https://deci.ai/blog/llm-inference-optimization-key-challenges-and-solutions/

LLM Inference Optimization: Key Challenges and Solutions

LLM needs context of entire sequence generated prior which is very memory intensive

Prompt sizes vary widely causing issues in any preallocation of blocks of memory

Fused kernels on CUDA improve efficiency by some factor for inference

Python has severe limits on parallelization

Efficient LLM inference on CPUs
Automatic INT4 Quantization of weights and not activations
Optimized CPU tensor library for Lin Alg operations, using KV cache more appropriately
Brought CPU inference about 1.3 - 1.8x faster
Memory usage lowered to about 0.25x

Paper points out places for improvement
CPU is ubiquitous and doesn't look like much work is going on improving libraries for this
Memory usage improvements by manually managing KV cache very impressive

What workloads within generative AI require sequential logic? Can we run those selectively on CPU?

This paper itself is whatever, if interested will have to closely read the lin alg operations they implemented

---

LLMLingua
https://github.com/microsoft/LLMLingua

-> Compressing prompts by removing unnecessary tokens
-> Uses small model to remove non-essential tokens in prompts

Seems like workaround for high inference costs rather than any real change in technique etc.

Assume the model internally anyways is supposed to discard irrelevant information

---

Speculative Streaming: Fast LLM Inference without Auxiliary Models

https://arxiv.org/pdf/2402.11131

Speculative Decoding: Run a small draft model and a large target model.
Draft model makes a set of predictions for tokens that can all be parallely checked by the target model at once.

Speculative streaming: Model predicts n-gram or n tokens at once
and verifies them together - single model

Performance on-par or better than speculative decoding and Medusa
Memory efficient

Really like this idea, models more thought for harder questions as streaming prediction might be wrong

Seems straightforward to scale based on device capacity (i.e. can pick size of predicted stream based on available memory)

LLMCad: Fast and Scalable On-device Large Language Model Inference

LLMs need certain parameter sizes for emergent abilities
-> 1B : meaningful representations
-> 10B: Arithmetic reasoning
-> 30B: Multi task comprehension

Memory wall: Beyond parameter size, we hit a memory wall wherein model weights need to be loaded in/out of memory constantly. Reduces inference latency by >60x

LLMCad: similar to other approaches, runs a smaller and a larger model
In-memory model generates a tree rather than a sequence of tokens to verify
Verification with larger model is triggered when a threshhold of uncertainty is crossed

3 main optimizations of LLMCollaboration:
1. Token tree generation with many pathways
2. Self adjusting fallback strategy
3. Speculatively generates tokens even while verification is taking place

This is probably most interesting of the papers.
The memory bound nature as the limiting factor makes sense
Tree generation also seems smart
Wonder if there is a way to combine this into a single model like speculative streaming
Optimizations are also independent so can implement any of them separately
Generate-then-verify paradigm introduces a sequential dependency or a CPU kind of computation.

MEDUSA: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads

LLMCad is a development of this
Most operations are restricted by memory bandwidth
Introduces 2 versions:
1. Medusa-1: Fine tuned on top of frozen backbone LLM
2. Medusa-2: fine tuned with backbone LLM

LLMcad builds on this so might make more sense to start there
Medusa-1 can be applied directly to a base model so we should look for any existing implementations for Llama 3 to see how it has improved inference latency
Currently only supports single-GPU framework so we will need to extend this for CPU or some mobile processor etc. Will probably be lots of upfront work

---

Fast Inference of Mixture-of-Experts Language Models with Offloading

Mixture of Experts: Run multiple small fine-tuned model and combine their results into single output.
Need a separate model (gating model) that assigns selectively weighs "expert" models weights

Specifically targeted to cheap hardware and interactive assistance
Quantization and Parameter offloading from GPU memory to RAM / SSD with on-demand memory load

Techniques:
LRU caching - keeps an LRU cache of last used experts
Speculative Expert Loading: Guess next set of experts by applying next layer's gating funciton to prev layer's hidden states

Good application of systems optimizations
Want to see how common deployment of MoE models will be and if there are other open source versions beyond Mistral
LRU cache optimization seems the simplest in principle optimization to implement among all optimizations seen today

---

Fast Distributed Inference Serving for Large Language Models

Problem: Orca (current SOTA) uses FCFS and uses iteration level scheduling where a full inference job is the granularity at which new jobs can be scheduled
Goal: Eliminate head of line blocking and minimize job completion time across a set of inference jobs
Issues with any solution: Large memory overhead for maintaining KV cache or any other intermediate representations in GPU memory for scheduling

Methods:
Schedules jobs with a Skip Join Multi Level Feedback Queue

MLFQ - known for scheduling in information agnostic settings
Skip Join MLFQ - Skip join aspects use information we know about LLM inference for better scheduling heuristics
Creates and loads entire KV cache with the first input token into GPU memory
Also explains how to distribute this over multiple GPUs

There is lots of systems related research over the years for scheduling jobs and this seems to be a very good implementation tailored to LLM inference
Expect that as we understand more of LLM inference workload types, many more systems optimizations techniques can be applied here so there is a large theoretical overhang
Already builds on FasterTransformer which is good.
Interesting to explore their Orca implementation and see how that improves on FasterTransformer baseline. Seems like open sourcing a full implementation of just that would be a step to matching existing paid solutions

Not too suitable for us as we lack compute resources for this. Not related to edge computing

---

Power Infer
Problem: Single GPU cannot hold entire memory.
Solution: Only load subset of neurons onto GPU, leaving rest on CPU. Works because few "hot" neurons needed for most inference

Neuron activaltion in LLM follows a skewed power law distribution
GPU and CPU independently process respective set of neurons reducing need for PCIe transfer which removes bottleneck.

PowerInfer split into 2 phases:
Phase 1 (Offline): Profiles LLM for hot and cold neurons. Builds an LLM predictor that aims for a training accuracy of 0.95 for guessing which neurons are needed. Uses ILP to decide which (if any) neurons to place on GPU based on activation frequency
Phase 2 (Online): Runs inference using predicted activated neurons ONLY.
Computes vector-vector multiplications for cold neurons on CPU itself and transfers to GPU for combining results
Only using predicted neurons negligibly impacts accuracy
Minimizes data transmission b/w CPU and GPU recognizing bandwidth there as bottleneck

Results seem very promising, likely greatest one seen so far
Setup is limited to CPU-GPU hybrid. Not sure how it scales with multiple GPUs or some other metrics. Not that useful for small models that may fit fully onto a GPU. Will need some extra hardware to test.

Like the ILP optimizer and the offline optimization phase - feel like reduces non-determinism in the running of the model - probably makes optimizing further easier / simpler