CSE234-PA3-PART3

Methods

Speculative decoding is a technique that using a smaller faster model to draft a sequence of tokens. Then, verifying these tokens with the larger model in a single forward pass and accepting correct tokens and regenerating from the first error. Here is some implementation details:

- **initialize_target_model and initialize_draft_model**: These functions load the respective Hugging Face models and tokenizers while ensuring compatibility.
- **generate_draft_tokens**: This function uses `AutoModelForCausalLM.generate()` with controlled parameters (`temperature`, `top_k`, etc.) to generate draft tokens.
- **verify_tokens_vectorized**: This function performs batch verification of the speculative tokens in a single forward pass using the target model.
- **verify_tokens_cached**: This function using `use_cache` parameter to control the KV cache of the target model to accelarate generation.
- **speculative_decode**: Implements the core speculative decoding algorithm by iterating through draft generation and verification cycles while leveraging KV caching for efficiency.

First I the draft model will generate a fixed size, which is 15 in my case, of output with autoregressive way, then using the function `verify_tokens_vectorized` function to verify the tokens. This function will verify the tokens once with attention mask. Then we will only save the tokens which is exactly match. If we are encountered with all rejected or all accepted situations, then the target model will generate 1 next tokens.

Results Analysis

Here is a result:

Prompt	Draft Token Acceptance Rate	Speculative Decoding Time (s)	Speculative Tokens/s	Baseline Decoding Time (s)	Baseline Tokens/s	Speedup	Latency Reduction
Prompt1	56.97%	5.09	21.22	2.64	37.91	0.52x	-93.06%
Prompt2	34.74%	7.74	14.36	2.64	37.83	0.34x	-192.66%
Prompt3	54.87%	4.74	22.42	2.64	37.93	0.56x	-79.71%

We can see that the acceptance rate is not high and the accelarate speed is even slower.

Improvements

Then I use the KV Cache configuration and tune the temperature and generated number of tokens to balance the generated tokens and verified tokens. We optimized speculative decoding by using KV Cache (`use_cache=True`) to Reduced redundant computation in the target model. We also tuning speculative token count (`num_speculative_tokens=12`) to Balanced draft model efficiency and verification accuracy. Besides, we lowered temperature (`temperature=0.34`) to Made the draft model generate more deterministic outputs, increasing acceptance rate.

After I set the following hyper parameters: `num_speculative_tokens = 12`, `use_cache = True`, `temperature=0.34`, `top_k=50`, `top_p=0.85`.

As a result, our speculative decoding improved from **0.56x to 1.11x speedup** and the acceptance rate exceeded **75%** in all cases.

The final results can achieve to acceptance rate higher than 75% and accelarate more than 1x.

Here is the results:

Benchmarking Prompt	Avg Speculative Time (s)	Avg Speculative Tokens/s	Avg Baseline Time (s)	Avg Baseline Tokens/s	Speedup	Latency Reduction	Draft Acceptance Rate
Prompt 1	1.34	79.57	1.48	67.42	1.11x	9.80%	88.89%
Prompt 2	1.43	78.91	1.49	67.14	1.04x	4.04%	94.44%
Prompt 3	1.43	74.69	1.48	67.38	1.04x	3.47%	81.67%

After choosing suitable hyperparameters, the this can balance how many tokens the draft model need to generate and how frequently the target model need to verify. This can achieve better performance on both time and acceptance rate.

One of the challenges we encountered is that the we first do not consider the edge situation that all the draft tokens are rejected or accepted. At this point, the target model need to generate one more next token to continuely leed the generation. Otherwise, it will force the draft model generate again and again.

Bonus

We changed the models to

```
target_model_name = "EleutherAI/pythia-6.9b"
draft_model_name = "EleutherAI/pythia-1.4b"
```

With just bare settings without tuning any parameters. In this way, the model has generally higher acceptance rate and accelate rate.

Then we also explore how to elevate the accpetance rate, we found that, if the smaller model generate less token, then the acceptance rate will increase.

Final Average Draft Token Acceptance Rate is **91.43%**. This is because the variance of the model is smaller.

From this part, we can observe that larger model will gain a higher speedup and smaller `num_speculative_tokens` can gain higher acceptance rate.

`Model Change` in the table means the performance after we change the model, `Token Change` represents we changed the `num_speculative_tokens`. The result is shown below.

Benchmarking Prompt	Average Speedup(Model Change)	Latency Reduction(Model Change)	Draft Token Acceptance Rate(Token Change)
Prompt 1	**1.26x**	**20.70%**	**93.33%**
Prompt 2	**0.45x**	**-120.38%** (Slower than baseline)	**89.37%**
Prompt 3	**1.20x**	**16.44%**	**91.58%**