Chenxi Ma

LLM Inference Systems / Machine Learning Systems Engineer

Beijing, China | +86 15387087330 | shannon20051127@gmail.com | github.com/algebra-MCX

Education

Wuhan University of Technology Sep 2024 – Jun 2028

B.Eng. in Mechanical Engineering *Wuhan, China*

Open-Source Engineering

vLLM Project Oct 2026 – Present

Open-Source Contributor – Model Runner V2 / DBO / MoE *GitHub*

For vLLM MRV2, Dual-Batch Overlap (DBO) is particularly well-suited for Mixture-of-Experts (MoE) models, Expert Parallelism (EP), high-concurrency decoding, and distributed inference workloads with substantial communication overhead.

I. MRV2 DBO Execution Correctness and Failure Handling

- **Routed-Expert Indexing Fix (Merged):** #63372 (Merged). Fixed incorrect routed-expert writes during MRV2 DBO microbatch execution by indexing each microbatch using its corresponding token offset. In a 32-request DeepSeek-V2-Lite test, expert-routing agreement improved from 0% on the main branch to 89.4% with DBO enabled.
- **Microbatch Exception Propagation (Merged):** #63476 (Merged). Improved exception handling in `UBatchRunner.finish()` to report the first recorded microbatch failure, preventing the original root cause from being obscured by subsequent exceptions. Added GPU regression tests with 2/2 passing on NVIDIA H200.
- **DBO Worker Failure Recovery (Draft):** #64718 (Draft). Designed an exception-publication and worker-unwinding mechanism to prevent sibling workers from remaining blocked on synchronization primitives after a microbatch failure. The proposed implementation preserves the original exception and prevents indefinite hangs in `UBatchRunner.finish()`. Integration is planned after the MRV1 dependency-removal work in #64668 (Draft).

II. CUDA Graph Execution and Inference Performance

- **MLA Decode Staging (Open):** #63556 (Open). Extended MLA decode staging to support captured FULL microbatch CUDA Graphs under imbalanced data-parallel workloads, targeting more efficient CUDA Graph replay in MRV2 DBO execution. The proposal received maintainer attention and prompted technical discussion of an alternative implementation approach, which remains under evaluation.
- **Performance Evaluation:** In the documented 63/97-load A/B benchmark, the proposed implementation improved output throughput by 15.52% and reduced p95 Time Per Output Token (TPOT) from 132.017 ms to 113.930 ms, a reduction of approximately 13.7%.

III. MoE Communication Correctness – NIXL / DeepEP

- **NIXL Expert Mapping (Open):** #64086 (Open). Fixed invalid expert-ID handling during global-to-physical expert remapping, preserving the `-1` padding sentinel instead of incorrectly mapping invalid entries to valid physical expert indices. Received positive technical feedback from a reviewer; awaiting the next ready-label and CI stage.
- **DeepEP Low-Latency Expert Mapping (Open):** #64340 (Open). Fixed sentinel propagation in round-robin expert remapping across DeepEP Low-Latency dispatch and combine paths. Consolidated the relevant mapping logic and regression coverage. Received positive technical feedback from a reviewer; awaiting the next ready-label and CI stage.
- **Correctness Validation:** Developed CPU adapter regression tests covering mixed valid/invalid expert IDs, all-padding inputs, and linear expert placement. All 6 regression cases passed across the NIXL and DeepEP LL backends.

IV. MRV2 DBO Architecture Refactoring and MRV1 Decoupling

- **Core Execution-Path Decoupling (Draft):** #64668 (Draft). Refactored MRV2 DBO microbatch slicing to eliminate its dependency on MRV1's `maybe_create_ubatch_slices`. Implemented direct padded-slice computation within MRV2's `create_ubatch_slices`, preserving microbatch boundaries and padding semantics. Removed redundant cross-runner slicing operations and simplified the data flow, supporting the eventual removal of legacy MRV1 implementations.
- **Test Decoupling and Architecture Improvements:** Removed MRV2 test dependencies on MRV1's `split_attn_metadata`

and pinned explicit expected slicing results for five batch layouts. Moved `UBatchWrapper` padding-mask tests into a separate test module, reducing cross-runner coupling and improving regression-test independence. Established explicit test oracles to reduce behavioral regression risks during future DBO refactoring and support independent MRV2 evolution.

Industry Experience

Meshy AI Apr 2026 – Aug 2026

Inference Systems Engineering Intern *Remote*

- **Compilation and Kernel Optimization:** Integrated regional `torch.compile` into the two-stage Venus image-to-3D flow-model pipeline, enabling `dynamic=True` for variable vertex counts and leveraging Triton kernel fusion to reduce elementwise overhead. Improved Stage 1 denoising GPU utilization from 37% to 65%.
- **CFG and Denoising Optimization:** Combined conditional and unconditional classifier-free guidance (CFG) passes into a single batched forward (Batch=2), reducing kernel launches by approximately 50%. Precomputed and reused loop-invariant image/voxel context, RoPE caches, and attention metadata to eliminate redundant computation, achieving GPU utilization of up to 95%.
- **Mesh Decoder Optimization:** Batched Sinkhorn operations by vertex degree, consolidated GPU-to-CPU transfers to eliminate per-vertex synchronization, and parallelized Linear Assignment Problem (LAP) execution using a CPU thread pool. Reduced mesh-decoder latency for 6,250-vertex workloads from approximately 20s to under 1s.
- **End-to-End Performance:** Preserved output equivalence while reducing inference latency from 8.7s to 5.2s (approximately 40% reduction) for 600-vertex workloads, and from 26.8s to 10.4s (approximately 2.6× speedup) for 6,250-vertex workloads.

Research Experience

University of Washington – Systems Lab May 2026 – Present

Research Intern *Seattle, WA / Remote*

Advised by: Baris Kasikci and Arvind Krishnamurthy

- Third author of *TraceLab: Characterizing Coding Agent Workloads for LLM Serving* (arXiv:2606.30560), investigating coding-agent execution patterns and their implications for LLM serving systems.
- Contributed to the analysis of approximately 4,300 coding-agent sessions, 350,000 LLM inference steps, and 430,000 tool calls, characterizing request distributions, context lengths, and tool-interaction behavior.
- Analyzed long-running autonomous execution loops, long-input/short-output patterns, heavy-tailed tool-call distributions, and prefix-cache hit rates, providing empirical insights for performance modeling and optimization of agent-oriented LLM serving systems.

Tsinghua University – JittorInfer Nov 2025 – Mar 2026

Research Intern *Beijing, China / Remote*

- **Transformer Inference Graphs:** Implemented Qwen2.5 inference computation graphs in C++ / GGML, covering RMSNorm, QKV projections, Grouped-Query Attention (GQA), RoPE, KV-cache attention, SwiGLU feed-forward networks, and LM-head projection. Integrated model architecture dispatch and graph construction into JittorInfer.
- **Ascend NPU and Graph Engine:** Developed standard and GE-specific execution graphs for Qwen2.5, extending attention indices, query/KV sequence-length metadata, and attention-mask handling. Integrated GGML KV-cache management with GE attention execution paths for Ascend NPU.
- **Inference Framework Integration:** Extended Qwen2.5 model metadata parsing, tokenizer and GGUF loading support, and CANN Flash Attention mask handling. Added model configurations, CMake integration, and Ascend NPU inference launch scripts. Submitted the implementation as JittorInfer PR #7.

Shanghai Jiao Tong University Aug 2025 – Nov 2025

Research Intern *Shanghai, China / Remote*

- **Request Scheduling and Continuous Batching:** Studied continuous batching, prefill/decode scheduling, and request preemption in LLM serving systems. Analyzed GPU utilization, Time to First Token (TTFT), Time Per Output Token (TPOT), and throughput under varying concurrency levels.
- **KV-Cache Memory Management:** Investigated paged KV caches, block allocation, prefix caching, and cache eviction strategies. Designed experiments across varying context lengths and request-arrival distributions to study memory-management behavior.
- **Performance Modeling and Benchmarking:** Developed inference benchmarks across batch sizes, input/output sequence lengths, and concurrency levels. Analyzed relationships among scheduling overhead, KV-cache capacity, and inference tail latency. This experience established the foundation for my subsequent work in LLM inference systems.

Technical Skills

LLM Inference Systems: vLLM, LLM Serving, MRV2, Dual-Batch Overlap, Microbatch Execution, CUDA Graphs, Prefill / Decode

Distributed Inference and MoE: Data Parallelism (DP), Expert Parallelism (EP), NIXL, DeepEP, Expert Routing, Dispatch / Combine

Performance and Correctness: GPU Profiling, Benchmarking, Throughput / TPOT, CUDA Graph Replay, Regression Testing, Failure Propagation

Programming and Engineering: Python, C++, PyTorch, Triton, torch.compile, GGML, pytest, Git / GitHub, Linux, NVIDIA H200 / B200, Ascend NPU