OPTIMIZING TEXT GENERATION WITH LARGE LANGUAGE MODELS VIA SPECULATIVE SAMPLING

PETROS SIDERIS

Abstract. This paper studies the speculative sampling technique, with the aim of choosing a small language model (SLM) instead of a large one (LLM), when the SLM can perform equally well, activating the LLM only when necessary. Experiments will be conducted to compare different approaches, with response time and energy consumption measurements in relation to the initial large models. Particular emphasis will be given to (a) the selection of appropriate models and (b) their impact on the accuracy and efficiency of the results

1. Introduction

An inference scheduling problem, drawing from compiler optimization theory (e.g., branch prediction, PGO, instruction scheduling) to minimize latency and energy usage. These overlaps stem from the fact that LLM inference can be viewed as a computational pipeline, similar to how compilers treat code as a graph or sequence to transform (See: `Abstract Syntax Tree`). Research shows speedups of 2-3x from speculative decoding alone, and layering compiler-inspired opts could push this further, especially for edge cases like long sequences or low-acceptance-rate drafts

1.1. Speculative decoding/sampling works like this:.

1. The Small LLM generates a bunch of sequential tokens. Then, the big LLM runs all these in one go.
2. For each token, if the probability in the large LLM is higher than the probability of the small, it's taken directly (therefore, it's not messing with the large LLM's statistics). If the probability is lower, the chances of it being taken is proportional to the difference in probabilities. This makes it likely that the token is not taken, and all the effort is wasted. I.e. if the small model is pretty good, we get a speed up, and we don't change the output, but if its bad, we are wasting lots of compute for nothing, and its overall slower. *See* `Branch Predictor`
3. These speculative decoding variants allow tokens to exit the model early if confident, analogous to compiler optimizations like function inclining or dead code elimination to skip unnecessary computations.
4. Speculative sampling often builds a tree of possible token paths during drafting. We could apply compiler-style graph transformations (e.g., pruning redundant branches via static analysis or common sub expression elimination on token probabilities) to make the tree exploration more efficient.
5. In compilers, scheduling reorders operations for parallelism. In speculative sampling, we could "schedule" draft token generation to prioritize high-probability paths, inspired by compiler techniques for out-of-order execution.

1.2. **Keywords.**

1. `Scheduling (computing)`
2. Compiler-inspired runtime design for speculative text generation with multi-model pipelines
3. A prediction-and-scheduling runtime with compiler-like heuristics
4. Code generation for inference graphs
5. SLM–LLM pipeline as an IR (intermediate representation)
6. Adaptive runtime scheduler (like a JIT) for inference.
7. Profiling framework that drives decisions.
8. Performance/energy trade-off curves similar to compiler optimization trade-offs.

1.3. **Resources.**

1. `Accelerating Large Language Model Decoding with Speculative Sampling (DeepMind)`
2. `Accelerating LLM Inference with Staged Speculative Decoding`
3. `Looking Back at Speculative Decoding`
4. `Instantaneous Grammatical Error Correction with Shallow Aggressive Decoding`
5. `A Hitchhiker's Guide to Speculative Decoding - By Team PyTorch at IBM`
6. `Speculative Decoding for 2x Faster Whisper Inference`
7. `Looking back at speculative decoding`
8. `Learning Harmonized Representations for Speculative Sampling`
9. `Llama.cpp speculative sampling: 2x faster inference for large models`
10. `Speculative: PoC for speeding-up inference via speculative sampling by ggerganov`
11. `Speculative Sampling Explained`
12. `llama.cpp: add example for speculative sampling #2030`
13. `llama.cpp: PoC for speeding-up inference via speculative sampling #2926`
14. `ollama: Enable speculative decoding #5800`
15. `Understanding LLM System with 3-layer Abstraction`
16. `Speculative Decoding for 2x Faster Whisper Inference`

Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
Email address: petrside@ee.duth.gr