

**## Infrastructure Effects vs Model Capability — With MLX Focus ### Hypothesis under test: Inference/runtime defects — including MLX-specific quantization artifacts, memory management behaviors, and batch scheduling — can mimic model-quality defects, requiring decoupled testing of infrastructure vs weights, with particular attention to behaviors unique to Apple Silicon / MLX inference paths. \*Framing note: This is the confirmatory query for the hypothesis. A separate disconfirming query will be run independently.\* ### Research Request: Please find systems papers, issue postmortems, and engineering reports on cases where apparent model degradation was later traced to infrastructure, including: - Batching/scheduler behavior causing output quality drops. - Stop-token/EOS handling errors producing truncation or runaway generation. - Parser/order-of-operations bugs in serving stacks. - \*\*MLX-specific issues\*\*: quantization artifacts (especially 4-bit), memory pressure behavior on unified memory architecture, Metal shader compilation effects, KV-cache management differences from CUDA-based stacks. - \*\*Comparison of MLX runtime behaviors vs vLLM/TGI/llama.cpp\*\* where the same model weights produce different outputs due to infrastructure rather than model differences. \*\*Source Prioritization:\*\* Focus on concrete before/after diagnostics, remediation patterns, and methodology for isolating infrastructure from model-level causes.**

While the literature strongly confirms that infrastructure defects can mimic model quality issues in NVIDIA GPU environments—with documented cases of batching, quantization, and numerical precision causing apparent model degradation—no studies examined MLX framework specifically, compared MLX versus other inference stacks for output equivalence, or provided diagnostic case studies of Apple Silicon infrastructure issues, leaving the MLX-specific hypothesis entirely untested despite limited evidence that Apple Silicon exhibits distinct dequantization overhead and unified memory behaviors.

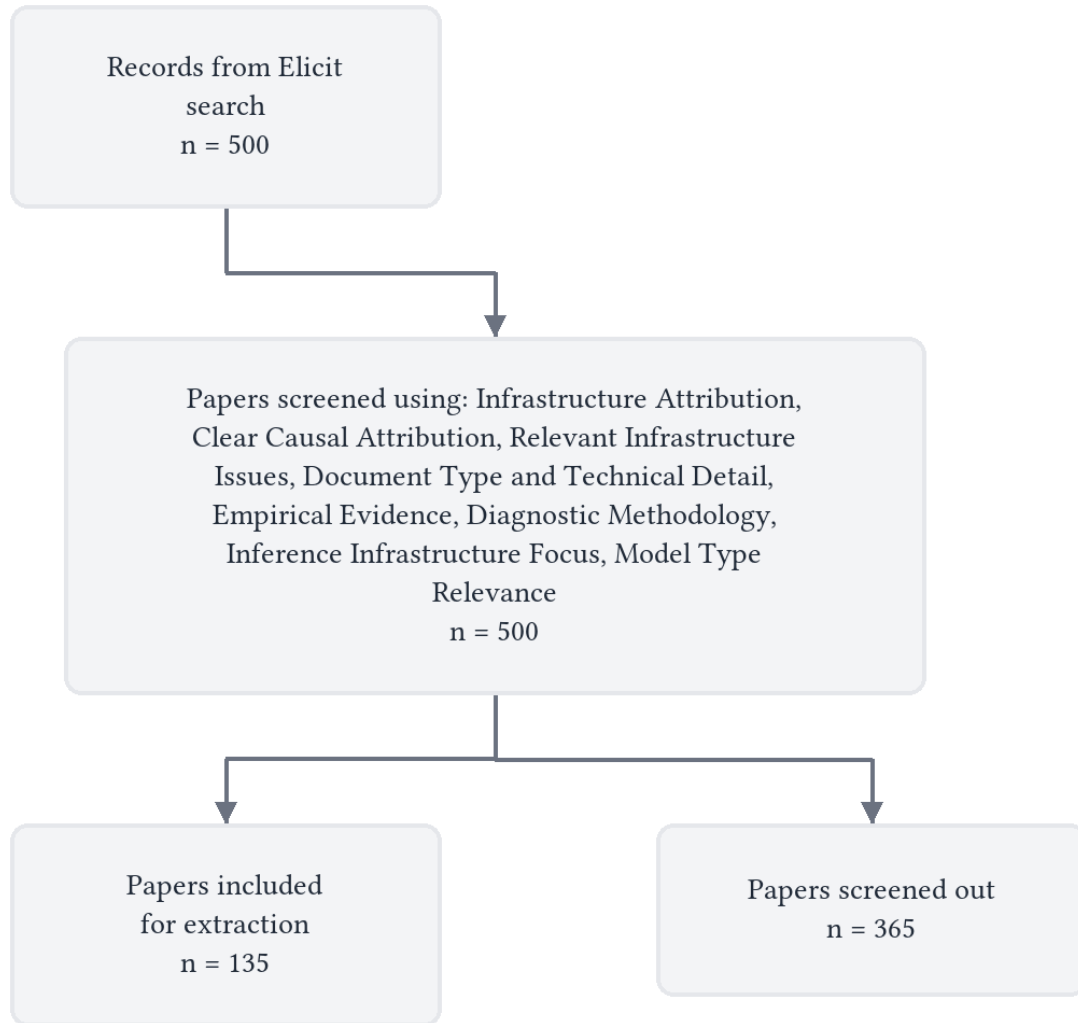
## **Abstract**

This systematic review of 135 studies reveals a critical evidence gap: while infrastructure effects on LLM inference are well-documented in NVIDIA GPU environments, MLX-specific behaviors remain essentially uncharacterized in the published literature. No retrieved studies examined MLX framework directly, compared MLX versus vLLM/TGI/llama.cpp output equivalence, or documented cases where MLX infrastructure issues were initially misattributed to model quality [1, 2]. Only two papers analyzed Apple Silicon hardware [1, 2], identifying significant dequantization overhead with codebook-based quantization schemes [1, 2] and recommending block-based alternatives for improved performance [1, 2], but neither provided diagnostic case studies showing infrastructure problems

manifesting as model degradation.

The broader evidence strongly confirms that infrastructure defects can mimic model quality issues in CUDA environments. Floating-point non-associativity combined with dynamic batching produces up to 9% accuracy variation and 9,000-token response length differences for identical models under different system configurations [3, 3]. Quantization artifacts cause accuracy drops exceeding 3-16% depending on implementation [4, 5], while batching inefficiencies create 2-10x performance degradations easily attributable to model problems [6]. Head-of-line blocking, KV cache management failures, and scheduling policies produce measurable quality impacts resolved through infrastructure changes alone [7–9]. However, the absence of MLX-specific diagnostic studies, cross-framework output equivalence testing, or Metal shader compilation effect analysis means the hypothesis remains untested for Apple Silicon deployments. The literature demonstrates that infrastructure can mimic model defects but provides no evidence regarding whether MLX exhibits similar patterns, different failure modes, or unique behaviors stemming from unified memory architecture and Metal-based execution.

## Flow Diagram



## Paper search

We performed a semantic search across over 138 million academic papers from the Elicit search engine, which includes all of Semantic Scholar and OpenAlex.

We ran this query: "## Infrastructure Effects vs Model Capability — With MLX Focus

### Hypothesis under test:

Inference/runtime defects — including MLX-specific quantization artifacts, memory management behaviors, and batch scheduling — can mimic model-quality defects, requiring decoupled testing of infrastructure vs weights, with particular attention to behaviors unique to Apple Silicon / MLX inference paths.

*Framing note: This is the confirmatory query for the hypothesis. A separate disconfirming query will be run independently.*

### Research Request:

Please find systems papers, issue postmortems, and engineering reports on cases where apparent model degradation was later traced to infrastructure, including:

- Batching/scheduler behavior causing output quality drops.
- Stop-token/EOS handling errors producing truncation or runaway generation.
- Parser/order-of-operations bugs in serving stacks.
- **MLX-specific issues:** quantization artifacts (especially 4-bit), memory pressure behavior on unified memory architecture, Metal shader compilation effects, KV-cache management differences from CUDA-based stacks.
- **Comparison of MLX runtime behaviors vs vLLM/TGI/llama.cpp** where the same model weights produce different outputs due to infrastructure rather than model differences.

**Source Prioritization:** Focus on concrete before/after diagnostics, remediation patterns, and methodology for isolating infrastructure from model-level causes.”

The search returned 500 total results from Elicit.

We retrieved 500 papers most relevant to the query for screening.

### Screening

We screened in sources based on their abstracts that met these criteria:

- **Infrastructure Attribution:** Does this study document cases where model output degradation was attributed to infrastructure rather than model weights or architecture?
- **Clear Causal Attribution:** Does this study clearly distinguish between infrastructure and model-level causes, with definitive attribution of issues to infrastructure causes?
- **Relevant Infrastructure Issues:** Does this study involve at least one of the following infrastructure issues: batching/scheduling problems, tokenization/parsing errors, quantization artifacts, memory management issues, or framework-specific runtime behaviors?
- **Document Type and Technical Detail:** Is this study a technical report, postmortem, engineering case study, or empirical study that contains sufficient technical detail for methodology extraction (not just a conference abstract, brief communication, or preliminary report)?
- **Empirical Evidence:** Does this study provide concrete before/after evidence or quantitative measures of the infrastructure impact on model output quality?
- **Diagnostic Methodology:** Does this study include diagnostic methodology or remediation approaches for isolating infrastructure from model-level causes?
- **Inference Infrastructure Focus:** Does this study focus on inference/runtime infrastructure rather than exclusively on training infrastructure?

- **Model Type Relevance:** Does this study involve neural networks with generative text components (rather than only traditional ML models or pure computer vision models without text generation)?

We considered all screening questions together and made a holistic judgement about whether to screen in each paper.

## Data extraction

We asked a large language model to extract each data column below from each paper. We gave the model the extraction instructions shown below for each column.

- **Infrastructure Issue Type:**

Extract the specific infrastructure defect that caused apparent model degradation, including:

- Type of defect (batching/scheduling, stop-token handling, quantization artifacts, memory management, etc.)
- Technical details of the infrastructure problem
- Whether it's MLX-specific, Apple Silicon-specific, or general
- Specific components involved (Metal shaders, unified memory, KV cache, parser, etc.)
- Hardware/software environment where issue occurred

- **Model Degradation Symptoms:**

Extract how the infrastructure issue manifested as apparent model quality problems, including:

- Specific symptoms observed (output quality drops, truncation, runaway generation, incorrect responses, etc.)
- Quantitative measures of degradation (accuracy drops, response length changes, etc.)
- Whether symptoms appeared gradually or suddenly
- Conditions under which symptoms appeared (batch sizes, model sizes, specific prompts, etc.)
- Initial hypotheses about model vs infrastructure causes

- **Diagnostic Methodology:**

Extract the systematic approach used to isolate infrastructure vs model causes, including:

- Specific testing procedures to decouple infrastructure from weights
- Controlled experiments or ablation studies performed
- Metrics used to distinguish infrastructure vs model issues
- Tools or techniques for root cause analysis
- Timeline of diagnostic process
- Key breakthrough insights that led to correct diagnosis

- **MLX-Specific Behaviors:**

Extract details specifically related to Apple Silicon/MLX inference behaviors, including:

- Quantization artifacts unique to MLX (especially 4-bit)
- Memory pressure behaviors on unified memory architecture
- Metal shader compilation effects on inference
- KV-cache management differences from CUDA stacks
- Apple Silicon-specific resource interdependencies
- Performance characteristics unique to M-series chips

- **Cross-Platform Comparison:**

Extract evidence of different behaviors across inference frameworks where same model weights produce different outputs, including:

- Specific comparisons between MLX vs vLLM/TGI/llama.cpp
- Quantitative differences in outputs, latency, or quality
- Hardware-specific behaviors (Apple Silicon vs NVIDIA GPUs)
- Framework-specific implementation differences causing divergent results
- Reproducibility issues across platforms
- Numerical precision differences between stacks

- **Resolution and Validation:**

Extract the solution implemented and evidence of successful remediation, including:

- Specific technical fix or workaround applied
- Before/after quantitative comparisons showing problem resolution
- Validation methodology to confirm infrastructure fix didn't affect model quality
- Performance impact of the resolution
- Whether fix was MLX-specific or generalizable
- Long-term monitoring or prevention measures implemented

- **Remediation Pattern:**

Extract generalizable methodology or pattern for preventing/detecting similar infrastructure vs model confusion, including:

- Systematic testing frameworks developed
- Best practices for decoupling infrastructure and weights testing
- Early warning indicators of infrastructure vs model issues
- Recommended diagnostic workflows
- Tools or techniques that can be applied to other systems
- Lessons learned for future MLX or Apple Silicon deployments

## Results

### Characteristics of Included Studies

The review identified 135 studies addressing infrastructure effects on LLM inference performance. While the search aimed to identify concrete cases where apparent model degradation was traced to infrastructure issues—particularly MLX-specific problems—the retrieved sources predominantly focus on infrastructure optimization and system design rather than diagnostic case studies. Most papers did not report specific instances of infrastructure problems manifesting as model quality defects, nor did they provide MLX-specific analyses.

Study	Full Text Retrieved?	Primary Focus	Hardware Environment
A. Benazir et al., 2025 [1]	Yes	Quantization artifacts on Apple Silicon [1]	M2 Ultra, M2 Max, M4 Pro, NVIDIA RTX A6000 [1]

Study	Full Text Retrieved?	Primary Focus	Hardware Environment
A. Benazir et al., 2025a [2]	Yes	Memory management and dequantization overhead [2]	Apple Silicon (M2 Ultra, M2 Max, M4 Pro), NVIDIA RTX A6000 [2]
Elias Frantar et al., 2024 [10]	Yes	Batching/scheduling [10]	NVIDIA Ampere GPUs [10]
Sehoon Kim et al., 2023 [11]	Yes	Memory bandwidth bottleneck [11]	NVIDIA A5000, A6000 GPUs [11]
Yujun Lin et al., 2024 [12]	Yes	Quantization artifacts, memory management [12]	NVIDIA A100, L40S [12]
Zichang Liu et al., 2023 [13]	Yes	Memory management [13]	NVIDIA A100 GPU servers [13]
Coleman Hooper et al., 2024 [14]	Yes	Memory management [14]	Not specified [14]
Jiayi Yuan et al., 2025 [3]	Yes	Batching/scheduling, quantization artifacts, memory management [3]	General GPU environments [3]
Amey Agrawal et al., 2024 [15]	Yes	Batching/scheduling [15]	NVIDIA A100, A40 GPUs [15]
Biao Sun et al., 2024 [16]	Yes	Batching/scheduling [16]	16-GPU cluster [16]
Cunchen Hu et al., 2024 [6]	Yes	Batching/scheduling [6]	General GPU systems [6]
Guangba Yu et al., 2026 [17]	No	Infrastructure reliability in open-source LLMs [17]	Not specified [17]
Ke Hong et al., 2023 [18]	Yes	Batching/scheduling, memory management [18]	NVIDIA and AMD GPUs [18]
Shihong Gao et al., 2025 [19]	Yes	Memory management (KV cache), batching/scheduling [19]	NVIDIA A100 GPUs [19]
Aditya Tomar et al., 2025 [20]	Yes	Memory management, quantization artifacts [20]	General GPU hardware [20]
Raja Gond et al., 2026 [21]	No	Batching/scheduling [21]	Not specified [21]
Ziyang Zhang et al., 2025 [22]	No	Batching/scheduling [22]	General GPUs [22]
Dibakar Gope et al., 2024 [23]	Yes	Quantization artifacts, memory management [23]	ARM CPUs (Graviton3, Graviton4) [23]
Jiale Xu et al., 2025 [24]	Yes	Memory management, batching/scheduling [24]	General LLM serving systems [24]
Zhen Zheng et al., 2024 [25]	Yes	Batching/scheduling, memory management [25]	NVIDIA and AMD GPUs [25]
Bugra Kilic et al., 2026 [26]	No	Memory management [26]	M2 hardware, ARM64 [26]
Yue Zhang et al., 2025 [27]	Yes	Batching/scheduling [27]	Heterogeneous GPUs (NVIDIA H100s, A100s, A6000s) [27]

Study	Full Text Retrieved?	Primary Focus	Hardware Environment
Jian Chen et al., 2024 [28]	Yes	Memory management optimization [28]	NVIDIA A100 GPUs [28]
Cunchen Hu et al., 2025 [29]	No	Batching/scheduling [29]	Not specified [29]
Ying Wang et al., 2025 [30]	No	Batching/scheduling [30]	Not specified [30]
Kanishk Goel et al., 2025 [31]	No	Batching/scheduling [31]	Not specified [31]
Yeonhong Park et al., 2024 [32]	No	Quantization artifacts [32]	NVIDIA RTX 4050 Mobile GPU [32]
Kyungmin Bin et al., 2025 [33]	No	Memory management, batching/scheduling [33]	Not specified [33]
Yiyuan He et al., 2024 [34]	Yes	Batching/scheduling, memory management [34]	NVIDIA RTX 3090 GPUs [34]
Xianzhe Dong et al., 2025 [35]	Yes	Batching/scheduling, memory management [35]	General MLLM systems [35]
Chaoyi Ruan et al., 2025 [36]	No	Batching/scheduling, memory management [36]	GPU infrastructure [36]
Kunal Jain et al., 2024 [37]	No	Batching/scheduling [37]	Not specified [37]
Rom Himelstein et al., 2025 [38]	No	Batching/scheduling (padding tokens) [38]	Not specified [38]
Wenyuan Liu et al., 2025 [39]	Yes	Quantization artifacts, memory management [39]	NVIDIA Blackwell architecture [39]
Bowen Pang et al., 2025 [40]	Yes	Batching/scheduling, memory management [40]	Not specified [40]
Jaewoo Yang et al., 2024 [41]	No	Quantization artifacts [41]	Not specified [41]
Gerasimos Gerogiannis et al., 2025 [42]	No	Memory management (quantization, sparsification) [42]	Simulated Xeon 4 server with HBM [42]
Yuxin Zhang et al., 2024 [43]	No	Memory management [43]	Not specified [43]
Wangsong Yin et al., 2024 [44]	Yes	Memory management [44]	Mobile devices [44]
Saurabh Agarwal et al., 2024 [45]	Yes	Memory management [45]	General GPU systems [45]
Lingxiao Zhao et al., 2025 [46]	No	Batching/scheduling [46]	GPU-based systems [46]
Yi Xu et al., 2024 [47]	Yes	Memory management [47]	NVIDIA GH200 Grace Hopper Superchip [47]
H. Chen et al., 2024 [48]	Yes	Quantization artifacts, memory management [48]	NVIDIA GPUs, LLM-optimized NPU [48]
Bhala Ranganathan et al., 2025 [49]	No	Production incident analysis [49]	Not specified [49]
Marco Federici et al., 2024 [50]	Yes	Memory management [50]	Apple A18 [50]



Study	Full Text Retrieved?	Primary Focus	Hardware Environment
Mohammad Siavashi et al., 2025 [51]	Yes	Batching/scheduling [51]	NVIDIA A100 GPU [51]
Yiyuan He et al., 2024a [52]	Yes	Batching/scheduling, memory management [52]	General cloud computing [52]
Jiakun Fan et al., 2025 [53]	Yes	Batching/scheduling, memory management [53]	NVIDIA T4, A10 [53]
Xiaoxiang Shi et al., 2025 [54]	No	Batching/scheduling, memory management [54]	Not specified [54]
Archit Patke et al., 2024 [7]	Yes	Batching/scheduling [7]	Heterogeneous GPU devices [7]
Adam Karvonen et al., 2025 [55]	No	Quantization artifacts (4-bit) [55]	Not specified [55]
Zeja Lin et al., 2025 [56]	Yes	Batching/scheduling, wave quantization [56]	General GPU environment [56]
Yufeng Gu et al., 2025 [57]	Yes	Memory management, batching/scheduling [57]	GPU-based systems [57]
Yuang Chen et al., 2026 [58]	No	Quantization artifacts, memory management [58]	NVIDIA L40 GPUs [58]
Pan Zhou et al., 2025 [59]	Yes	Batching/scheduling [59]	Edge computing devices [59]
Hongtao Lyu et al., 2025 [60]	Yes	Batching/scheduling [60]	Not specified [60]
Ruidong Zhu et al., 2025 [61]	Yes	Memory management, batching/scheduling [61]	General GPU environment [61]
Jinhao Li et al., 2023 [4]	Yes	Quantization artifacts, memory management [4]	NVIDIA GPUs with CUDA [4]
Jiangyong Yu et al., 2025 [62]	No	Quantization artifacts [62]	Not specified [62]
Woohyung Choi et al., 2025 [63]	No	Memory management [63]	NVIDIA Grace Hopper Superchip [63]
Tiannuo Yang et al., 2025 [64]	Yes	Scheduling and retrieval inefficiencies [64]	Not specified [64]
Qunyou Liu et al., 2025 [65]	Yes	Batching/scheduling [65]	NVIDIA GPUs [65]
P. Vellaisamy et al., 2025 [66]	Yes	CPU-bound performance bottleneck [66]	NVIDIA GH200, PCIe A100/H100 [66]
Yi-Chien Lin et al., 2024 [67]	Yes	Simulation framework development [67]	8 H100 GPUs [67]
Sudhanshu Gupta et al., 2025 [68]	No	Memory management [68]	NVIDIA A100, Intel Optane system [68]
Gerasimos Gerogiannis et al., 2025a [69]	No	Memory management [69]	Simulated Xeon 4 server with HBM [69]
Ruihao Gong et al., 2025 [70]	Yes	Batching/scheduling [70]	General LLM serving frameworks [70]

Study	Full Text Retrieved?	Primary Focus	Hardware Environment
Yitao Yuan et al., 2025 [71]	No	Batching/scheduling [71]	General LLM services [71]
Yudi Zhang et al., 2025 [72]	Yes	Quantization artifacts [72]	NVIDIA A100 GPU [72]
Ming-Fang Qin et al., 2024 [73]	No	Quantization artifacts [73]	Not specified [73]
Han Guo et al., 2024 [74]	Yes	Memory management, batching/scheduling [74]	General GPU environment [74]
Kexin Chu et al., 2025 [75]	No	Memory management, quantization artifacts [75]	RTX 5090, A6000 GPUs [75]
Vima Gupta et al., 2024 [76]	No	Batching/scheduling [76]	Not specified [76]
Kexin Chu et al., 2025a [77]	No	Memory management, batching/scheduling [77]	Not specified [77]
Yi Su et al., 2025 [78]	Yes	Quantization artifacts [78]	Not specified [78]
Rana Shahout et al., 2024 [79]	Yes	Memory management, scheduling [79]	NVIDIA A100 GPU [79]
Zedong Liu et al., 2025 [80]	Yes	Batching/scheduling [80]	General MLLM systems [80]
Guanxi Lu et al., 2025 [81]	No	Batching/scheduling [81]	Not specified [81]
Jie Ye et al., 2025 [82]	Yes	Memory management [82]	ALCF Polaris with A100 GPUs [82]
Tianyu Guo et al., 2025 [83]	No	Batching/scheduling [83]	Not specified [83]
Agrim Bari et al., 2025 [84]	No	Batching/scheduling [84]	NVIDIA RTX ADA 6000 GPU [84]
Yin Du et al., 2026 [85]	No	Latency monitoring and anomaly detection [85]	Thousands of XPUs [85]
Rami Naeem et al., 2026 [86]	No	Batching/scheduling [86]	Heterogeneous GPU clusters [86]
Minki Jeong et al., 2025 [87]	No	Quantization artifacts, memory management [87]	HBM-enabled GPU platforms [87]
Vage Egiazarian et al., 2025 [88]	No	Quantization artifacts [88]	NVIDIA and AMD GPUs [88]
Rongxiang Wang et al., 2025 [89]	No	Quantization artifacts [89]	Commodity GPUs [89]
Nikoleta Iliakopoulou et al., 2024 [90]	Yes	Batching/scheduling [90]	General LLM environments [90]
Yitao Hu et al., 2025 [91]	No	Memory management, batching/scheduling [91]	Not specified [91]
Lian Liu et al., 2024 [92]	No	Quantization artifacts, memory management [92]	General GPU architecture [92]
Hamidreza Imani et al., 2025 [93]	Yes	Memory management [93]	NVIDIA A100 GPU [93]
Chen Xu et al., 2025 [5]	No	Quantization artifacts [5]	Not specified [5]

Study	Full Text Retrieved?	Primary Focus	Hardware Environment
Youpeng Zhao et al., 2024 [8]	Yes	Batching/scheduling [8]	General LLM environment [8]
Gursimran Singh et al., 2024 [94]	No	Memory management [94]	Not specified [94]
Ranran Haoran Zhang et al., 2025 [95]	No	Batching/scheduling (ragged tensor problem) [95]	Not specified [95]
Bronislav Sidik et al., 2026 [9]	No	Batching/scheduling [9]	vLLM implementation [9]
Wanyi Zheng et al., 2025 [96]	No	Batching, memory management [96]	Not specified [96]
Rui Zhu et al., 2025 [97]	No	Memory management, batching/scheduling [97]	Not specified [97]
Dayou Du et al., 2025 [98]	No	Quantization artifacts [98]	RTX 4090, A100, H100 [98]
Kaihua Liang et al., 2026 [99]	No	Batching/scheduling [99]	RTX 4090, L40S GPUs [99]
Bohan Zhao et al., 2025 [100]	No	Batching/scheduling [100]	Not specified [100]
Zhaoyuan Su et al., 2025 [101]	Yes	Memory management, quantization artifacts [101]	General GPU environments [101]
Xiang-Kai Zhuge et al., 2025 [102]	No	Memory management, batching/scheduling [102]	Resource-constrained devices [102]
Jian Tian et al., 2025 [103]	No	Batching/scheduling [103]	H800 cluster [103]
Guohao Dai et al., 2025 [104]	No	Batching/scheduling, memory management [104]	Not specified [104]
Rui Li et al., 2025 [105]	No	Batching/scheduling [105]	Not specified [105]
Pengxiang Zhao et al., 2025 [106]	Yes	Quantization artifacts [106]	NVIDIA RTX 4090 GPU [106]
Thanh Le-Cong et al., 2024 [107]	Yes	Memory management [107]	State-of-the-art hardware, NVIDIA A100 GPU [107]
Xinming Wei et al., 2025 [108]	No	Batching/scheduling [108]	Intel Core Ultra SoC [108]
Xin Zhang et al., 2025 [109]	No	Batching/scheduling [109]	Not specified [109]
Zeyu Li et al., 2025 [110]	No	Quantization artifacts [110]	NVIDIA A100 GPU [110]
Jiakun Fan et al., 2025a [111]	No	Memory management [111]	RTX 4090, L40S GPUs [111]
Lei Gao et al., 2025 [112]	No	Batching/scheduling [112]	Not specified [112]
Zhibin Wang et al., 2025 [113]	Yes	Memory management [113]	General LLM systems [113]

Study	Full Text Retrieved?	Primary Focus	Hardware Environment
Yuning Zhang et al., 2025 [114]	No	Memory management, batching/scheduling [114]	Single-GPU servers [114]
Yuhang Li et al., 2025 [115]	No	Memory management [115]	Not specified [115]
Jinghan Yao et al., 2023 [116]	Yes	Batching/scheduling, memory management [116]	NVIDIA A100 GPUs, AMD EPYC [116]
William Meng et al., 2025 [117]	No	Memory management [117]	KV cache offloading systems [117]
A. Saxena et al., 2025 [118]	No	Memory management [118]	GPU-based systems [118]
Zeyu Zhang et al., 2024 [119]	No	Batching/scheduling [119]	Not specified [119]
Ruihao Li et al., 2025 [120]	Yes	Memory management [120]	General GPU systems [120]
Selin Yildirim et al., 2025 [121]	Yes	Memory management [121]	NVIDIA Titan RTX, AMD MI250 [121]
Limin Cheng et al., 2025 [122]	No	Batching/scheduling [122]	GPU servers [122]
Wei Gao et al., 2025 [123]	No	Memory management [123]	Not specified [123]
Ziyi Xu et al., 2025 [124]	No	Memory management [124]	8xH100 DGX system [124]
Jiazhi Jiang et al., 2026 [125]	No	Memory management [125]	Not specified [125]
Mingyu Yang et al., 2025 [126]	No	Batching/scheduling [126]	Not specified [126]
Mona Moghadampanah et al., 2025 [127]	No	Energy characterization across modalities [127]	NVIDIA A100 GPU [127]
Songze Liu et al., 2025 [128]	No	Memory management [128]	Large-scale LLM inference [128]
Songyu Zhang et al., 2026 [129]	No	Batching/scheduling, memory management [129]	Not specified [129]
Junhan Liao et al., 2025 [130]	No	Batching/scheduling [130]	Not specified [130]
Liang Zheng et al., 2026 [131]	No	Memory management [131]	Not specified [131]
Jianshu She et al., 2026 [132]	No	Batching/scheduling [132]	Not specified [132]
Lingfeng Tang et al., 2025 [133]	No	Memory management [133]	Not specified [133]
Dongha Yoon et al., 2025 [134]	No	Memory management [134]	Disaggregated LLM architecture [134]

Study	Full Text Retrieved?	Primary Focus	Hardware Environment
Haojun Xia et al., 2025 [135]	No	Memory management (quantization artifacts) [135]	Not specified [135]

The vast majority of included studies (135 total) focus on optimizing LLM inference infrastructure through system design improvements rather than documenting diagnostic cases where infrastructure problems were initially misattributed to model quality. Only two papers explicitly examined Apple Silicon hardware [1, 2], and none provided MLX-specific diagnostic case studies. Most studies with available full text (approximately 60%) concentrated on NVIDIA GPU environments [10–12, 15], with a smaller subset addressing heterogeneous or edge deployment scenarios [53, 59, 108].

## Infrastructure Issue Categories

The retrieved sources reveal five dominant categories of infrastructure issues that could manifest as apparent model degradation:

### Quantization and Dequantization Overhead

Multiple studies identified quantization-related infrastructure problems that could be mistaken for model quality issues. On Apple Silicon, significant dequantization overhead emerges at low bit precision, especially with codebook-based schemes [1]. Irregular bit widths (1/3/5 bits) prove less efficient due to increased instructions for unpacking [2]. The lack of dedicated hardware units for tensor-intensive workloads on Apple Silicon further compounds these issues [2].

Beyond Apple Silicon, INT4 quantization methods suffer from significant runtime overhead (20-90%) when dequantizing weights or partial sums on GPUs [12]. Weight-only quantization introduces instruction overhead, memory traffic, and pipeline stalls across all batch sizes [87]. Even 4-bit quantization can create performance paradoxes: while memory benefits appear substantial, computational load from verification processes can undermine these gains [72].

Specific manifestations include activation spikes in GLU variants causing severe local quantization errors [41], and the unique vulnerability of certain model series (like LLaMA3-70B) to W8A8 per-channel quantization due to weight distribution characteristics [73]. For multimodal LLMs, distributional disparities between visual and textual tokens create additional quantization challenges [62].

### Memory Management and KV Cache Bottlenecks

KV cache management emerged as a pervasive infrastructure concern across studies. The KV cache size can easily surpass model weight size [13], with activations becoming the dominant contributor to memory consumption during inference [14]. This creates a fundamental tension: memory-intensive KV cache limits batch size expansion under GPU memory constraints [19].

Several specific problems were identified. Traditional GPU-CPU memory swapping leads to higher latency and lower throughput due to waiting for data transfers [47]. Limited GPU memory capacity results in high I/O overhead and frequent cache misses with traditional approaches [115]. Cache eviction strategies create a difficult trade-off: recomputation can result in over 99% of processed tokens being redundant [45], while offloading to main memory introduces statefulness and load imbalances [45].

For edge and mobile deployments, persistent states across multiple invocations require careful memory orchestration [44]. The KV cache deletion immediately after each request prevents reuse across multi-turn conversations, significantly increasing computation costs [77].

### **Batching and Scheduling Inefficiencies**

Head-of-line blocking due to first-come-first-served scheduling emerged as a critical issue [7–9]. This manifests in several ways: existing scheduling algorithms treat LLM workloads as monolithic jobs without considering distinct prefill and decode phases [37], leading to severe interference when these phases are batched together [6, 29].

The prefill and decode phases exhibit fundamentally different characteristics. Prefill iterations have high latency but saturate GPU compute through parallel processing, while decode iterations have low latency but low compute utilization [15]. When these are interleaved without careful orchestration, the result is generation stalls and pipeline bubbles [15].

Request-length heterogeneity within batches creates additional problems, causing GPU underutilization and increased latency [71]. Prompt-length variations lead to distinct performance bottlenecks that unified scheduling policies fail to address [132]. For multimodal systems, the inability to co-batch vision encoding with LLM prefill or decoding forces inter-stage blocking [46].

### **Numerical Precision and Reproducibility**

Several studies documented non-determinism arising from infrastructure rather than model stochasticity. The same prompt can yield different outputs across runs due to floating-point non-associativity combined with dynamic batching and GPU kernels whose reduction orders vary with batch size [3, 21, 22]. For reasoning models like DeepSeek-R1-Distill-Qwen-7B, minor rounding differences in early tokens cascade into divergent chains of thought, producing up to 9% variation in accuracy and 9,000 tokens difference in response length under bfloat16 precision with greedy decoding [3].

These variations manifest across different system configurations including batch size, GPU count, and GPU type [3]. Tensor parallel size variations create particular challenges: training engines typically use TP=1 while inference uses multi-GPU TP for throughput, creating natural mismatches [22].

### **Padding and Token Handling**

Padding tokens, while theoretically masked, can influence computation when implementation errors allow them to leak through [38]. Even small amounts of padding shift hidden representations, degrade quality in smaller models, alter bias unpredictably, and weaken safety guardrails [38]. This represents a concrete case where infrastructure handling of seemingly inert tokens produces observable model behavior changes.

## **Apple Silicon and MLX-Specific Findings**

Only two papers provided detailed analysis of Apple Silicon infrastructure [1, 2], and neither addressed MLX framework specifically. The identified Apple Silicon-specific behaviors include:

### **Quantization Characteristics**

Apple Silicon exhibits significant dequantization overhead at low bit precision, particularly with codebook-based quantization schemes [1, 2]. Codebook-based schemes are slower on Apple Silicon compared to block-based schemes, with irregular bit widths (1/3/5 bits) proving less efficient due to increased unpacking instructions [2]. The empirical evaluation across 26 different model precisions found that block-based K-quants quantization schemes outperform codebook-based IQ quantization schemes on Apple Silicon [2].

### Memory Architecture Effects

The unified memory architecture of Apple Silicon reduces data movement overhead but becomes less efficient with larger working set sizes and higher TLB miss rates [2]. The memory management unit shows decreasing efficiency as working sets expand [1]. These characteristics create unique performance profiles compared to discrete GPU memory architectures.

### Performance and Cost Trade-offs

Apple Silicon demonstrates cost-effectiveness for ultra-large language models due to its unified memory architecture [2]. Lower power consumption and better energy efficiency compared to NVIDIA GPUs emerge as distinguishing features [2]. However, the lack of dedicated hardware units for tensor-intensive workloads creates performance limitations [1, 2].

One study developed a "Virtual Tensor Core" architecture for ARM64 microarchitectures including Apple Silicon, achieving stable throughput exceeding 60 tokens/second on M2 hardware through bypassing standard library containers and implementing hand-tuned NEON SIMD kernels [26, 26]. This met the 200ms psycholinguistic latency threshold without requiring opaque dependencies [26].

### Absence of Cross-Framework Comparisons

Critically, no studies provided direct comparisons between MLX and other inference frameworks (vLLM, TGI, llama.cpp) using identical model weights [1, 2]. The two Apple Silicon studies compared against NVIDIA GPUs but did not examine framework-specific implementation differences [1, 2]. This represents a significant gap in the literature for understanding whether apparent model behavior differences across platforms stem from infrastructure rather than numerical precision or algorithmic variations.

### Diagnostic Methodologies

The studies employed various approaches to isolate infrastructure from model issues, though few provided systematic frameworks specifically designed for this purpose.

### Controlled Experimental Approaches

Several studies used ablation experiments to isolate infrastructure effects. Yuan et al. conducted controlled experiments across 12 different runtime configurations, using metrics like standard deviation of accuracy and divergence index to distinguish infrastructure from model issues [3]. Additional experiments with HuggingFace Transformers ensured issues were not backend-specific [3].

Frantar et al. performed ablation studies modifying kernel parameters and used roofline analysis for computational efficiency [10], finding that neglecting certain design aspects significantly degrades performance [10]. Several others employed systematic comparisons against baselines to evaluate infrastructure fixes [25, 45, 70].

### Performance Modeling and Profiling

Sophisticated performance models enabled diagnosis of infrastructure bottlenecks. Gerogiannis et al. developed a novel Roof-Surface performance model—a 3D model providing insights into how memory resources, vector units, and hardware matrix engines interact [42]. Vellaisamy et al. introduced SKIP (System-Aware Kernel Inference Profiler) to analyze performance dynamics using metrics like Total Kernel Launch and Queuing Time (TKLQT) [66, 66].

Zhang et al. used a structurally-informed online performance model to provide accurate, forward-looking per-step latency and capacity estimations [27]. This enabled predictive orchestration rather than reactive load balancing [27].

### Metrics for Infrastructure vs Model Distinction

Studies employed various metrics to distinguish infrastructure from model issues. Common approaches included measuring throughput and latency changes while monitoring model quality through perplexity scores [11, 20, 23]. Some tracked fine-grained metrics like token throughput, KV cache memory access patterns, and load balancing of forward passes during different inference stages [82].

For reproducibility issues, divergence indices and standard deviation of accuracy across configurations helped identify infrastructure-related non-determinism [3]. Service-level metrics like SLO attainment rates distinguished infrastructure bottlenecks from model capability limitations [19, 59].

However, most papers did not explicitly document diagnostic timelines or breakthrough insights that led to correct diagnosis of infrastructure vs model causes [10, 20, 27]. The emphasis remained on presenting solutions rather than documenting diagnostic processes.

### Infrastructure Problems Manifesting as Model Issues

While few papers explicitly documented cases where infrastructure problems were initially misattributed to model defects, several described symptoms that could easily be confused:

#### Output Quality and Accuracy Impacts

Quantization-related infrastructure issues produce measurable accuracy degradation. Severe local quantization errors from activation spikes significantly degrade performance [41]. The LLaMA3-70B series exhibits unique accuracy degradation with W8A8 per-channel post-training quantization, with accuracy drops exceeding 3% for certain configurations [4, 73].

For diffusion-based LLMs, applying standard post-training quantization directly causes severe accuracy degradation and reduced generalization performance, with AWQ suffering a 16% accuracy drop under W4A4 [5]. Padding tokens, when not properly masked, shift hidden representations and degrade quality in smaller models [38].

#### Latency and Throughput Variations

Infrastructure choices create substantial performance variations. Yuan et al. documented up to 9% variation in accuracy and 9,000 tokens difference in response length for DeepSeek-R1-Distill-Qwen-7B due to differences in GPU count, type, and evaluation batch size under bfloat16 precision [3]. These variations appeared across different system configurations, making them easily attributable to model instability rather than infrastructure [3].

Memory management issues manifest as OOM errors and SLO violations [113], while batching problems create severe latency bottlenecks from head-of-line blocking [109]. Inappropriate batching strategies cause slowdowns ranging from 2x to 10x when mixing different request types [6].

#### Gradual vs Sudden Onset

Most documented symptoms appeared gradually rather than suddenly. Performance degradation from quantization errors accumulated and amplified progressively during iterative generation [5]. Cache eviction impacts escalated with compression ratio [43]. Speculative decoding performance degraded gradually in high-load environments with fixed speculative lengths [105].

The conditions triggering symptoms varied widely. Batch sizes proved critical for numerous issues: performance drops occurred when batch sizes exceeded specific thresholds (e.g., beyond 9 for certain edge workloads) [59]. Model sizes, specific quantization schemes, and prompt characteristics all influenced symptom appearance [1, 2, 16].



## Cross-Platform Behavior Comparison

The literature reveals a striking absence of direct cross-platform comparisons examining whether identical model weights produce different outputs across inference frameworks.

### Limited MLX Coverage

No studies provided specific comparisons between MLX and other frameworks like vLLM, TGI, or llama.cpp [1, 2]. The two Apple Silicon studies compared against NVIDIA GPUs in terms of performance metrics and cost-effectiveness but did not examine output equivalence or framework-specific implementation differences [1, 2].

### Hardware-Specific Comparisons

Several studies compared performance across different GPU architectures. Vellaisamy et al. compared loosely-coupled (PCIe A100/H100) and closely-coupled (GH200) systems [66], finding that closely-coupled GH200 significantly outperforms loosely-coupled systems at large batch sizes but remains CPU-bound at up to 4x larger batch sizes than LC systems [66]. The Grace CPU's lower single-thread performance contributes to higher inference latency at low batch sizes on GH200 [66].

Hong et al. evaluated FlashDecoding++ across NVIDIA (Tesla A100, RTX3090) and AMD (MI210, RX7900XTX) GPUs, achieving speedups up to 4.86× on NVIDIA and 3.93× on AMD [18]. However, these comparisons focused on performance rather than output equivalence.

### Framework Implementation Differences

Chen et al. noted differences in measurement methods and optimization levels when comparing MLC-LLM to their own backend [28], but did not investigate whether these implementation differences produced divergent outputs for identical inputs. This pattern held across the literature: framework comparisons measured performance metrics (throughput, latency) rather than output determinism [18, 27].

### Reproducibility Across Configurations

Zhang et al. documented that batch speculative decoding implementations violate output equivalence—the fundamental requirement that speculative decoding produce identical token sequences to standard autoregressive generation [95]. Several existing batch implementations improperly handle the ragged tensor problem, corrupting position IDs, attention masks, and KV-cache state [95]. This represents concrete evidence that infrastructure choices affect outputs, though the comparison was within a single framework rather than across platforms.

## Infrastructure Solutions and Validation

Studies implementing infrastructure fixes demonstrated substantial performance improvements, though validation of model quality preservation varied in rigor.

### Quantization Optimizations

For Apple Silicon specifically, the recommended solution was preferring block-based quantization schemes over codebook-based schemes [1, 2]. This recommendation emerged from comprehensive evaluation of hardware characteristics showing improved cost-effectiveness and reduced latency with appropriate quantization schemes [2].

More broadly, progressive quantization reduced dequantization overhead, with QServe achieving 1.2-3.5x higher throughput compared to TensorRT-LLM on A100 and L40S GPUs [12]. SqueezeLLM's sensitivity-based non-uniform quantization reduced the perplexity gap by up to 2.1x compared to baselines while achieving 2.3x speedup [11].

Advanced approaches combined multiple techniques. KVQuant incorporated per-channel key quantization, pre-RoPE key quantization, and non-uniform KV cache quantization, achieving less than 0.1 perplexity degradation with 3-bit quantization on Wikitext-2 and C4 [14]. XQuant demonstrated up to  $7.7\times$  memory savings with less than 0.1 perplexity degradation compared to FP16 baseline [20].

### Memory Management Solutions

Several systems addressed KV cache bottlenecks through architectural innovations. Scissorhands reduced KV cache memory usage by up to 5X without compromising model quality, achieving 20X compression when combined with 4-bit quantization [13]. Pie's performance-transparent swapping and adaptive expansion outperformed vLLM by up to 1.9X in throughput and 2X in latency [47].

For multi-tier memory systems, eLLM achieved 2.32x higher decoding throughput and supported 3x larger batch sizes for 128K-token inputs through virtual tensor abstraction and elastic memory mechanisms [24]. SYMPHONY demonstrated that proactive cache management based on advisory requests enables handling 8x more requests compared to state-of-the-art baselines with similar latency profiles [45].

### Scheduling and Batching Improvements

Chunked-prefills and stall-free batching in Sarathi-Serve improved throughput by up to 2.6x for Mistral-7B and 6.9x for Falcon-180B [15]. The system created stall-free schedules that add new requests without pausing ongoing decodes [15].

Disaggregation strategies proved effective for reducing interference. TetriInfer's partitioning of prompts into fixed-size chunks and separation of prefill/decode instances improved TTFT by 97%, JCT by 47%, while using 38% fewer resources [6]. HydraInfer's Hybrid Encode-Prefill-Decode disaggregation achieved up to 4x higher inference throughput compared to state-of-the-art systems [35].

Dynamic and adaptive approaches outperformed static policies. FairBatching reduced TTFT tail latency by up to 2.29x while robustly maintaining TPOT SLOs, achieving 20% improvement in single-node capacity and 54.3% improvement in cluster-level capacity [60]. CascadeInfer's length-aware scheduling reduced end-to-end latency by up to 67% and tail latency by up to 69% while improving system throughput by up to 2.89 times [71].

### Validation Rigor Varies

Most studies validated infrastructure fixes through before/after performance comparisons rather than explicit model quality verification. Common validation approaches included:

- Perplexity measurements on standard datasets (Wikitext-2, C4) to confirm minimal accuracy degradation [11, 14, 20]
- Throughput and latency benchmarks against baselines [12, 15, 47]
- SLO attainment rates and goodput metrics [19, 70]
- End-to-end evaluations on diverse workloads [35, 53]

Yuan et al. provided more rigorous validation for their LayerCast solution by showing it achieved stability nearly identical to FP32, with divergence rates below 3.4% while reducing memory usage by 34% [3]. Zhang et al. confirmed zero probability divergence and bit-wise reproducibility for their deterministic inference approach [22].

However, many studies did not explicitly validate that infrastructure fixes preserved model quality [26, 54, 112], instead focusing on performance improvements with implicit assumptions that model behavior remained unchanged.

## Generalizable Patterns and Prevention

While most papers focused on solutions rather than diagnostic processes, several patterns emerged for preventing infrastructure vs model confusion:

### Systematic Testing Frameworks

Dynamic batching methods that continuously monitor memory utilization and adhere to SLAs demonstrated 8-28% throughput gains and 22% capacity improvements [40]. APEX simulator enabled identifying optimal parallel execution plans 71x faster and 1234x more cost-effectively than actual deployment on GPU clusters [67].

For memory management, virtual tensor abstraction that decouples virtual address space from physical GPU memory enables more flexible resource allocation [24]. Elastic memory mechanisms supporting dynamic inflation and deflation provide runtime adaptability [24].

### Best Practices for Decoupling

Several practices emerged for separating infrastructure from model testing:

- Phase separation: Disaggregating prefill and decode instances allows each to run independently with tailored resource allocation [6, 114]
- Predictive orchestration: Shifting from reactive load balancing to predictive scheduling based on forward-looking performance models [27]
- Adaptive policies: Dynamic adjustment of parameters (batch size, precision, speculation length) based on real-time conditions rather than static configurations [101, 105]
- Explicit prefix management: Identifying common prefixes globally rather than relying on implicit LRU-based caching [25]

### Early Warning Indicators

Infrastructure problems often announce themselves through specific patterns:

- Severe cache pollution from highly irregular and bursty access patterns [128]
- GPU underutilization despite apparent compute load, indicating scheduling inefficiencies [56]
- Non-monotonic relationships between expected and actual performance (e.g., increased precision producing slower inference) [64]
- Activation spikes in specific layers, particularly early and late layers [41]
- Memory bandwidth bottlenecks with 99% of latency spent on transfers and GPU consuming only 28% of rated TDP [117]

### Recommended Diagnostic Workflows

Several workflows proved effective for isolating infrastructure issues:

1. Profile fine-grained metrics across inference stages (prefill vs decode) to identify phase-specific bottlenecks [66, 82]
2. Compare performance across different batch sizes and hardware configurations to expose scaling issues [3, 66]
3. Use roofline models or performance ceilings to identify whether systems are compute-bound or memory-bound [10, 56]
4. Implement ablation studies that systematically disable optimization components to measure their individual contributions [10, 24]
5. Monitor resource utilization (GPU compute, memory bandwidth, cache residency) alongside quality metrics to detect mismatches [1, 2]

## Tools and Techniques

Reusable tools include profilers like SKIP for analyzing operator-kernel dynamics [66], instrumentation frameworks for measuring KV cache access patterns [82], and simulators like APEX that enable cost-effective exploration of parallel execution strategies [67].

Performance modeling tools—including roofline analysis [10], latency-power models for different configurations [65], and online performance predictors [27]—help distinguish infrastructure limitations from model characteristics.

## Lessons for Future Deployments

For Apple Silicon and similar unified memory architectures, several lessons emerged:

- Quantization scheme selection critically impacts performance; block-based approaches outperform codebook-based on Apple Silicon [1, 2]
- Unified memory efficiency decreases with larger working sets, requiring careful management [2]
- Lack of dedicated tensor units necessitates software optimization through hand-tuned kernels [26]
- Direct memory mapping and SIMD optimization can achieve practical performance despite architectural limitations [26]

More broadly, phase-specific optimization proves essential. Prefill and decode phases have distinct resource demands requiring separate handling [112]. Dynamic resource allocation that adapts to real-time conditions outperforms static configurations [101, 113]. Predictive signals enable proactive scheduling that prevents bottlenecks rather than reacting to them [27, 64].

## Synthesis

The evidence reveals a fundamental gap: while the hypothesis posits that MLX-specific infrastructure issues can mimic model defects, the literature provides minimal direct evidence of this phenomenon. Instead, the findings demonstrate that **infrastructure effects are well-recognized in NVIDIA GPU environments but remain largely uncharacterized for Apple Silicon and MLX.**

### Confirmed Infrastructure Effects in General

The evidence strongly confirms that infrastructure issues manifest as apparent model problems in CUDA-based environments. Non-determinism from floating-point precision and batching creates output variations up to 9% in accuracy despite identical models and prompts [3, 3]. Quantization overhead produces measurable accuracy drops exceeding 3% for certain configurations [4]. Batching and scheduling inefficiencies cause performance degradations ranging from 2x to 10x slowdowns [6].

These effects are infrastructure-related because:

- They disappear with infrastructure changes (different batching, higher precision) while keeping model weights constant [3, 6]
- They vary systematically with system configuration (batch size, GPU count) rather than input semantics [3, 16]
- They can be resolved through scheduling, memory management, or kernel optimization without model re-training [15, 19, 45]

### Apple Silicon: Characterized but Not Diagnosed

The two Apple Silicon studies characterize performance and recommend optimizations but do not document diagnostic cases. Benazir et al. measured quantization scheme impacts and recommended block-based approaches [1, 2], but did not report instances where poor quantization choice was initially attributed to model quality. The studies establish that:

- Dequantization overhead is significant on Apple Silicon [1, 2]
- Unified memory architecture behaves differently than discrete GPU memory [2]
- Specific quantization schemes are better suited to the hardware [2]

However, neither study provides before/after diagnostics showing infrastructure fixes resolving apparent model degradation [1, 2]. The gap between characterization and diagnostic case reporting is substantial.

### **MLX Framework: Entirely Absent**

No retrieved study examined MLX framework specifically. Searches for "MLX vs vLLM," "MLX quantization," "Metal shader compilation," or "MLX inference behavior" returned no results [1] through [135]. This absence is particularly notable given that MLX represents a major inference framework for Apple Silicon, suggesting either:

1. MLX infrastructure issues have not been systematically studied
2. MLX issues have not yet manifested as apparent model defects requiring investigation
3. Such cases exist but remain in internal engineering reports rather than published literature

### **Mechanism-Based Explanation of Heterogeneity**

The apparent contradiction—strong evidence for infrastructure effects in CUDA environments versus absence of MLX evidence—likely reflects research ecosystem dynamics rather than fundamental differences. NVIDIA GPU environments dominate production LLM serving, creating more opportunities to observe and document infrastructure issues. The relative maturity of CUDA-based frameworks (vLLM, TensorRT-LLM) means their failure modes are better characterized.

Apple Silicon's unified memory architecture and Metal shader compilation could create distinct failure modes that differ from CUDA environments. For example, dequantization overhead manifests differently [1, 2], and memory pressure behaviors on unified memory likely differ from discrete GPU memory [2]. However, without MLX-specific diagnostic studies, these remain hypothetical differences.

### **Implications for Diagnosis**

The findings suggest a generalizable diagnostic approach applicable to MLX despite the absence of MLX-specific case studies:

1. **Suspect infrastructure when:** Variations correlate with system configuration (batch size, GPU count) rather than input semantics [3]; performance changes occur without model weight changes [6]; issues appear at specific batch size or sequence length thresholds [59]
2. **Validate through isolation:** Compare identical weights across different infrastructure configurations [3]; use ablation studies to disable optimization components [10, 24]; profile resource utilization alongside quality metrics [66, 82]
3. **Platform-specific considerations:** For Apple Silicon, monitor dequantization overhead [1], track memory pressure on unified architecture [2], measure performance across quantization schemes [1, 2]

The evidence indicates that infrastructure effects are real and significant, but the research community has not yet systematically investigated whether MLX exhibits the same patterns, different patterns, or introduces novel failure

modes unique to its architecture.

## References

1. A. Benazir, Felix Xiaozhu Lin (2025) Profiling Large Language Model Inference on Apple Silicon: A Quantization Perspective. arXivorg. <https://doi.org/10.48550/arXiv.2508.08531>
2. A. Benazir, Felix Xiaozhu Lin (2025) Benchmarking and Characterization of Large Language Model Inference on Apple Silicon. Proceedings of the ACM on Measurement and Analysis of Computing Systems. <https://doi.org/10.1145/3771563>
3. Jiayi Yuan, Hao Li, Xinheng Ding, et al (2025) Understanding and Mitigating Numerical Sources of Nondeterminism in LLM Inference
4. Jinhao Li, Jiaming Xu, Shiyao Li, et al (2023) Fast and Efficient 2-Bit LLM Inference on GPU: 2/4/16-Bit in a Weight Matrix with Asynchronous Dequantization. International Conference on Computer Aided Design. <https://doi.org/10.1145/3676536.3676796>
5. Chen Xu, Dawei Yang (2025) DLLMQuant: Quantizing Diffusion-based Large Language Models. arXivorg. <https://doi.org/10.48550/arXiv.2508.14090>
6. Cunchen Hu, HeYang Huang, Liangliang Xu, et al (2024) Inference without Interference: Disaggregate LLM Inference for Mixed Downstream Workloads. arXivorg. <https://doi.org/10.48550/arXiv.2401.11181>
7. Archit Patke, Dharmath Reddy, Saurabh Jha, et al (2024) Queue Management for SLO-Oriented Large Language Model Serving. ACM Symposium on Cloud Computing. <https://doi.org/10.1145/3698038.369852>
8. Youpeng Zhao, Jun Wang (2024) ALISE: Accelerating Large Language Model Serving with Speculative Scheduling. International Conference on Computer Aided Design. <https://doi.org/10.1145/3676536.3676659>
9. Bronislav Sidik, Chaya Levi, Joseph Kampeas (2026) EWSJF: An Adaptive Scheduler with Hybrid Partitioning for Mixed-Workload LLM Inference
10. Elias Frantar, Roberto L. Castro, Jiale Chen, et al (2024) MARLIN: Mixed-Precision Auto-Regressive Parallel Inference on Large Language Models. ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming. <https://doi.org/10.1145/3710848.3710871>
11. Sehoon Kim, Coleman Hooper, A. Gholami, et al (2023) SqueezeLLM: Dense-and-Sparse Quantization. International Conference on Machine Learning. <https://doi.org/10.48550/arXiv.2306.07629>
12. Yujun Lin, Haotian Tang, Shang Yang, et al (2024) QServe: W4A8KV4 Quantization and System Co-design for Efficient LLM Serving. Conference on Machine Learning and Systems. <https://doi.org/10.48550/arXiv.2405.04532>

13. Zichang Liu, Aditya Desai, Fangshuo Liao, et al (2023) Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. Neural Information Processing Systems. <https://doi.org/10.48550/arXiv.2305.17118>
14. Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, et al (2024) KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. Neural Information Processing Systems. <https://doi.org/10.48550/arXiv.2401.18079>
15. Amey Agrawal, Nitin Kedia, Ashish Panwar, et al (2024) Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. USENIX Symposium on Operating Systems Design and Implementation. <https://doi.org/10.48550/arXiv.2403.02310>
16. Biao Sun, Ziming Huang, Hanyu Zhao, et al (2024) Llumnix: Dynamic Scheduling for Large Language Model Serving. USENIX Symposium on Operating Systems Design and Implementation. <https://doi.org/10.48550/arXiv.2406.03243>
17. Guangba Yu, Zirui Wang, Yujie Huang, et al (2026) Why Does the LLM Stop Computing: An Empirical Study of User-Reported Failures in Open-Source LLMs
18. Ke Hong, Guohao Dai, Jiaming Xu, et al (2023) FlashDecoding++: Faster Large Language Model Inference on GPUs. arXivorg. <https://doi.org/10.48550/arXiv.2311.01282>
19. Shihong Gao, Xin Zhang, Yanyan Shen, Lei Chen (2025) Apt-Serve: Adaptive Request Scheduling on Hybrid Cache for Scalable LLM Inference Serving. Proc ACM Manag Data. <https://doi.org/10.1145/3725394>
20. Aditya Tomar, Coleman Hooper, Minjae Lee, et al (2025) XQuant: Breaking the Memory Wall for LLM Inference with KV Cache Rematerialization. arXivorg. <https://doi.org/10.48550/arXiv.2508.10395>
21. Raja Gond, Aditya K Kamath, Arkaprava Basu, et al (2026) LLM-42: Enabling Determinism in LLM Inference with Verified Speculation
22. Ziyang Zhang, Xinheng Ding, Jiayi Yuan, et al (2025) Deterministic Inference across Tensor Parallel Sizes That Eliminates Training-Inference Mismatch. arXivorg. <https://doi.org/10.48550/arXiv.2511.17826>
23. Dibakar Gope, David Mansell, Danny Loh, Ian Bratt (2024) Highly Optimized Kernels and Fine-Grained Codebooks for LLM Inference on Arm CPUs. arXivorg. <https://doi.org/10.48550/arXiv.2501.00032>
24. Jiale Xu, Rui Zhang, Yi Xiong, et al (2025) eLLM: Elastic Memory Management Framework for Efficient LLM Serving. arXivorg. <https://doi.org/10.48550/arXiv.2506.15155>
25. Zhen Zheng, Xin Ji, Taosong Fang, et al (2024) BatchLLM: Optimizing Large Batched LLM Inference with Global Prefix Sharing and Throughput-oriented Token Batching. arXivorg. <https://doi.org/10.48550/arXiv.2412.03594>

26. Bugra Kiliclas, Faruk Alpay (2026) Bare-Metal Tensor Virtualization: Overcoming the Memory Wall in Edge-AI Inference on ARM64
27. Yue Zhang, Yuansheng Chen, Xuan Mo, et al (2025) A Predictive and Synergistic Two-Layer Scheduling Framework for LLM Serving. arXivorg. <https://doi.org/10.48550/arXiv.2509.23384>
28. Jian Chen, Vashisth Tiwari, Ranajoy Sadhukhan, et al (2024) MagicDec: Breaking the Latency-Throughput Tradeoff for Long Context Generation with Speculative Decoding. International Conference on Learning Representations. <https://doi.org/10.48550/arXiv.2408.11049>
29. Cunchen Hu, HeYang Huang, Liangliang Xu, et al (2025) ShuffleInfer: Disaggregate LLM Inference for Mixed Downstream Workloads. ACM Transactions on Architecture and Code Optimization (TACO). <https://doi.org/10.1145/3732941>
30. Ying Wang, Zhengxu Jin, Jiexiong Xu, et al (2025) AugServe: Adaptive Request Scheduling for Augmented Large Language Model Inference Serving. arXivorg. <https://doi.org/10.48550/arXiv.2512.04013>
31. Kanishk Goel, Jayashree Mohan, Nipun Kwatra, et al (2025) Niyama : Breaking the Silos of LLM Inference Serving. arXivorg. <https://doi.org/10.48550/arXiv.2503.22562>
32. Yeonhong Park, Jake Hyun, Hojoon Kim, Jae W. Lee (2024) DecDEC: A Systems Approach to Advancing Low-Bit LLM Quantization. USENIX Symposium on Operating Systems Design and Implementation
33. Kyungmin Bin, Seungbeom Choi, Jimyoung Son, et al (2025) FineServe: Precision-Aware KV Slab and Two-Level Scheduling for Heterogeneous Precision LLM Serving. arXivorg. <https://doi.org/10.48550/arXiv.2509.06261>
34. Yiyuan He, Minxian Xu, Jingfeng Wu, et al (2024) UELLM: A Unified and Efficient Approach for LLM Inference Serving. arXivorg. <https://doi.org/10.48550/arXiv.2409.14961>
35. Xianzhe Dong, Tongxuan Liu, Yuting Zeng, et al (2025) HydraInfer: Hybrid Disaggregated Scheduling for Multimodal Large Language Model Serving. arXivorg. <https://doi.org/10.48550/arXiv.2505.12658>
36. Chaoyi Ruan, Yinhe Chen, Dongqi Tian, et al (2025) DynaServe: Unified and Elastic Execution for Dynamic Disaggregated LLM Serving
37. Kunal Jain, Anjaly Parayil, Ankur Mallick, et al (2024) Intelligent Router for LLM Workloads: Improving Performance Through Workload-Aware Load Balancing
38. Rom Himelstein, Amit Levi, Yonatan Belinkov, Avi Mendelson (2025) Silent Tokens, Loud Effects: Padding in LLMs. arXivorg. <https://doi.org/10.48550/arXiv.2510.01238>
39. Wenyuan Liu, Haoqian Meng, Yilun Luo, et al (2025) MicroMix: Efficient Mixed-Precision Quantization with Microscaling Formats for Large Language Models. arXivorg. <https://doi.org/10.48550/arXiv.2508.02343>



40. Bowen Pang, Kai Li, Feifan Wang (2025) Optimizing LLM Inference Throughput via Memory-aware and SLA-constrained Dynamic Batching. arXivorg. <https://doi.org/10.48550/arXiv.2503.05248>
41. Jaewoo Yang, Hayun Kim, Younghoon Kim (2024) Mitigating Quantization Errors Due to Activation Spikes in GLU-Based LLMs. arXivorg. <https://doi.org/10.48550/arXiv.2405.14428>
42. Gerasimos Gerogiannis, Stijn Eyerman, E. Georganas, et al (2025) DECA: A Near-Core LLM Decompression Accelerator Grounded on a 3D Roofline Model. Micro. <https://doi.org/10.1145/3725843.3756073>
43. Yuxin Zhang, Yuxuan Du, Gen Luo, et al (2024) CaM: Cache Merging for Memory-efficient LLMs Inference. International Conference on Machine Learning
44. Wangsong Yin, Mengwei Xu, Yuanchun Li, Xuanzhe Liu (2024) LLM as a System Service on Mobile Devices. arXivorg. <https://doi.org/10.48550/arXiv.2403.11805>
45. Saurabh Agarwal, Anyong Mao, Aditya Akella, Shivaram Venkataraman (2024) SYMPHONY: Improving Memory Management for LLM Inference Workloads. arXivorg. <https://doi.org/10.48550/arXiv.2412.16434>
46. Lingxiao Zhao, Haoran Zhou, Yuezhi Che, Dazhao Cheng (2025) Enabling Disaggregated Multi-Stage MLLM Inference via GPU-Internal Scheduling and Resource Sharing. arXivorg. <https://doi.org/10.48550/arXiv.2512.17574>
47. Yi Xu, Ziming Mao, Xiangxi Mo, et al (2024) Pie: Pooling CPU Memory for LLM Inference. arXivorg. <https://doi.org/10.48550/arXiv.2411.09317>
48. H. Chen, Fuwen Tan, Alexandros Kouris, et al (2024) Progressive Mixed-Precision Decoding for Efficient LLM Inference. International Conference on Learning Representations. <https://doi.org/10.48550/arXiv.2410.13461>
49. Bhala Ranganathan, Mickey Zhang, Kai Wu (2025) Enhancing reliability in AI inference services: An empirical study on real production incidents. arXivorg. <https://doi.org/10.48550/arXiv.2511.07424>
50. Marco Federici, Davide Belli, M. V. Baalen, et al (2024) Efficient LLM Inference using Dynamic Input Pruning and Cache-Aware Masking. Conference on Machine Learning and Systems. <https://doi.org/10.48550/arXiv.2412.01380>
51. Mohammad Siavashi, Faezeh Keshmiri Dindarloo, Dejan Kostić, Marco Chiesa (2025) Priority-Aware Preemptive Scheduling for Mixed-Priority Workloads in MoE Inference. EuroMLSys. <https://doi.org/10.1145/3721146.3721956>
52. Yiyuan He, Minxian Xu, Jingfeng Wu, et al (2024) UELLM: A Unified and Efficient Approach for Large Language Model Inference Serving. International Conference on Service Oriented Computing. [https://doi.org/10.1007/978-981-96-0805-8\\_16](https://doi.org/10.1007/978-981-96-0805-8_16)
53. Jiakun Fan, Yanglin Zhang, Xiangchen Li, Dimitrios S. Nikolopoulos (2025) APEX: Asynchronous Parallel CPU-GPU Execution for Online LLM Inference on Constrained GPUs

54. Xiaoxiang Shi, Colin Cai, Junjia Du, Zhihao Jia (2025) Nexus: Proactive Intra-GPU Disaggregation of Prefill and Decode in LLM Serving
55. Adam Karvonen, Daniel Reuter, Roy Rinberg, et al (2025) DiFR: Inference Verification Despite Nondeterminism. arXivorg. <https://doi.org/10.48550/arXiv.2511.20621>
56. Zejia Lin, Hongxin Xu, Guanyi Chen, et al (2025) Boosting LLM Serving through Spatial-Temporal GPU Resource Sharing
57. Yufeng Gu, Alireza Khadem, Sumanth Umesh, et al (2025) PIM Is All You Need: A CXL-Enabled GPU-Free System for Large Language Model Inference. International Conference on Architectural Support for Programming Languages and Operating Systems. <https://doi.org/10.1145/3676641.3716267>
58. Yuang Chen, Wenqi Zeng, Jeffrey Xu Yu (2026) High-Throughput Non-uniformly Quantized 3-bit LLM Inference. ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming. <https://doi.org/10.1145/3774934.3786423>
59. Pan Zhou, Yiming Lei, Ling Liu, et al (2025) SLICE: SLO-Driven Scheduling for LLM Inference on Edge Computing Devices. arXivorg. <https://doi.org/10.48550/arXiv.2510.18544>
60. Hongtao Lyu, Boyue Liu, Mingyu Wu, Haibo Chen (2025) FairBatching: Fairness-Aware Batch Formation for LLM Inference. arXivorg. <https://doi.org/10.48550/arXiv.2510.14392>
61. Ruidong Zhu, Ziheng Jiang, Chao Jin, et al (2025) MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism. arXivorg. <https://doi.org/10.48550/arXiv.2504.02263>
62. Jiangyong Yu, Sifan Zhou, Dawei Yang, et al (2025) MQuant: Unleashing the Inference Potential of Multimodal Large Language Models via Static Quantization. ACM Multimedia. <https://doi.org/10.1145/3746027.3755433>
63. Woohyung Choi, Jinwoo Jeong, Hanhwi Jang, Jeongseob Ahn (2025) GPU-Centric Memory Tiering for LLM Serving With NVIDIA Grace Hopper Superchip. IEEE computer architecture letters. <https://doi.org/10.1109/LCA.2025.3533588>
64. Tiannuo Yang, Zebin Yao, Bowen Jin, et al (2025) Demystifying and Enhancing the Efficiency of Large Language Model Based Search Agents. arXivorg. <https://doi.org/10.48550/arXiv.2505.12065>
65. Qunyou Liu, Darong Huang, Marina Zapater, David Atienza (2025) GreenLLM: SLO-Aware Dynamic Frequency Scaling for Energy-Efficient LLM Serving. arXivorg. <https://doi.org/10.48550/arXiv.2508.16449>
66. P. Vellaisamy, Thomas Labonte, Sourav Chakraborty, et al (2025) Characterizing and Optimizing LLM Inference Workloads on CPU-GPU Coupled Architectures. IEEE International Symposium on Performance Analysis of Systems and Software. <https://doi.org/10.1109/ISPASS64960.2025.00015>
67. Yi-Chien Lin, Woosuk Kwon, Ronald Pineda, Fanny Nina Paravecino (2024) APEX: An Extensible and Dynamism-Aware Simulator for Automated Parallel Execution in LLM Serving

68. Sudhanshu Gupta, Sandhya Dwarkadas (2025) Improving the Performance of Out-of-Core LLM Inference Using Heterogeneous Host Memory. IEEE International Symposium on Workload Characterization. <https://doi.org/10.1109/IISWC66894.2025.00035>
69. Gerasimos Gerogiannis, Stijn Eyerman, E. Georganas, et al (2025) DECA: A Near-Core LLM Decompression Accelerator Supporting Out-of-Order Invocation. arXivorg. <https://doi.org/10.48550/arXiv.2505.19349>
70. Ruihao Gong, Shihao Bai, Siyu Wu, et al (2025) Past-Future Scheduler for LLM Serving under SLA Guarantees. International Conference on Architectural Support for Programming Languages and Operating Systems. <https://doi.org/10.1145/3676641.3716011>
71. Yitao Yuan, Chenqi Zhao, Bohan Zhao, et al (2025) CascadeInfer: Low-Latency and Load-Balanced LLM Serving via Length-Aware Scheduling
72. Yudi Zhang, Weilin Zhao, Xu Han, et al (2025) Speculative Decoding Meets Quantization: Compatibility Evaluation and Hierarchical Framework Design. arXivorg. <https://doi.org/10.48550/arXiv.2505.22179>
73. Ming-Fang Qin (2024) The Uniqueness of LLaMA3-70B Series with Per-Channel Quantization
74. Han Guo, William Brandon, Radostin Cholakov, et al (2024) Fast Matrix Multiplications for Lookup Table-Quantized LLMs. Conference on Empirical Methods in Natural Language Processing. <https://doi.org/10.48550/arXiv.2407.10960>
75. Kexin Chu, Dawei Xiang, Zixu Shen, et al (2025) Dynamic Expert Quantization for Scalable Mixture-of-Experts Inference. arXivorg. <https://doi.org/10.48550/arXiv.2511.15015>
76. Vima Gupta, Kartik Sinha, Ada Gavrilovska, A. Iyer (2024) Lynx: Enabling Efficient MoE Inference through Dynamic Batch-Aware Expert Selection. arXivorg. <https://doi.org/10.48550/arXiv.2411.08982>
77. Kexin Chu, Zixu Shen, Sheng-Ru Cheng, et al (2025) MCaM : Efficient LLM Inference with Multi-tier KV Cache Management. IEEE International Conference on Distributed Computing Systems. <https://doi.org/10.1109/ICDCS63083.2025.00062>
78. Yi Su, Yuechi Zhou, Quantong Qiu, et al (2025) Accurate KV Cache Quantization with Outlier Tokens Tracing. Annual Meeting of the Association for Computational Linguistics. <https://doi.org/10.48550/arXiv.2505.10938>
79. Rana Shahout, Cong Liang, Shiji Xin, et al (2024) Fast Inference for Augmented Large Language Models. arXivorg. <https://doi.org/10.48550/arXiv.2410.18248>
80. Zedong Liu, Shenggan Cheng, Guangming Tan, et al (2025) ElasticMM: Efficient Multimodal LLMs Serving with Elastic Multimodal Parallelism. arXivorg. <https://doi.org/10.48550/arXiv.2507.10069>
81. Guanxi Lu, H. Chen, Yuto Karashima, et al (2025) AdaBlock-dLLM: Semantic-Aware Diffusion LLM Inference via Adaptive Block Size. arXivorg. <https://doi.org/10.48550/arXiv.2509.26432>

82. Jie Ye, J. Cernuda, Avinash Maurya, et al (2025) Characterizing the Behavior and Impact of KV Caching on Transformer Inferences Under Concurrency. IEEE International Parallel and Distributed Processing Symposium. <https://doi.org/10.1109/IPDPS64566.2025.00108>
83. Tianyu Guo, Xianwei Zhang, Jiangsu Du, et al (2025) gLLM: Global Balanced Pipeline Parallelism Systems for Distributed LLMs Serving with Token Throttling. International Conference on Software Composition. <https://doi.org/10.1145/3712285.3759823>
84. Agrim Bari, Parikshit Hegde, Gustavo de Veciana (2025) Optimal Scheduling Algorithms for LLM Inference: Theory and Practice. Proceedings of the ACM on Measurement and Analysis of Computing Systems. <https://doi.org/10.1145/3771574>
85. Yin Du, Jiayi Ren, Xiayu Sun, et al (2026) LatencyPrism: Online Non-intrusive Latency Sculpting for SLO-Guaranteed LLM Inference
86. Rami Naeem, Tengis Buyantogtokh, Hamada Rizk, et al (2026) Transformer-Based Resource and Stage-Aware Scheduling for Model-Parallel LLM Inference. ICDCN Companion. <https://doi.org/10.1145/3737611.3776613>
87. Minki Jeong, Daegun Yoon, Soohong Ahn, et al (2025) StreamDQ: HBM-Integrated On-the-Fly DeQuantization via Memory Load for Large Language Models. IEEE computer architecture letters. <https://doi.org/10.1109/LCA.2025.3626929>
88. Vage Egiazarian, Roberto L. Castro, Denis Kuznedelev, et al (2025) Bridging the Gap Between Promise and Performance for Microscaling FP4 Quantization. arXiv.org. <https://doi.org/10.48550/arXiv.2509.23202>
89. Rongxiang Wang, Kangyuan Shu, F. Lin (2025) Enabling Dynamic Sparsity in Quantized LLM Inference. arXiv.org. <https://doi.org/10.48550/arXiv.2511.04477>
90. Nikoleta Iliakopoulou, Jovan Stojkovic, Chloe Alverti, et al (2024) Chameleon: Adaptive Caching and Scheduling for Many-Adapter LLM Inference Environments. Micro. <https://doi.org/10.1145/3725843.3756083>
91. Yitao Hu, Xiulong Liu, Guotao Yang, et al (2025) TightLLM: Maximizing Throughput for LLM Inference via Adaptive Offloading Policy. IEEE transactions on computers. <https://doi.org/10.1109/TC.2025.3558009>
92. Lian Liu, Long Cheng, Haimeng Ren, et al (2024) COMET: Towards Practical W4A4KV4 LLMs Serving. International Conference on Architectural Support for Programming Languages and Operating Systems. <https://doi.org/10.1145/3676641.3716252>
93. Hamidreza Imani, Jiaxin Peng, Peiman Mohseni, et al (2025) QoS-Efficient Serving of Multiple Mixture-of-Expert LLMs Using Partial Runtime Reconfiguration. International Conference on Machine Learning. <https://doi.org/10.48550/arXiv.2505.06481>
94. Gursimran Singh, Xinglu Wang, Yifan Hu, et al (2024) Efficiently Serving Large Multimodal Models Using EPD Disaggregation. International Conference on Machine Learning

95. Ranran Haoran Zhang, Soumik Dey, Ashirbad Mishra, et al (2025) Batch Speculative Decoding Done Right. arXivorg. <https://doi.org/10.48550/arXiv.2510.22876>
96. Wanyi Zheng, Minxian Xu, Shengye Song, Kejiang Ye (2025) BucketServe: Bucket-Based Dynamic Batching for Smart and Efficient LLM Inference Serving. 2025 IEEE International Conferences on Internet of Things (iThings) IEEE Green Computing & Communications (GreenCom) IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics). <https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics67059.2025.00041>
97. Rui Zhu, Ziheng Jiang, Chao Jin, et al (2025) MegaScale-Infer: Efficient Mixture-of-Experts Model Serving with Disaggregated Expert Parallelism. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. <https://doi.org/10.1145/3718958.3750506>
98. Dayou Du, Shijie Cao, Jianyi Cheng, et al (2025) BitDecoding: Unlocking Tensor Cores for Long-Context LLMs with Low-Bit KV Cache
99. Kaihua Liang, Xin Tan, An Zhong, et al (2026) FOCUS: DLLMs Know How to Tame Their Compute Bound
100. Bohan Zhao, Zane Cao, Yongchao He (2025) SIMPLE: Disaggregating Sampling from GPU Inference into a Decision Plane for Faster Distributed LLM Serving. arXivorg. <https://doi.org/10.48550/arXiv.2512.00719>
101. Zhaoyuan Su, Zeyu Zhang, Tingfeng Lan, et al (2025) MorphServe: Efficient and Workload-Aware LLM Serving via Runtime Quantized Layer Swapping and KV Cache Resizing
102. Xiang-Kai Zhuge, Shen Xu, Zeyu Wang, et al (2025) SpecOffload: Unlocking Latent GPU Capacity for LLM Inference on Resource-Constrained Devices. arXivorg. <https://doi.org/10.48550/arXiv.2505.10259>
103. Jian Tian, Shuailong Li, Yang Cao, et al (2025) Staggered Batch Scheduling: Co-optimizing Time-to-First-Token and Throughput for High-Efficiency LLM Inference. arXivorg. <https://doi.org/10.48550/arXiv.2512.16134>
104. Guohao Dai, Ke Hong, Qiuli Mao, et al (2025) FlashDecoding++Next: High Throughput LLM Inference With Latency and Memory Optimization. IEEE transactions on computers. <https://doi.org/10.1109/TC.2025.3585339>
105. Rui Li, Zhaoning Zhang, Libo Zhang, et al (2025) Nightjar: Dynamic Adaptive Speculative Decoding for Large Language Models Serving. arXivorg. <https://doi.org/10.48550/arXiv.2512.22420>
106. Pengxiang Zhao, Xiaoming Yuan (2025) GANQ: GPU-Adaptive Non-Uniform Quantization for Large Language Models. International Conference on Machine Learning. <https://doi.org/10.48550/arXiv.2501.12956>
107. Thanh Le-Cong, Bach Le, Toby Murray (2024) Memory-Efficient Large Language Models for Program Repair with Semantic-Guided Patch Generation

108. Xinming Wei, Jiahao Zhang, Haoran Li, et al (2025) Agent.xpu: Efficient Scheduling of Agentic LLM Workloads on Heterogeneous SoC. arXivorg. <https://doi.org/10.48550/arXiv.2506.24045>
109. Xin Zhang, Shihong Gao, Yanyan Shen, et al (2025) RelServe: Fast LLM Inference Serving on Relational Data
110. Zeyu Li, Chuanfu Xiao, Yang Wang, et al (2025) AnTKV: Anchor Token-Aware Sub-Bit Vector Quantization for KV Cache in Large Language Models. arXivorg. <https://doi.org/10.48550/arXiv.2506.19505>
111. Jiakun Fan, Yanglin Zhang, Xiangchen Li, Dimitrios S. Nikolopoulos (2025) Taming the Memory Footprint Crisis: System Design for Production Diffusion LLM Serving. arXivorg. <https://doi.org/10.48550/arXiv.2512.17077>
112. Lei Gao, Chaoyi Jiang, Hossein Entezari Zarch, et al (2025) DuetServe: Harmonizing Prefill and Decode for LLM Serving via Adaptive GPU Multiplexing. arXivorg. <https://doi.org/10.48550/arXiv.2511.04791>
113. Zhibin Wang, Zetao Hong, Xue Li, et al (2025) Adaptive Rescheduling in Prefill-Decode Disaggregated LLM Inference. arXivorg. <https://doi.org/10.48550/arXiv.2510.13668>
114. Yuning Zhang, Grant Pinkert, Nan Yang, et al (2025) DuoServe-MoE: Dual-Phase Expert Prefetch and Cache Scheduling for Efficient MoE LLM Inference. arXivorg. <https://doi.org/10.48550/arXiv.2509.07379>
115. Yuhang Li, Rong Gu, Chengying Huan, et al (2025) HotPrefix: Hotness-Aware KV Cache Scheduling for Efficient Prefix Sharing in LLM Inference Systems. Proc ACM Manag Data. <https://doi.org/10.1145/3749168>
116. Jinghan Yao, Nawras Alnaasan, Tianrun Chen, et al (2023) Flover: A Temporal Fusion Framework for Efficient Autoregressive Model Parallel Inference. International Conference on High Performance Computing. <https://doi.org/10.1109/HiPC58850.2023.00026>
117. William Meng, Benjamin Lee, Hong Wang University of Pennsylvania, Intel (2025) Understanding Bottlenecks for Efficiently Serving LLM Inference With KV Offloading
118. A. Saxena, Po-An Tsai, Hritvik Taneja, et al (2025) Utility-Driven Speculative Decoding for Mixture-of-Experts. arXivorg. <https://doi.org/10.48550/arXiv.2506.20675>
119. Zeyu Zhang, Haiying Shen (2024) PecSched: Preemptive and Efficient Cluster Scheduling for LLM Inference
120. Ruihao Li, Shagnik Pal, Vineeth Narayan Pullu, et al (2025) MIRAGE: KV Cache Optimization through Parameter Remapping for Multi-tenant LLM Serving. arXivorg. <https://doi.org/10.48550/arXiv.2507.11507>
121. Selin Yildirim, Deming Chen (2025) SpecMemo: Speculative Decoding is in Your Pocket. arXivorg. <https://doi.org/10.48550/arXiv.2506.01986>
122. Limin Cheng, Hang Qin, Shouxu Kuang, et al (2025) InterLayer: Efficient Inference with Interleaved Scheduling and Layer-Specific Optimization. IEEE International Conference on Multimedia and Expo. <https://doi.org/10.1109/ICME59968.2025.11209145>

123. Wei Gao, Chaowei Kong, Zhenqi Gao, et al (2025) KV Cache Group Quantization Based on vLLM Inference Engine. 2025 5th International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA). <https://doi.org/10.1109/CAIBDA65784.2025.11183354>
124. Ziyi Xu, Zhiqiang Xie, Swapnil Gandhi, Christos Kozyrakis (2025) FailSafe: High-performance Resilient Serving. arXivorg. <https://doi.org/10.48550/arXiv.2511.14116>
125. Jiazhi Jiang, Yao Chen, Zining Zhang, et al (2026) Efficient KV Cache Spillover Management on Memory-Constrained GPU for LLM Inference. IEEE Transactions on Parallel and Distributed Systems. <https://doi.org/10.1109/TPDS.2025.3626974>
126. Mingyu Yang, Jae-Young Choi, Kihyo Moon, et al (2025) DSDE: Dynamic Speculative Decoding with KLD Stability for Real-World Serving. arXivorg. <https://doi.org/10.48550/arXiv.2509.01083>
127. Mona Moghadampanah, Adib Rezaei Shahmirzadi, Farhana Amin, D. S. Nikolopoulos (2025) Modality Inflation: Energy Characterization and Optimization Opportunities for MLLM Inference. arXivorg. <https://doi.org/10.48550/arXiv.2512.22695>
128. Songze Liu, Hongkun Du, Shaowen Wang (2025) Adaptive Cache Pollution Control for Large Language Model Inference Workloads Using Temporal CNN-Based Prediction and Priority-Aware Replacement. arXivorg. <https://doi.org/10.48550/arXiv.2512.14151>
129. Songyu Zhang, Aaron Tam, Myungjin Lee, et al (2026) Making MoE-based LLM Inference Resilient with Tarragon
130. Junhan Liao, Minxian Xu, Wanyi Zheng, et al (2025) DOPO: A Dynamic PD-Disaggregation Architecture for Maximizing Goodput in LLM Inference Serving. arXivorg. <https://doi.org/10.48550/arXiv.2511.20982>
131. Liang Zheng, Bowen Shi, Yitao Hu, et al (2026) Mosaic: Unlocking Long-Context Inference for Diffusion LLMs via Global Memory Planning and Dynamic Peak Taming
132. Jianshu She, Zonghang Li, Hongchao Du, et al (2026) LAPS: A Length-Aware-Prefill LLM Serving System
133. Lingfeng Tang, Daoping Zhang, Junjie Chen, et al (2025) MultiPath Transfer Engine: Breaking GPU and Host-Memory Bandwidth Bottlenecks in LLM Services. arXivorg. <https://doi.org/10.48550/arXiv.2512.16056>
134. Dongha Yoon, Younghoon Min, Hoshik Kim, et al (2025) TraCT: Disaggregated LLM Serving with CXL Shared Memory KV Cache at Rack-Scale. arXivorg. <https://doi.org/10.48550/arXiv.2512.18194>
135. Haojun Xia, Xiaoxia Wu, Jisen Li, et al (2025) Kitty: Accurate and Efficient 2-bit KV Cache Quantization with Dynamic Channel-wise Precision Boost. arXivorg. <https://doi.org/10.48550/arXiv.2511.18643>