible by 45?
>
> **Base model:** 9 ×
> Misses the case $a + b = 18$, especially the valid pair $(a, b) = (9, 9)$.
> **Base model:** 12 ×
> Overcounts by allowing invalid leading-zero digit pairs.
> **Terran:** 10
> Counts both $a+b = 9$ and $a+b = 18$, giving $9+1 = 10$.

Figure 3: Case study on question 4278. The base model makes systematic counting errors across samples, while Terran corrects the reasoning by separating the valid digit-sum cases.

A.5 Self-Consistency Metrics

For each question, we sample the model 10 times and evaluate whether each sampled answer is correct. The self-consistency metrics in Table 2 are defined as follows.

- **Sample-level accuracy** is the accuracy over all sampled responses.
- **Majority-vote accuracy** is the accuracy of the answer obtained by majority voting over the 10 sampled responses.
- **Any-correct rate** is the percentage of questions for which at least one sampled response is correct.
- **All-correct rate** is the percentage of questions for which all sampled responses are correct.
- **All-wrong rate** is the percentage of questions for which all sampled responses are incorrect.
- **Average consistency** measures how often the sampled responses agree with the majority answer for each question, averaged across questions.

A.6 Sampling Configuration

Table 3 summarizes the configuration used to generate sampled responses for constructing Terran training data. For each question, we sample $K = 10$ responses using the sampling profile. For evaluation, we use deterministic decoding.

Item	Value
Base model	Qwen/Qwen3-8B
Dataset	EleutherAI/hendrycks_math
Split	train
Levels	1, 2, 3, 4, 5
Sample size	6,216
Sample seed	42
Max new tokens	8,192
Max concurrency	8
Use chat template	true
Enable thinking	false
Sampling temperature	0.7
Sampling top-p	0.8
Samples per prompt	10
Evaluation temperature	0.0
Evaluation top-p	1.0
Evaluation samples per prompt	1
Python seed	42
NumPy seed	42
PyTorch seed	42

Table 3: Configuration for response sampling and deterministic evaluation.

A.7 Terran SFT Configuration

Table 4 summarizes the main configuration used for Terran supervised fine-tuning.

Item	Value
Model	Qwen/Qwen3-8B
Data type	bfloat16
Max model length	8,196
Generated data path	outputs/stage_c/generated_data.jsonl
Prompt template	prompts/multiple_choice_prompt.txt
System prompt	prompts/system_prompt_mc.txt
Assistant template	`{answer}`
Use math reasoning	false
Normalize final boxed answer	false
Enable thinking	false
Number of epochs	8
Learning rate	5.0×10^{-5}
Batch size per device	16
Gradient accumulation steps	1
Warmup ratio	0.03
Weight decay	0.01
LoRA rank	16
Logging steps	50
Save steps	200
Save total limit	2
Export name	math_stage_c_0022
Python seed	42
NumPy seed	42
PyTorch seed	42

Table 4: Configuration for Terran supervised fine-tuning.

A.8 Prompts

Figure 4 shows the prompts used for response sampling and Terran multiple-choice SFT.

Sampling Prompt

```
You are a mathematical reasoning assistant.
For each problem, you must:
1. Provide a clear, step-by-step logical derivation of the solution.
2. Use LaTeX for all mathematical expressions and formulas.
3. Place your final numerical or algebraic answer inside a \boxed{} command at the very end.

Example of final line:
The final answer is \boxed{42}
```

Terran Multiple-choice Prompt

```
You are given a math problem and multiple candidate solutions.

Problem:
{question}

Candidate Options:
{options}

Select the best option.
First output the option label on its own line.
Then provide the full reasoning process of the selected option.

Format:
<LABEL>
<PROCESS>

Example:
A
Because ...
```

Figure 4: Prompts used for response sampling and Terran multiple-choice SFT. The sampling prompt is used to generate candidate responses from the base model, while the multiple-choice prompt is used to train Terran to select the correct response from sampled candidates.

A.9 Use of AI Assistants

We used AI assistants for language polishing, LaTeX formatting, and code/debugging assistance during the writing and experimental process. All technical claims, experimental results, analyses, and final manuscript content were reviewed and verified by the authors.