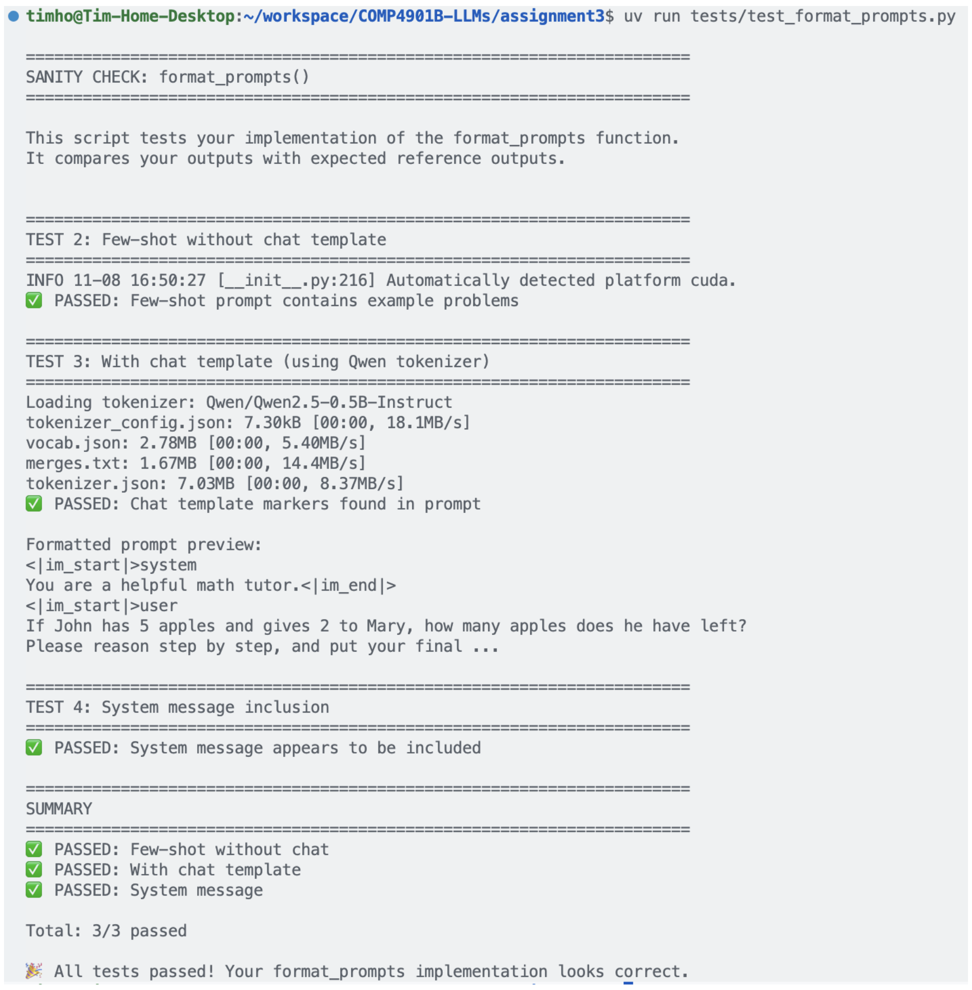
COMP4901B: Large Language Models

Assignment 3 Report

HE, Wenqian
Student ID: 20860896

November 14, 2025

1 Part 1 — VLLM Inference Implementation



```
● timho@Tim-Home-Desktop:~/workspace/COMP4901B-LLMs/assignment3$ uv run tests/test_format_prompts.py

======================================================================
SANITY CHECK: format_prompts()
======================================================================

This script tests your implementation of the format_prompts function.
It compares your outputs with expected reference outputs.


======================================================================
TEST 2: Few-shot without chat template
======================================================================
INFO 11-08 16:50:27 [__init__.py:216] Automatically detected platform cuda.
✅ PASSED: Few-shot prompt contains example problems

======================================================================
TEST 3: With chat template (using Qwen tokenizer)
======================================================================
Loading tokenizer: Qwen/Qwen2.5-0.5B-Instruct
tokenizer_config.json: 7.30kB [00:00, 18.1MB/s]
vocab.json: 2.78MB [00:00, 5.40MB/s]
merges.txt: 1.67MB [00:00, 14.4MB/s]
tokenizer.json: 7.03MB [00:00, 8.37MB/s]
✅ PASSED: Chat template markers found in prompt

Formatted prompt preview:
<|im_start|>system
You are a helpful math tutor.<|im_end|>
<|im_start|>user
If John has 5 apples and gives 2 to Mary, how many apples does he have left?
Please reason step by step, and put your final ...

======================================================================
TEST 4: System message inclusion
======================================================================
✅ PASSED: System message appears to be included

======================================================================
SUMMARY
======================================================================
✅ PASSED: Few-shot without chat
✅ PASSED: With chat template
✅ PASSED: System message

Total: 3/3 passed

🎉 All tests passed! Your format_prompts implementation looks correct.
```

Figure 1: VLLM Inference

1.1 Implementation

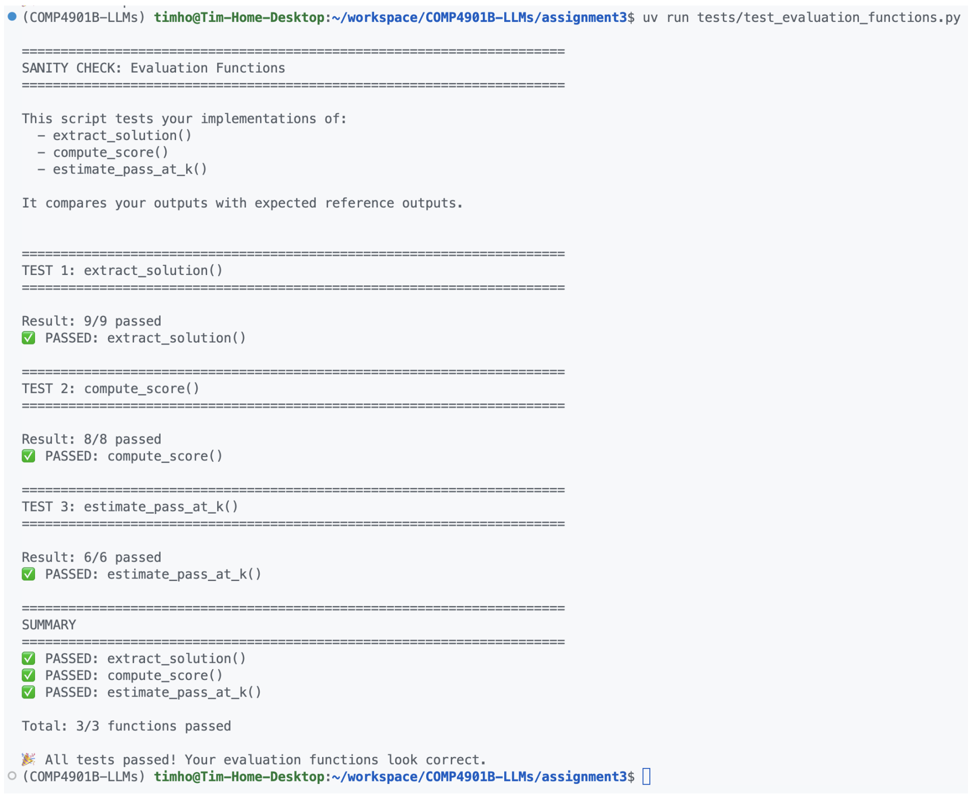
`format_prompts()` I followed the all instructions to implement this function. The quesiton is formatted using specified prompt template by default, then the formatted prompt is applied

chat template from tokenizer with optional system message unless exception occurs, otherwise the raw formatted prompt without chat template is used.
`run_inference()` I followed the all instructions to implement this function. I pass all necessary parameters to LLM engine and sampling parameters. Specially, If the number of rollouts is greater than 1, but temperature is 0.0, the temperature is increased to 0.6 by default to avoid deterministic output. If the number of rollouts is 1, the temperature is set to 0.0 to avoid stochastic output. Otherwise the temperature is set to the specified temperature.

1.2 Answer the question

We need temperature > 0 when generating multiple rollouts because when temperature is 0, the model will generate the same output for each rollout, then there is no meaning to generate multiple rollouts. So we should set temperature > 0 to generate diverse outputs.

2 Part 2 — Answer Verification Implementation



```
● (COMP4901B-LLMs) timho@Tim-Home-Desktop:~/workspace/COMP4901B-LLMs/assignment3$ uv run tests/test_evaluation_functions.py

======================================================================
SANITY CHECK: Evaluation Functions
======================================================================

This script tests your implementations of:
  - extract_solution()
  - compute_score()
  - estimate_pass_at_k()

It compares your outputs with expected reference outputs.


======================================================================
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.
○ (COMP4901B-LLMs) timho@Tim-Home-Desktop:~/workspace/COMP4901B-LLMs/assignment3$ 
```

Figure 2: Answer Verification

2.1 Implementation

```
BOX_PATTERN = r'\\boxed\{([^}]+)\}'
NUMBER_PATTERN = r'(-?(?:\d{1,3}(?:,\d{3})*|\d+)(?:\.\d+)?)'
```

`extract_solution()` I first try to extract the answer in boxed format using the pattern `BOX_PATTERN`, then if not found, I try to extract the answer in number format using the pattern `NUMBER_PATTERN`. The last extracted answer is returned without commas and dollar signs. If both failed, I return None.

`compute_score()` I casted the extracted answer and ground truth to float and back to string to compare the values. This operation is performed in try-except block to handle non-numeric values. If the comparison is successful, it returned the comparison result otherwise 0 is returned.

2.2 Derivation of the pass@k metric

$$\begin{aligned}
\text{pass@k} &= P(\text{at least one correct}) \\
&= 1 - P(\text{all are wrong}) \\
&= 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \\
&= 1 - \frac{(n-c)!}{k!(n-c-k)!} \times \frac{k!(n-k)!}{n!} \\
&= 1 - \frac{(n-c-k+1)(n-c-k+2)\cdots(n-c)}{(n-k+1)(n-k+2)\cdots(n)} \\
&= 1 - \prod_{i=1}^{k} \frac{n-c-k+i}{n-k+i}
\end{aligned}$$

2.3 Answer the question

```
ratios = np.arange(n-c-k+1, n-c+1) / np.arange(n-k+1, n+1)
pass_at_k_values[i] = 1 - np.prod(ratios)
```

The numerical stability is achieved by using the product formulation of the pass@k metric. I created a 1d array of

$$n-c-k+1, n-c-k+2, \cdots, n-c$$

and a 1d array of

$$n-k+1, n-k+2, \cdots, n$$

then divide the corresponding elements to get the 1d array of ratios. Then the pass@k is 1 minus the product of the ratios. This avoids the computation of large factorials and division of large numbers.

3 Part 3 — LoRA Training Implementation

```
def _resolve_lora_target_modules(self) -> List[str]:
    VALID_TARGETS_BY_MODEL = {
        "qwen3": {"q_proj", "k_proj", "v_proj", "o_proj", "gate_proj",
            "up_proj", "down_proj"},
        "llama": {'o_proj', 'k_proj', 'up_proj', 'gate_proj', 'v_proj',
            'q_proj', 'down_proj'},
        "mistral": {'k_proj', 'o_proj', 'down_proj', 'q_proj', 'up_proj
            ', 'gate_proj', 'v_proj'},
        "opt": {'k_proj', 'q_proj', 'fc1', 'v_proj', 'out_proj', 'fc2'}
    }

    model_type: str = self.model.config.model_type

    if self.lora_args.lora_target_modules:
        targets = set(self.lora_args.lora_target_modules)
    elif model_type in VALID_TARGETS_BY_MODEL:
        targets = VALID_TARGETS_BY_MODEL[model_type]
    else:
        raise ValueError(f"No valid target modules found for model type
            : {model_type}")

    available_modules = {name.split(".")[-1] for name, module in self.
       model.named_modules() if isinstance(module, nn.Linear)}

    assert targets <= available_modules, f"Invalid target modules: {
       targets} not in {available_modules}"
    valid_targets = list(targets)

    return valid_targets
```

3.1 Detect architecture and select target modules

I use `self.model.config.model_type` to detect the architecture of the model. I downloaded `Qwen/Qwen3-0.6B`, `unsloth/Llama-3.2-1B`, `unsloth/mistral-7b-bnb-4bit` and `facebook/opt-125m` models from Hugging Face, then printed the model type string, model architecture, and linear module names. Then I picked the name of linear modules in decoder layers for each model. They are the default valid target modules for forementioned models if no target modules are specified. Details can be found in the `scripts/model.ipynb` file.

3.2 Answer the question

In decoder-only transformer models, the attention layers are used to correlate different parts of the input sequence, i.e. attend to appropriate positions. The FFN layers are used to store the knowledge, such as math operation patterns. Finetuning lora on attention and FFN layers can help the model focus on the important numbers and quesiton intention keywords for GSM8K-like math problems, then match the learned patterns in FFN layers to perform math reasoning.

If we only apply lora to attention layers, although it might improve the model's attention on some important number of keywords, but it cannot learn new patterns, such as for which kind of question should it apply division or multiplication.

Moreover, since the model cannot learn new pattern, it might not be able to extract better features in lower layers through FFN, then higher attention layers cannot pay attention to good positions since they are not similar in terms of features, even though they might relevant in eyes of humans.

4 Part 4 — Part 4 — Self-Training Execution

4.1 Training Configuration Summary

```
NUM_ITERATIONS=1
MAX_TOKENS=512
N_ROLLOUTS=8
ENABLE_THINKING=false
N_QUERIES=2000

LEARNING_RATE=2e-5
TOTAL_BATCH_SIZE=128
NUM_EPOCHS=1
SAVE_STEPS=30
LORA_R=64
```

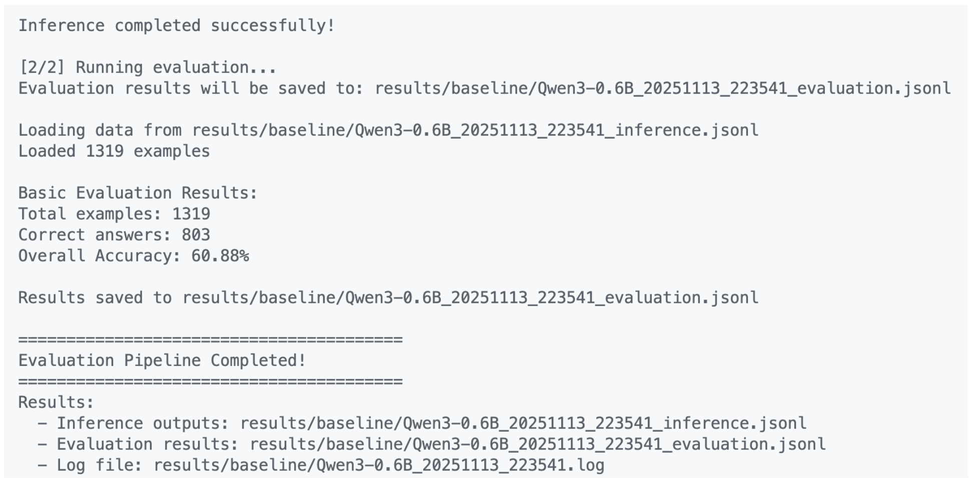
4.2 Metrics

Iteration	#Correct	Pass@1	Pass@4	Pass@8
0	1000	0.57	0.57	0.57

Table 1: Metrics

5 Part 5 — Final Evaluation and Analysis

5.1 Baseline Evaluation



```
Inference completed successfully!

[2/2] Running evaluation...
Evaluation results will be saved to: results/baseline/Qwen3-0.6B_20251113_223541_evaluation.jsonl

Loading data from results/baseline/Qwen3-0.6B_20251113_223541_inference.jsonl
Loaded 1319 examples

Basic Evaluation Results:
Total examples: 1319
Correct answers: 803
Overall Accuracy: 60.88%

Results saved to results/baseline/Qwen3-0.6B_20251113_223541_evaluation.jsonl

========================================
Evaluation Pipeline Completed!
========================================
Results:
  - Inference outputs: results/baseline/Qwen3-0.6B_20251113_223541_inference.jsonl
  - Evaluation results: results/baseline/Qwen3-0.6B_20251113_223541_evaluation.jsonl
  - Log file: results/baseline/Qwen3-0.6B_20251113_223541.log
```

Figure 3: Baseline Evaluation

The baseline accuracy is 60.88% as shown in the figure 3.

5.2 Trained Model Evaluation