COMP4901B LLM ASSIGNMENT 3

Lam Yeung Kong Sunny 20857162 ykslam@connect.ust.hk

November 19, 2025

1 VLLM Inference Implementation

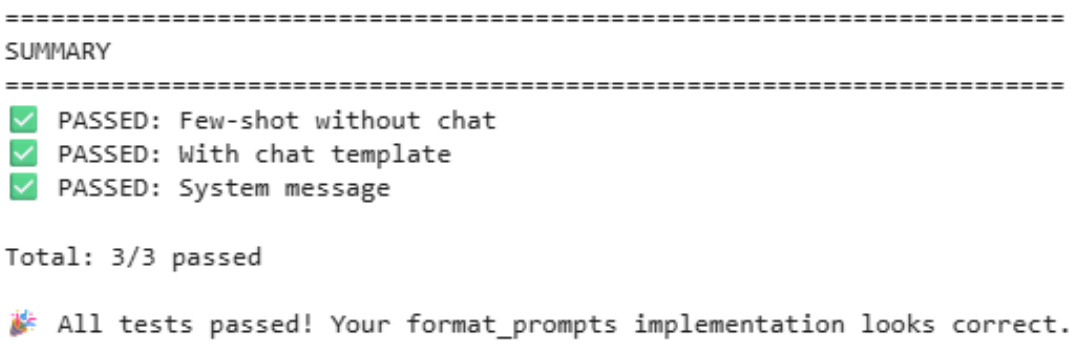


Figure 1: Terminal Output 2

- Brief explanation of the implementation approach:
  - `format_prompts`: For each single question from a list of questions, I format the question with few shot prompt and append the formatted prompt if chat template is not used or tokenizer is missing. Otherwise, I will first append the system message before the question if it need to be included and apply the chat template from tokenizer and append the formatted prompt.
  - `run_inference`: First, I construct the `llm` using `vllm.LLM` api with necessary attributes provided. Second, I create the `sampling_inference` to store the parameters which will be used using SamplingParams api. Lastly, I generate outputs using `llm` with formatted prompt retrieved by using `format_prompts` and `sampling_inference` as argument.
- Question Answer: $temperature > 0$ allows the multiple rollouts to produce different outputs with diversity. If $temperature = 0$, the token generated will be deterministic as the model will always pick the largest probability token. In this way, all the outputs from rollouts will be the same and it will be meaningless to use multiple rollouts without diversity.

2 Answer Verification Implementation

```
================================================================================
TEST 1: extract_solution()
================================================================================

Result: 9/9 passed
✅ PASSED: extract_solution()

================================================================================
TEST 2: compute_score()
================================================================================

Result: 8/8 passed
✅ PASSED: compute_score()

================================================================================
TEST 3: estimate_pass_at_k()
================================================================================

Result: 6/6 passed
✅ PASSED: estimate_pass_at_k()

================================================================================
SUMMARY
================================================================================
✅ PASSED: extract_solution()
✅ PASSED: compute_score()
✅ PASSED: estimate_pass_at_k()

Total: 3/3 functions passed

🎉 All tests passed! Your evaluation functions look correct.
```

Figure 2: Terminal Output 2

- Explanation of extraction logic: First, I use the regular expression `\\boxed\{([^}]+)\}`. to get the `\boxed{value}` and return the last one to fulfill the requirement. If non of the parts fulfill the requirement, I use the regular expression `(\-?[0-9\.\,]+)` to find the related value, otherwise return None. Lastly, I replace the `,.` syntax in the string before returning the solution.
- Explanation of scoring logic: First, I change type of `answer` and `ground_truth` to float and then convert back to string. In this way, both values will be float in string, helping the comparison to compare correctly. The function returns 1 if both are equal, otherwise return 0. Note that a try except block is used in case if `answer` and `ground_truth` encounter error during conversion and it will return 0 directly.
- Derivation:

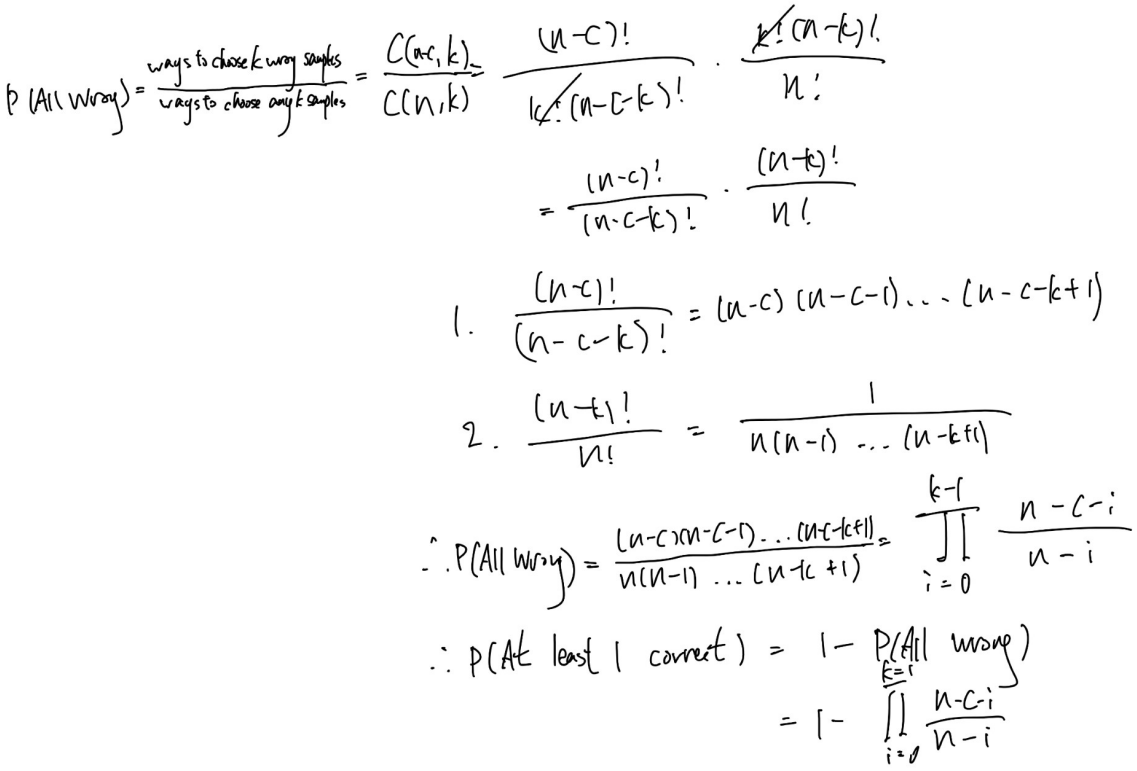


Figure 3: Derivation of pass@k formula

- Explanation of handling numerical stability: To avoid the overflow, loss of precision or other instability for large n, I use the product formulation according to 3, i.e. $\prod_{i=0}^{k-1}(n-c-i)/(n-i)$. Therefore, it prevents directly calculate the large intermediate factorial value and maintain stability.

3 LoRA Training Implementation

```python
# ================== TODO: Implement this method ==================
valid_targets = []
linear_mod_names = set()

for name, module in self.model.named_modules():
    # Get all linear modules
    if isinstance(module, nn.Linear):
        rank0_print(f"Found linear module: {name}")
        linear_mod_names.add(name.split(".")[-1])

# Determine model architecture
model_type = getattr(getattr(self.model, "config", None), "model_type", "").lower()
rank0_print(f"Model type detected: {model_type}")

if "llama" in model_type:
    attn_candidates = {"q_proj", "k_proj", "v_proj", "o_proj"}
    ffn_candidates = {"gate_proj", "up_proj", "down_proj"}
elif "mistral" in model_type:
    attn_candidates = {'k_proj', 'o_proj', 'q_proj', 'v_proj'}
    ffn_candidates = {'up_proj', 'gate_proj', 'down_proj'}
elif "opt" in model_type:
    attn_candidates = {"q_proj", "k_proj", "v_proj", "out_proj"}
    ffn_candidates = {"fc1", "fc2"}
elif "qwen" in model_type:
    attn_candidates = {"q_proj", "k_proj", "v_proj", "o_proj"}
    ffn_candidates = {"gate_proj", "up_proj", "down_proj"}
else:
    # helper sets of typical suffixes for attention and FFN layers
    attn_candidates = {"q_proj", "k_proj", "v_proj", "o_proj", "out_proj", "query_key_value", "proj"}
    ffn_candidates = {"gate_proj", "up_proj", "down_proj", "fc1", "fc2", "wi", "wo", "wi_proj", "wo_proj"}

user_spec = getattr(self.lora_args, "lora_target_modules", None)
if user_spec:
    rank0_print(f"Using user-specified LoRA target modules: {user_spec}")
    user_spec = set(user_spec.split(","))
    if "all" in user_spec:
        # Apply LoRA to all linear modules as long as they exist
        valid_targets = list(linear_mod_names)
    elif "attn" in user_spec:
        valid_targets = list(linear_mod_names.intersection(attn_candidates))
    elif "ffn" in user_spec:
        valid_targets = list(linear_mod_names.intersection(ffn_candidates))
    elif "attn_ffn" in user_spec:
        valid_targets = list(linear_mod_names.intersection(attn_candidates.union(ffn_candidates)))
    else:
        valid_targets = list(user_spec.intersection(linear_mod_names.intersection(attn_candidates.union(ffn_candidates))))
else:
    # Auto-select based on architecture
    valid_targets = list(linear_mod_names.intersection(attn_candidates.union(ffn_candidates)))
    rank0_print(f"Auto-selected LoRA target modules: {valid_targets}")
# ==================================================================
return valid_targets
```

Figure 4: `_resolve_lora_target_modules()` implementation

- Explanation of code: I used (self.model.named_modules()) to collect all linear layers and got their names. Then suggested by the question, I used (self.model.config.model_type) to detect architecture. Then, for each architecture, the attention and FNN modules' name were defined, which is suggested in the hugging face transformer source code. Finally, the returned list will only contain the modules that are contained in the model. If the user specifies (–lora_target_modules), only modules that fit the user choice will be chosen and put in the list to be returned.
- Question Answer: By applying LoRA to both attention and FFN, the model can adjust its focus of attention and refine the transformation in FNN stage for fitting the new task. In this way, both stage can be adapted to have better performance. However, if only attention is applied, it will only change the attention focus and cannot learn new representation to fit the new task in FNN state. Therefore, the adaptation will not be as good as applying to both sections.

4 Self-Training Execution

4.1 Training configuration summary

Model	Iteration	Rollouts	LoRA rank	lr
Qwen3-0.6B	1	8	64	2e-5

Table 1: Training configuration summary

4.2 Iteration metrics

Iteration	No. of correct solutions	pass@1	pass@4	pass@8	Training loss
0	9338	0.5805	0.8065	0.8755	-
1	10552	0.5944	0.7885	0.8423	0.272

Table 2: Iteration metrics

4.3 Screenshot of accuracy improvement across iterations

```
Loading data from ckpt/first-run/iteration_0/inference.jsonl
Loaded 16000 examples

Basic Evaluation Results:
Total examples: 16000
Correct answers: 9338
Overall Accuracy: 58.36%
```

Figure 5: Iteration 0

```
Loading data from results/model_iter_1-merged_20251115_060728_inference.jsonl
Loaded 10552 examples

Basic Evaluation Results:
Total examples: 10552
Correct answers: 6349
Overall Accuracy: 60.17%
```

Figure 6: Iteration 1

- Analysis: The self-training does improve the performance. By self reinforcing the reasoning step using the 9338 verified samples and training the adapter, the accuracy of the model improved from 0.5836% to 0.6017%. In addition, the passk generally increased around 0.1, indicating that the performance has slightly improved.

5 Final Evaluation and Analysis

5.1 Baseline Evaluation

Please refer to figure 5, which shows that the baseline accuracy is $58.36\% > 57\%$.

5.2 Trained Model Evaluation

```
Loading data from results/default/rollout1/model_iter_1-merged_20251115_070341_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 789
Overall Accuracy: 59.82%
```

(a) Default rollout 1 evaluation

```
Loading data from results/batch_256/rollout1/model_iter_1-merged_20251117_102749_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 799
Overall Accuracy: 60.58%

Results saved to results/batch_256/rollout1/model_iter_1-merged_20251117_102749_evaluation.jsonl
```

(b) Batch size 256 rollout 1 evaluation

Figure 7: Evaluation comparison: default vs. batch size 256 (rollout = 1).

```
Loading data from results/batch_64/rollout1/model_iter_1-merged_20251117_100606_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 788
Overall Accuracy: 59.74%

Results saved to results/batch_64/rollout1/model_iter_1-merged_20251117_100606_evaluation.jsonl
```

(a) Batch size 64 rollout 1 evaluation

```
Loading data from results/lr_4e-5/rollout1/model_iter_1-merged_20251118_062208_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 792
Overall Accuracy: 60.05%

Results saved to results/lr_4e-5/rollout1/model_iter_1-merged_20251118_062208_evaluation.jsonl
```

(b) Increased lr (4e-5) rollout 1 evaluation

Figure 8: Evaluation comparison: batch size 64 vs. increased lr (4e-5) (rollout = 1).

```
Loading data from results/lr_1e-5/rollout1/model_iter_1-merged_20251118_055902_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 810
Overall Accuracy: 61.41%

Results saved to results/lr_1e-5/rollout1/model_iter_1-merged_20251118_055902_evaluation.jsonl
```

(a) Decreased lr (1e-5) rollout 1 evaluation

```
Basic Evaluation Results:
Total examples: 1319
Correct answers: 796
Overall Accuracy: 60.35%

Results saved to results/queries_3000/rollout1/model_iter_1-merged_20251117_104943_evaluation.jsonl
```

(b) Increased queries (3000) rollout 1 evaluation

Figure 9: Evaluation comparison: decreased lr vs. increased queries (rollout = 1).

```
Loading data from results/queries_1000/rollout1/model_iter_1-merged_20251117_111139_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 795
Overall Accuracy: 60.27%

Results saved to results/queries_1000/rollout1/model_iter_1-merged_20251117_111139_evaluation.jsonl
```

(a) Decreased queries (1000) rollout 1 evaluation

```
Loading data from results/rank_128/rollout1/model_iter_1-merged_20251117_113406_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 786
Overall Accuracy: 59.59%

Results saved to results/rank_128/rollout1/model_iter_1-merged_20251117_113406_evaluation.jsonl
```

(b) Increased rank (128) rollout 1 evaluation

Figure 10: Evaluation comparison: decreased queries vs. increased rank (rollout = 1).

```
Loading data from results/rank_32/rollout1/model_iter_1-merged_20251118_054629_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 812
Overall Accuracy: 61.56%

Results saved to results/rank_32/rollout1/model_iter_1-merged_20251118_054629_evaluation.jsonl
```

(a) Decreased rank (32) rollout 1 evaluation

```
Loading data from results/baseline/rollout1/Qwen3-0.6B_20251118_090623_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 785
Overall Accuracy: 59.51%

Results saved to results/baseline/rollout1/Qwen3-0.6B_20251118_090623_evaluation.jsonl
```

(b) Baseline rollout 1 evaluation

Figure 11: Evaluation comparison: decreased rank vs. baseline (rollout = 1).

Table 3: Hyperparameter variation experiments and model performance.

Model	Iteration	Batch size	lr	No. queries	LoRA rank	Rollout	Accuracy
Baseline	0	-	-	2000	-	8	59.51%
Default para	1	128	2e-5	2000	64	8	59.82%
Increase batch size	1	**256**	2e-5	2000	64	8	60.58%
Decrease batch size	1	**64**	2e-5	2000	64	8	59.74%
Increase lr	1	128	**4e-5**	2000	64	8	60.05%
Decrease lr	1	128	**1e-5**	2000	64	8	61.41%
Increase queries	1	128	2e-5	**3000**	64	8	60.35%
Decrease queries	1	128	2e-5	**1000**	64	8	60.27%
Increase rank	1	128	2e-5	2000	**128**	8	59.59%
Decrease rank	1	128	2e-5	2000	**32**	8	61.56%

Note: The models are trained with rollout=8, but the evaluation in Table 3 uses rollout=1.

- Ablation Study: Overall, the training allows the model to improve slightly, from 59.51% to 59.82%, indicating that fine tuning is a good direction. Besides, by increasing the batch size to 256, it gives out one of the best performance improvement (60.58%) and it can be suggested that larger batch size provide better generalization. In addition, while the assignment mentions that a larger learning rate can be used for LoRA, the lower learning rate actually out perform a larger learning rate (61.41% > 60.05%), indicating the lower learning rate is actually more stable. Changing the query number both slightly enhances the number of queries, but it is believed it is just a variation and it is not guaranteed to have better performance for all evaluation trials. Lastly, it is found that limiting the LoRA rank to 32 contributes the highest accuracy (61.56%) among all models, and increasing rank to 128 only slightly rise the accuracy compared to the baseline, highlighting that smaller rank may provide better generalization and training efficiency for this particular task. Overall, compared to the default training, it shows that a smaller learning rate, a large batch size, a suitable number of queries and a smaller LoRA ranks can produce a stable and good performance.

5.3 Pass@k Analysis

```
Basic Evaluation Results:
Total examples: 10552
Correct answers: 6349
Overall Accuracy: 60.17%

Detected multiple rollouts: 1319 unique questions with 10552 total rollouts

============================================================
Multiple Rollout Metrics:
============================================================
k     avg@k           pass@k
------------------------------------------------------------
1     0.5944          0.5944
2     0.5986          0.7119
3     0.6037          0.7650
4     0.6025          0.7885
5     0.6020          0.8074
6     0.6022          0.8271
7     0.6026          0.8370
8     0.6017          0.8423
============================================================

Metrics saved to results/model_iter_1-merged_20251115_060728_evaluation_metrics.json
```

(a) Default rollout 8 evaluation

```
Basic Evaluation Results:
Total examples: 10552
Correct answers: 6340
Overall Accuracy: 60.08%

Detected multiple rollouts: 1319 unique questions with 10552 total rollouts

============================================================
Multiple Rollout Metrics:
============================================================
k     avg@k           pass@k
------------------------------------------------------------
1     0.6103          0.6103
2     0.6149          0.7127
3     0.6050          0.7566
4     0.6077          0.7892
5     0.6032          0.8059
6     0.6008          0.8188
7     0.6020          0.8324
8     0.6008          0.8400
============================================================

Metrics saved to results/batch_256/rollout8/model_iter_1-merged_20251117_103146_evaluation_metrics.json
```

(b) Batch size 256 rollout 8 evaluation

Figure 12: Evaluation comparison: default vs. batch size 256 (rollout = 8).

```
Basic Evaluation Results:
Total examples: 10552
Correct answers: 6484
Overall Accuracy: 61.45%

Detected multiple rollouts: 1319 unique questions with 10552 total rollouts

============================================================
Multiple Rollout Metrics:
============================================================
k     avg@k           pass@k
------------------------------------------------------------
1     0.5974          0.5974
2     0.6005          0.7058
3     0.6101          0.7680
4     0.6137          0.7991
5     0.6164          0.8249
6     0.6159          0.8400
7     0.6155          0.8484
8     0.6145          0.8552
============================================================

Metrics saved to results/batch_64/rollout8/model_iter_1-merged_20251117_100949_evaluation_metrics.json
```

(a) Batch size 64 rollout 8 evaluation

```
Basic Evaluation Results:
Total examples: 10552
Correct answers: 6443
Overall Accuracy: 61.06%

Detected multiple rollouts: 1319 unique questions with 10552 total rollouts

============================================================
Multiple Rollout Metrics:
============================================================
k     avg@k           pass@k
------------------------------------------------------------
1     0.6133          0.6133
2     0.6080          0.7134
3     0.6083          0.7642
4     0.6077          0.7923
5     0.6117          0.8135
6     0.6128          0.8302
7     0.6110          0.8332
8     0.6106          0.8469
============================================================

Metrics saved to results/lr_4e-5/rollout8/model_iter_1-merged_20251118_062607_evaluation_metrics.json

Results saved to results/lr_4e-5/rollout8/model_iter_1-merged_20251118_062607_evaluation.jsonl
```

(b) Increased lr (4e-5) rollout 8 evaluation

Figure 13: Evaluation comparison: batch size 64 vs. increased lr (4e-5) (rollout = 8).

```
Basic Evaluation Results:
Total examples: 10552
Correct answers: 6303
Overall Accuracy: 59.73%

Detected multiple rollouts: 1319 unique questions with 10552 total rollouts

============================================================
Multiple Rollout Metrics:
============================================================
k     avg@k           pass@k
------------------------------------------------------------
1     0.6126          0.6126
2     0.6096          0.7271
3     0.6035          0.7703
4     0.6018          0.7930
5     0.6023          0.8135
6     0.6010          0.8271
7     0.5993          0.8309
8     0.5973          0.8370
============================================================

Metrics saved to results/lr_1e-5/rollout8/model_iter_1-merged_20251118_060413_evaluation_metrics.json

Results saved to results/lr_1e-5/rollout8/model_iter_1-merged_20251118_060413_evaluation.jsonl
```

(a) Decreased lr (1e-5) rollout 8 evaluation

```
Basic Evaluation Results:
Total examples: 10552
Correct answers: 6451
Overall Accuracy: 61.14%

Detected multiple rollouts: 1319 unique questions with 10552 total rollouts

============================================================
Multiple Rollout Metrics:
============================================================
k     avg@k           pass@k
------------------------------------------------------------
1     0.6103          0.6103
2     0.6073          0.7096
3     0.6123          0.7574
4     0.6132          0.7908
5     0.6099          0.8127
6     0.6109          0.8271
7     0.6096          0.8362
8     0.6114          0.8506
============================================================

Metrics saved to results/queries_3000/rollout8/model_iter_1-merged_20251117_105342_evaluation_metrics.json

Results saved to results/queries_3000/rollout8/model_iter_1-merged_20251117_105342_evaluation.jsonl
```

(b) Increased queries (3000) rollout 8 evaluation

Figure 14: Evaluation comparison: decreased lr vs. increased queries (rollout = 8).

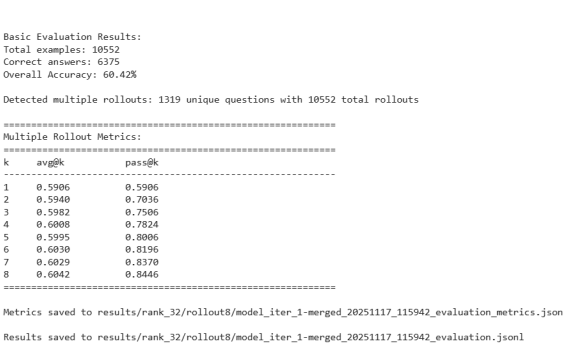
```
Basic Evaluation Results:
Total examples: 10552
Correct answers: 6320
Overall Accuracy: 59.89%

Detected multiple rollouts: 1319 unique questions with 10552 total rollouts

============================================================
Multiple Rollout Metrics:
============================================================
k     avg@k           pass@k
------------------------------------------------------------
1     0.5974          0.5974
2     0.5997          0.7020
3     0.5982          0.7491
4     0.5982          0.7809
5     0.5971          0.8059
6     0.5997          0.8165
7     0.5992          0.8256
8     0.5989          0.8362
============================================================

Metrics saved to results/queries_1000/rollout8/model_iter_1-merged_20251117_111610_evaluation_metrics.json

Results saved to results/queries_1000/rollout8/model_iter_1-merged_20251117_111610_evaluation.jsonl
```

(a) Decreased queries (1000) rollout 8 evaluation

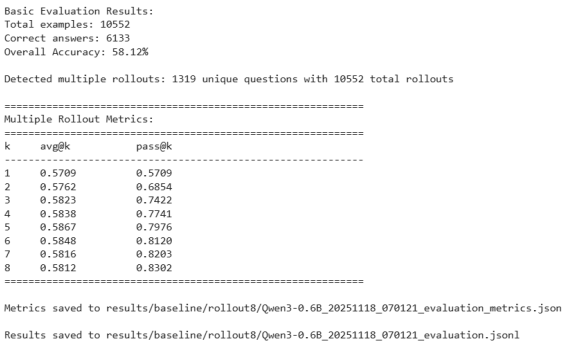
```
Basic Evaluation Results:
Total examples: 10552
Correct answers: 6322
Overall Accuracy: 59.91%

Detected multiple rollouts: 1319 unique questions with 10552 total rollouts

============================================================
Multiple Rollout Metrics:
============================================================
k     avg@k           pass@k
------------------------------------------------------------
1     0.5868          0.5868
2     0.5876          0.6967
3     0.5896          0.7453
4     0.5933          0.7817
5     0.5959          0.8014
6     0.5964          0.8188
7     0.5973          0.8287
8     0.5991          0.8415
============================================================

Metrics saved to results/rank_128/rollout8/model_iter_1-merged_20251117_113759_evaluation_metrics.json

Results saved to results/rank_128/rollout8/model_iter_1-merged_20251117_113759_evaluation.jsonl
```

(b) Increased rank (128) rollout 8 evaluation

Figure 15: Evaluation comparison: decreased queries vs. increased rank (rollout = 8).



(a) Decreased rank (32) rollout 8 evaluation



(b) Baseline rollout 8 evaluation

Figure 16: Evaluation comparison: decreased rank vs baseline (rollout = 8).

Table 4: Hyperparameter variation for pass@k analysis with rollouts=8.

Model	**Iteration**	**avg@1**	**avg@4**	**avg@8**	**pass@1**	**pass@4**	**pass@8**
Baseline	0	0.5709	0.5838	0.5812	0.5709	0.7741	0.8302%
Default para	1	0.5944	0.6025	0.6017	0.5944	0.7885	0.8423 %
Increase batch size	1	0.6103	0.6077	0.6008	0.6103	0.7892	0.8400%
Decrease batch size	1	0.5974	0.6137	0.6145	0.5974	0.7991	0.8552%
Increase lr	1	0.6133	0.6077	0.6106	0.6133	0.7923	0.8469%
Decrease lr	1	0.6126	0.6018	0.5973	0.6126	0.7930	0.8370%
Increase queries	1	0.6103	0.6132	0.6114	0.6103	0.7908	0.8506%
Decrease queries	1	0.5974	0.5982	0.5989	0.5974	0.7809	0.8362 %
Increase rank	1	0.5868	0.5933	0.5991	0.5868	0.7817	0.8415%
Decrease rank	1	0.5906	0.6008	0.6042	0.5906	0.7824	0.8446%

- How do avg@k and pass@k change as k increases: Across most models, both the avg@k and pass@k increase with k, except for the avg@k of "Increase batch size" and "lr". The reason for increasing avg@k is that more opportunities are given to generate correct response, hence the average enhances. In addition, the increasing pass@k indicates that $P(at\ least\ 1\ output\ is\ correct)$ increases with k. In conclusion, pass@k increases more evidently than avg@k.

- Comparison of metric before and after training: By comparing the baseline and all trained models, all avg@k and pass@k show manifest improvement according to the table. Thus, training can indeed improve the average and probability.

- Indication of trend to model diversity and reliability: The larger gap between pass@k and avg@k, the larger variety of the response. When k increases, as pass@k increases more evidently than avg@k, it means that the additional samples contribute to the diversity. However, the reliability only improves slightly over k, indicating that reliability can be improved but diversity will be more evident over k.

- Does the trained model show better diversity than baseline: The trianed model does not show noticeably better diversity than the baseline. While both avg@k and pass@k improve better after training, the gap between 2 evaluations are varying around 0.24 to 0.25. Yet, the improvement of evaluations indicate a better reliability, which is a good thing.

- Indication of the gap between avg@k and pass@k for baseline: As k increases, the gap between avg@k and pass@k increases as avg@k remains at around 58%. Therefore, the response from

baseline models are generated with diversity, which helps the success rate to rise, despite the correctness is not consistent.