Postgraduate Project

When Does Near-Memory KV Attention Help Local Multi-Agent Inference?

Project description

Large Language Model inference is increasingly moving from cloud servers to local AI systems, where multiple agents may execute concurrently or alternately on a shared device. These workflows exhibit execution patterns such as sequential collaboration, parallel fan-out, tool-use pauses and agent reactivation. As the number of active agents and context length increase, their Key-Value (KV) caches may generate substantial memory traffic during attention computation. Near-memory and processing-in-memory architectures have been proposed to accelerate the memory-intensive attention operations of LLM inference. However, existing studies primarily consider conventional batched requests and do not fully represent the dynamic execution patterns of local multi-agent applications. It therefore remains unclear when near-memory KV attention provides meaningful benefits over conventional GPU- or NPU-based execution. In this project, the student will extend an existing open-source architecture simulator, such as NeuPIMs, to model representative multi-agent execution patterns. The student will characterise how agent concurrency, context length, tool-use pauses, Grouped-Query Attention and memory bandwidth affect KV-cache traffic and system performance. Based on the analysis, the student will design a lightweight policy that dynamically selects whether KV-attention operations should execute on the conventional accelerator or near memory.

The central research question is therefore:

> *Under which architectural and workload conditions does near-memory KV attention provide meaningful benefits over conventional GPU- or NPU-based execution for local multi-agent LLM inference?*

The project will identify the architectural conditions under which near-memory KV attention is beneficial, neutral or detrimental. It will produce a reproducible simulator extension, public or synthetic agent workload traces, a break-even analysis and an evaluation of the proposed selection policy.

Deliverables

- A trace generator for several representative multi-agent execution patterns.
- A characterisation of KV-cache traffic and bottlenecks.
- An extension to an open-source NPU–PIM or memory-system simulator.
- A dynamic NPU/PIM execution-selection mechanism.
- Comparison against accelerator-only, always-PIM, static and oracle policies.
- A final dissertation and reproducible experimental package.

Suggested timeline

- **Months 1–2:** Literature review, familiarisation with the chosen simulator and trace generator for multi-agent execution patterns.
- **Months 2–3:** Simulator extension to model multi-agent execution patterns and KV-cache traffic.
- **Months 3–4:** Characterisation of KV-cache traffic, bottlenecks and break-even conditions.
- **Months 4–5:** Design and implementation of the dynamic NPU/PIM execution-selection policy.
- **Months 5–6:** Comparison against baseline policies, final evaluation and dissertation writing.

Equipment/Software required to carry out this project

- Standard personal computer or laptop: existing equipment, £0 additional cost.
- Python: open source, £0.
- PyTorch: open source, £0.
- Hugging Face Transformers: open source, £0.
- Open-source LLMs, such as Llama-family compatible models, Qwen, Gemma, or TinyLlama models: £0.
- System profiling tools, such as Python profiling libraries, psutil, Linux monitoring utilities, PyTorch Profiler, or equivalent tools: £0.
- Git/GitHub: free educational/open-source use, £0.

Total additional equipment/software cost: £0.

The experiments will be designed so that they can be completed using either a normal PC with appropriately sized models or the available Spark computing resources.

Skills developed in carrying out this project

- Understanding of Large Language Model inference and Transformer architectures.
- Understanding of computer architecture, von-Neumann architecture, GPU and memory systems.
- Performance profiling and benchmarking.
- Experimental methodology and workload characterisation.
- Python and PyTorch programming.
- Use of modern LLM frameworks and open-source models.
- Data analysis and visualisation.
- Scientific writing and presentation of experimental results.

Skills required to carry out this project

- Basic programming experience, preferably in Python.
- Basic understanding of computer systems or computer architecture.
- Basic knowledge of machine learning is desirable but not essential.
- Basic knowledge and experience with LLM Agents, e.g. Claude Code, Codex, OpenClaw, Hermes Agent and so on.
- Ability to work with Linux command-line tools is desirable.
- Previous experience with LLMs, GPU programming, or specialised hardware is desired but not required.