High Performance LLM serving on Nvidia GPUs – Sharan Chetlur Speaker Q&A

Question: What's the speculative decoding? How feasible or how common do you see it going to be used in the future?
Answer: Speculative decoding is going to be very, very common everywhere, especially as models become larger. The responsiveness and latency are going to matter more and more. The beauty of speculative decoding is that it turns a latency problem into a throughput problem, which is great for GPUs. Traditionally, optimizing for low latency scenarios on GPUs is hard, but with speculative decoding, you can turn that into a throughput problem, which GPUs are great at. So, in the future, it's going to be a really important way to balance lower latency with cost.

Question: Can you just do speculative decoding out of the box or does it have to be very model dependent?
Answer: There is some model dependency on it, and there's a design space around how you come up with your draft tokens. Some ways to do it include using a smaller language model to propose draft tokens, relying on cached history (which needs no additional work), or modifying the model architecture (techniques like Medusa).

Question: you also mentioned this difference between time to first token and time between token. So for you, which is harder to optimize for?
Answer: Usually, it's the time between tokens because the generate phase being memory-bound is much harder to optimize for. The time to first token can be a problem if you have a really, really giant prompt, but usually, crunching through a lot of math is what GPUs are great at. DRAM-bound workloads are much harder to make faster because the memory technology is not keeping up with how fast it is getting faster.

Question: There are multiple modes of serving, like online and batch. How common is each mode? What's the demand for online inference versus batch inference that you have seen?
Answer: Online is far more important, but the whole purpose of things like in-flight batching is you turn an online serving problem into a batching problem with sufficient flexibility in the underlying part. That's how you balance actually getting the efficiency benefits of batching while also maintaining good latency and responsiveness.

Question: So you're saying it's like online inference is way, way more important, but people have like, you are actually trying to batch requests as they come in online so that you can still get higher throughput with like minimizing the latency.
Answer: That's right.

Question: So I think it's a common mode for use is like the streaming mode for each of the token stream by stream. Oh, sorry, token by token. So is there any specialized optimizing technique we have to do to optimize for streaming mode?

Answer: Not really. The only thing that changes in streaming mode is that now instead of waiting for the response to be complete, you also stream out tokens at the end of every iteration. You do this with the implied batching, assessing the state of the world at the end of every iteration, and even if the request is not complete, you stream out whatever token you produced last time. This is usually done asynchronously and makes almost no difference to the job**.**

Question: You also like emphasize the importance of quantizations and it sounds like a free lunch, like everyone should just do it. Like, are there any drawbacks to like post-training quantization?
Answer: You have to do it carefully. Post-training quantization is absolutely model-dependent. The accuracy drop or not you see is very much dependent upon your model and the use case, and also upon your appetite for accuracy loss versus the performance benefits. Simply downcasting and converting won't work; you need to have some kind of technique, even if it's not necessarily a library that somebody else has written, to calculate the scaling factors appropriately and make sure your post-training quantized model still maintains accuracy.

Question: One of your techniques that you mentioned, I'm just blanking on the name, but one of the techniques you mentioned was basically redefining the workload method. What was the name? And you mentioned that this is like standing on the backs of other academics. So like, is there like a good introductory paper that starts to kind of talk about this technique that you can recommend?
Answer: The technique you're referring to is called in-flight batching. A good introductory paper to look at is called "Orca".

Question: Do you have any advice on how to go about choosing an inference service? Because there are like many, many of those out there.
Answer: Depending upon your goals, not everybody needs the absolute last drop of performance, and some people just need things that are more approachable. There are a number of options available, and they're all very good. You kind of have to explore them for yourself, especially if what you're looking for is an on-prem solution. There are some really great open-source projects.

Question: What are some of the questions people should be asking when they evaluating inference services?
Answer: You need to have a very clear understanding of what are your quality of service objectives that you want to hit, both in terms of the accuracy of the responses and the latency or responsiveness of the server. With latency, throughput, and accuracy, there's a Pareto frontier that you have to build for yourself, as you can give up one to gain the other. You have to decide where in this problem space you want to operate first, and then that informs a lot of the decisions that come. If you're working on something that really needs low latency, that's the part of the space you have to operate on. Or, in some cases, it could be that you need the absolute highest accuracy, and you don't want to mess with anything.

Question: What about multi-GPU optimization and single-GPU optimization. Which directions are you more interested in and which will be more common in the future?
Answer: There's no avoiding multi-GPU models; people are just making models bigger and bigger, and however much fancy techniques you come up with to transfer it to one GPU, they're just going to use it to build a bigger model. So, there's no non-multi-GPU; we can't ignore multi-GPU, there's no way to do that. But there are many cases in which single-GPU is just fine. You can quantize things more aggressively in future generation GPUs as they're getting bigger and bigger memory. The same model that previously needed multi-GPU could fit on one. Both single-GPU and multi-GPU are really important. Certainly, for among developers, single-GPU is by far the most important because it's just the easiest one to run.

Question: is inflight-batching coming to the python-api for trt-llm
Answer: We've put a Python wrapper around our c++ runtime, which allows access to IFB.

Question: wont token concatenation cause "context conflicts" . i mean if i have prompt of different context concated to generated token of a different prompt...wont that be a problem?
Answer: the concatenation is only applied to Matmuls and other ops that do not have any sharing across tokens. When you get to the attention part of the graph, you need to break the concat (ie slice at seqeunce boundaries). You get no benefit there. But since the iteration time is domimated by Matmul, it's a net win.

Question: in the context of GPU memory, is the paged KV cache living in GPU shared memory (for a block)? or is it a higher level cache spanning multiple blocks' shared memory (or L1/L2 cache etc)?
Answer: it's in Global memory and even spills into host memory if needed.

Question: With the paged KV cache, what do you do when a certain process's memory is freed? Do you coalesce?
Answer: No... Usually having a decently chunky page size (like 32 or 64 tokens) allows the overheads of non-contiguous KV cache to be small. Remember there is also a hidden dimension, so each page is a few kB.

Question: Is there anything I need to do in order to ensure that TensorRT shares and keeps the KV cache for the system prompt?
Answer: https://github.com/NVIDIA/TensorRT-LLM/blob/main/docs/source/kv_cache_reuse.md

Question: Can you talk about challenges with Training Aware Quantization?
Answer: QAT is just much more expensive since it's training the weights and that often needs more GPUs. PTQ is just based on running inference.

Question: How do you know which parts of memory correspond to the system prompt?
Answer: In TRT-LLM, we auto-detect. For a newly arrived request, we do a maximal prefix matching with existing KV cache blocks and reuse as much as possible.

Question: Does TRT perform kernel optimization like kernel fusion? Is it similar to torch.compile?
Answer: Yes, usually triggers hand-fused kernels.

Question: what's difference is latency and throughput ? is it not inverse of each other?
Answer: For a single request, that is reasonable. When processing things in parallel, the throughput increases by parallelism factor assuming perfect scaling. Of course, it is never perfect