STaR: Self-Taught Reasoner

CSE 576 — Small Project 1

Shashank Kumar Singh
ASU ID: 1235716991

October 15, 2025

Abstract

We reproduce the *STaR* (Self-Taught Reasoner) method on GSM8K dataset using Llama-3.2-3B-Instruct. Our pipeline comprises: (i) zero-shot chain-of-thought (CoT) generation on the training dataset; (ii) supervised fine-tuning (SFT) on examples with correct final answers; and (iii) STaR bootstrapping, which regenerates rationales conditioned on the gold answer for previously incorrect items and merges them into the SFT corpus. We evaluate using exact match (EM) on the GSM8K test set. On an A100 GPU (ASU Sol), accuracy increases from 38.27% (Zero-Shot) to 41.70% (Vanilla SFT) and 52.54% with STaR, absolute gains of +14.27 and +10.84 points over the baselines, respectively. These findings indicate that answer-conditioned rationale regeneration improves label quality and downstream performance on GSM8K dataset.

1 Setup & Environment

Hardware: ASU Sol cluster, A100 GPU. (Fig. 1);
Env: micromamba environment `star311`. (Fig. 2);
Model: Llama-3.2-3B-Instruct. (HuggingFace);
Data: GSM8K (`train/test` JSONL included).
All scripts and data are in the attached zip; model directories are excluded due to size.

2 Method Overview

We compare three regimes on GSM8K with Llama-3.2-3B-Instruct.

- **Zero-Shot CoT:** Generate a rationale and final answer per item; this is the baseline.
- **Vanilla SFT:** Keep training items whose zero-shot answer matches the gold; fine-tune on their (question, rationale, `FINAL_ANSWER`).
- **STaR:** For the remaining items, regenerate a rationale conditioned on the gold answer (a "hint"), merge these repaired examples with the vanilla set, and fine-tune.

Decoding & Evaluation. At test time we apply the same chat template with `add_generation_prompt=true` and left padding, decode only new tokens after the prompt, parse `FINAL_ANSWER`, and compute exact match (EM) on the GSM8K test split.

3 Prompts Used

3.1 Zero-Shot CoT generation (train-time)

Listing 1: `prompts/cot_gen.txt`

```
You are a careful math tutor. Solve step by step. Show your reasoning as a numbered list, then end with the final answer on a new line as:
FINAL_ANSWER: <number>

Q: {question}
A:
1)
```

3.2 Hinted rationale for STaR (train-time)

Listing 2: `prompts/cot_hint.txt`

```
You solved a similar problem before. Use the target final answer as a hint to guide a correct chain-of-thought. Show the reasoning as a numbered list and finish with:
FINAL_ANSWER: <number>

Question:
{question}

Target final answer:
{answer}

Now produce a correct, self-consistent rationale that leads to the target:
```

3.3 Evaluation prompt (test-time)

Listing 3: `prompts/sft_eval.txt`

```
Q: {question}
A:
1)
```

4 Implementation & Workflow

The zip contains:

- `data/` *(prepared)* `gsm8k_train.jsonl`, `gsm8k_test.jsonl`
- `prompts/` `cot_gen.txt`, `cot_hint.txt`, `sft_eval.txt`
- `runs/vanilla/` `sft.jsonl` (7,473 rows) built from CoT generations
- `runs/star_v1/` `sft.jsonl`, `sft_fixed.jsonl` (7,473 rows) bootstrapped dataset

- eval/ zeroshot.json, vanilla_class_final.json, star_class_v1.json
- scripts/01-05 (data, generation, SFT-building, bootstrapping, training, evaluation)

Data creation scripts/01_make_gsm8k.py : downloads GSM8K via HuggingFace, normalizes answers to their trailing number, and writes train/test JSONL.

Train-time CoT generation scripts/02_generate_train_cot.py : samples rationales using Llama-3.2-3B-Instruct ,writing runs/vanilla/gen_train.jsonl.

SFT dataset (vanilla) scripts/03_build_sft_from_cot.py : extracts rationale blocks and final answers; with -only_correct, it keeps only examples where predicted equals gold.

STaR bootstrapping scripts/03b_build_star_sft.py : finds items the vanilla pass got wrong, re-generates a rationale *conditioned on the gold answer* using prompts/cot_hint.txt, and writes merged SFT JSONL (all 7,473 rows).

SFT training scripts/04_train_sft_class.py : turns each example into a chat pair {prompt, response} (system/user → assistant). During training we only compute loss on the assistant's tokens (the model's output), not on the system/user text. The script uses mixed precision (bf16/fp16 if available) and a cosine-style learning-rate schedule.

Evaluation scripts/05b_infer_eval.py : builds the system + user prompt, enables add_generation_prompt=true, and decodes only the new tokens after the prompt to avoid echoing. It then extracts the value after FINAL_ANSWER: with a regex, normalizes it, compares it to the gold answer, and writes the metrics to JSON.

5 How to Reproduce on ASU Sol

Request a GPU shell

```
srun --partition=general --gres=gpu:a100:1 --cpus-per-task=4 --mem=24G \
    --time=04:00:00 --pty bash
```

Environment + HF login

```
eval "$SCRATCH/bin/micromamba shell hook -s bash"
micromamba activate $SCRATCH/conda/star311
export PYTHONPATH=$PWD:${PYTHONPATH:-}
# one-time
huggingface-cli login --add-to-git-credential
```

Pipeline commands

```
# 1) Prepare GSM8K
python scripts/01_make_gsm8k.py

# 2) Generate train-time CoT (vanilla)
python scripts/02_generate_train_cot.py \
  --model meta-llama/Llama-3.2-3B-Instruct \
  --train data/gsm8k_train.jsonl \
  --prompt prompts/cot_gen.txt \
  --out runs/vanilla/gen_train.jsonl --batch 8

# 3) Build vanilla SFT data
python scripts/03_build_sft_from_cot.py \
  --gen runs/vanilla/gen_train.jsonl \
  --sft_out runs/vanilla/sft.jsonl --only_correct

# 4) Train vanilla SFT
python scripts/04_train_sft_class.py \
  --base meta-llama/Llama-3.2-3B-Instruct \
  --data runs/vanilla/sft.jsonl \
  --out models/vanilla_sft_tuned --epochs 1 --bsz 1 \
  --grad_accum 8 --lr 2e-5

# 5) Build STaR SFT data (hinted regeneration for wrong items)
python scripts/03b_build_star_sft.py \
  --model meta-llama/Llama-3.2-3B-Instruct \
  --train data/gsm8k_train.jsonl \
  --gen runs/vanilla/gen_train.jsonl \
  --hint_prompt prompts/cot_hint.txt \
  --out runs/star_v1/sft.jsonl --batch 8

# 6) Train STaR SFT
python scripts/04_train_sft_class.py \
  --base meta-llama/Llama-3.2-3B-Instruct \
  --data runs/star_v1/sft.jsonl \
  --out models/star_sft_tuned --epochs 1 --bsz 1 \
  --grad_accum 8 --lr 2e-5

# 7) Evaluate (EM on GSM8K test)
python scripts/05b_infer_eval.py \
  --model meta-llama/Llama-3.2-3B-Instruct \
  --data data/gsm8k_test.jsonl \
  --prompt prompts/sft_eval.txt \
  --out eval/zeroshot.json

python scripts/05b_infer_eval.py \
  --model models/vanilla_sft_tuned \
  --data data/gsm8k_test.jsonl \
  --prompt prompts/sft_eval.txt \
  --out eval/vanilla_class_final.json
```

```

python scripts/05b_infer_eval.py \
  --model models/star_sft_tuned \
  --data data/gsm8k_test.jsonl \
  --prompt prompts/sft_eval.txt \
  --out eval/star_class_v1.json
```

6 Results

From your `eval/` JSONs:

Method	EM (%)	Correct	n
Zero-Shot CoT	38.27	505	1319
Vanilla SFT	41.70	550	1319
STaR (ours)	**52.54**	693	1319
Gains: STaR − Zero-Shot = +14.27 points; STaR − Vanilla = +10.84 points.			

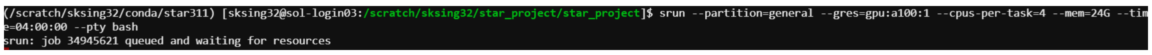
Observations (Zero-Shot → Vanilla SFT). Exact match (EM) rises from 38.27% (Zero-Shot CoT) to 41.70% with Vanilla SFT. The gain comes from training only on items the model already solved correctly, which yields clean (question, rationale, `FINAL_ANSWER`) pairs and stabilizes the model's formatting and reasoning style. However, this pipeline keeps mostly "easy/correct" cases and omits hard failures, so improvements taper: the model gets better at patterns it already handled, but coverage of difficult problem types remains limited.

Why STaR Helps. STaR closes that coverage gap by *repairing* previously wrong items: it regenerates a rationale conditioned on the gold answer and turns many failures into high-quality supervised examples. This boosts label density, reduces noise, and aligns the training distribution with test-time decoding (same base model, same style), which together drive a larger jump to 52.54% EM (+10.84 over Vanilla, +14.27 over Zero-Shot). Because all variants share the same evaluator (prompt-echo suppression and strict `FINAL_ANSWER` parsing), the gains reflect better training data and more reliable reasoning rather than scoring artifacts.

7 Limitations & Next Steps

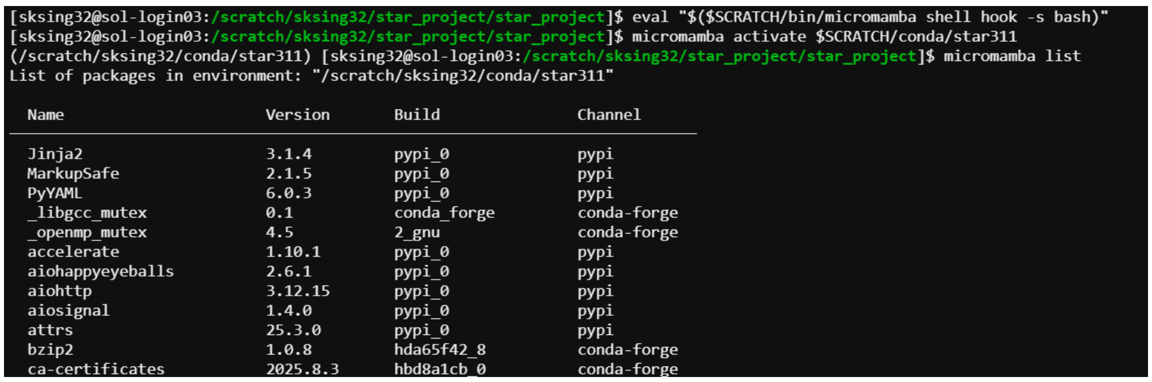
- Increase samples per item (self-consistency).
- Lengthen `max_new_tokens` for long arithmetic.
- Filter low-quality chains with verifier heuristics.

8 Figures (Screenshots)



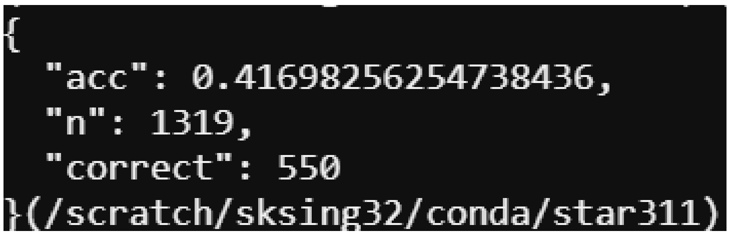
```
(/scratch/sksing32/conda/star311) [sksing32@sol-login03:/scratch/sksing32/star_project/star_project]$ srun --partition=general --gres=gpu:a100:1 --cpus-per-task=4 --mem=24G --tim
e=04:00:00 --pty bash
srun: job 34945621 queued and waiting for resources
```

Figure 1: GPU request on ASU Sol (A100).



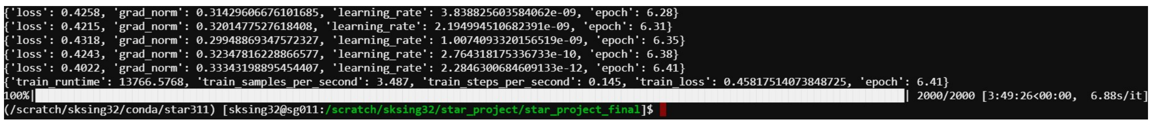
```
[sksing32@sol-login03:/scratch/sksing32/star_project/star_project]$ eval "$($SCRATCH/bin/micromamba shell hook -s bash)"
[sksing32@sol-login03:/scratch/sksing32/star_project/star_project]$ micromamba activate $SCRATCH/conda/star311
(/scratch/sksing32/conda/star311) [sksing32@sol-login03:/scratch/sksing32/star_project/star_project]$ micromamba list
List of packages in environment: "/scratch/sksing32/conda/star311"

  Name                      Version       Build               Channel
 ──────────────────────────────────────────────────────────────────────
  Jinja2                    3.1.4         pypi_0              pypi
  MarkupSafe                2.1.5         pypi_0              pypi
  PyYAML                    6.0.3         pypi_0              pypi
  _libgcc_mutex             0.1           conda_forge         conda-forge
  _openmp_mutex             4.5           2_gnu               conda-forge
  accelerate                1.10.1        pypi_0              pypi
  aiohappyeyeballs          2.6.1         pypi_0              pypi
  aiohttp                   3.12.15       pypi_0              pypi
  aiosignal                 1.4.0         pypi_0              pypi
  attrs                     25.3.0        pypi_0              pypi
  bzip2                     1.0.8         hda65f42_8          conda-forge
  ca-certificates           2025.8.3      hbd8a1cb_0          conda-forge
```

Figure 2: Micromamba environment `star311` with core packages.



```
{
  "acc": 0.41698256254738436,
  "n": 1319,
  "correct": 550
}(/scratch/sksing32/conda/star311)
```

Figure 3: Vanilla SFT EM summary.



```
{'loss': 0.4258, 'grad_norm': 0.31429606676101685, 'learning_rate': 3.838825603584062e-09, 'epoch': 6.28}
{'loss': 0.4215, 'grad_norm': 0.3201477527618408, 'learning_rate': 2.194994510682391e-09, 'epoch': 6.31}
{'loss': 0.4318, 'grad_norm': 0.29948869347572327, 'learning_rate': 1.0074093320156519e-09, 'epoch': 6.35}
{'loss': 0.4243, 'grad_norm': 0.32347816228866577, 'learning_rate': 2.764318175336733e-10, 'epoch': 6.38}
{'loss': 0.4022, 'grad_norm': 0.33343198895454407, 'learning_rate': 2.2846300684609133e-12, 'epoch': 6.41}
{'train_runtime': 13766.5768, 'train_samples_per_second': 3.487, 'train_steps_per_second': 0.145, 'train_loss': 0.45817514073848725, 'epoch': 6.41}
100%|████████████████████████████████████████████████████████████████████████████████| 2000/2000 [3:49:26<00:00,  6.88s/it]
(/scratch/sksing32/conda/star311) [sksing32@sg011:/scratch/sksing32/star_project/star_project_final]$
```

Figure 4: STAR training log snippet (loss, grad_norm, throughput).