

Krasis

Phase-Separated Inference for Large Mixture-of-Experts Models on Memory-Constrained NVIDIA GPUs

Living engineering white paper

Version 0.5 — 19 October 2026

Author: Northloom Ltd

About this living white paper

Northloom Ltd is developing Krasis, an inference engine designed to run large language models on accessible hardware. As our research evolves, this living white paper documents our approach, experimental findings, limitations, and open questions.

To keep this document current without diverting focus from core engineering, we use AI tools to help draft and update the text directly from our logs and experimental data. Every update is reviewed and curated by our team, and Northloom Ltd takes full responsibility for the published material.

Throughout the paper, we clearly distinguish ongoing research from validated results. All findings are specific to the models, configurations, and conditions tested, and earlier conclusions may be revised as new evidence emerges.

Abstract

Krasis is a Rust/CUDA inference engine for large mixture-of-experts models whose complete expert weights exceed available GPU memory. It separates fully GPU-executed prompt processing from residency-aware token generation. Bounded layer-group streaming amortizes weight movement over prompt batches; during decode, HCS keeps a measured expert subset in VRAM and transfers selected cold experts from host memory. Runtime calibration measures actual short- and long-prompt memory requirements rather than assuming a hardware-specific budget.

The architecture and dated benchmark record demonstrate useful inference for models much larger than one GPU's VRAM. Those measurements are configuration- and workload-specific, and are not a new benchmark of the pending October conversation-loading repair. The original architecture, optimisation inventory and negative results are retained in the technical chapters.

This edition adds a central qualification question: whether the constrained runtime preserves useful model capability. Agentic Development Qualification (ADQ) separates productive development tasks, numerical fidelity, determinism, cross-runtime behaviour and operational safety. In an archived GLM-5.3-Flash campaign, simple behavioural comparisons passed through 500,000 input tokens, but only two of four development tasks passed. A later DeepSeek precision sweep completed all four tasks for every treatment, including plain INT4, while every treatment independently failed its VRAM reserve requirement. More precision was neither a universal quality improvement nor a free performance improvement.

October research further shows why state reuse needs its own correctness proof: active and RAM-restored continuations can agree with one another while differing from fresh full-history execution. The current conversation-loading repair and subsequent GLM ADQ remain unqualified. These open findings are reported beside successful component repairs, rather than concealed by them.

Document status and evidence policy

This is an engineering white paper based on the Krasis source tree, complete benchmark logs, the internal changelog, design records, commit history, and stored validation artifacts. Chapters 1–18 and Appendices A–B preserve the August 2026 engineering baseline, with explicit corrections where subsequent evidence changed the claim. Their references to “current” describe that dated baseline, not qualification of the October candidate. The research update incorporates reviewed records through 19 October 2026 and takes precedence where later evidence limits an earlier conclusion. This is not peer-reviewed research, and it is not an independent audit of Krasis.

The linked benchmark and quality ledgers are maintained separately from this versioned paper. Earlier benchmark and quality values came from [STATS-BENCHMARKS.md](#), [STATS-QUALITY.md](#), and the raw-log index in [benchmarks/BENCHMARKS.md](#). The complete claim ledgers used to prepare this paper are the successful-decision inventory (internal record) and the negative-result inventory (internal record).

The paper uses six evidence descriptions:

- **Current benchmark evidence** is a timing-off run through a current built Krasis command with a stored raw log.
- **Controlled development evidence** is a measured A/B on a narrower model, machine, or workload.
- **Functional evidence** establishes correctness, portability, memory behaviour, or an execution contract but does not establish a speedup.
- **Measured planning evidence** combines real component measurements, loaded tensor sizes, route traces, and capacity arithmetic to reject or bound an idea before a production runtime is built. It is not reported as a benchmark.
- **Historical evidence** was valid for an earlier architecture or quantization surface but should not be presented as current production performance.

- **Community-reported evidence** is a result published by someone outside the Krasis development programme. It demonstrates independent use on additional hardware, but is not promoted to controlled benchmark evidence unless the exact release, config, prompts, raw log, correctness witness, and VRAM safety contract are available.

Where the available evidence is weaker, the paper says so. In particular, Krasis BF16 is not described as an independent correctness oracle, deprecated AWQ and FP8 results are not presented as current quality evidence, and the default-off adaptive cold-mass mode is kept separate from exact Krasis.

Contents

- [Executive summary](#)
- [Research update – 19 October 2026](#)
- [1. The problem Krasis is designed to solve](#)
- [2. Design goals and contributions](#)
- [3. System overview](#)
- [4. Phase one: fully GPU-executed prefill](#)
- [5. Phase two: residency-aware Rust/CUDA decode](#)
- [6. HCS: exact expert residency under a changing memory budget](#)
- [7. Optional approximate decode: adaptive cold-mass pruning](#)
- [8. Runtime VRAM calibration](#)
- [9. Multi-GPU execution](#)
- [10. Runtime, memory, and kernel optimisations](#)
- [11. Measurement, calibration, and validation methodology](#)
- [12. Experimental evaluation – August baseline](#)
- [13. Case studies](#)
- [14. Negative results and what they taught us](#)
- [15. Related system position](#)
- [16. Limitations and threats to validity](#)
- [17. Future work](#)
- [18. Conclusion](#)
- [Appendix A. Accepted architectural decisions and optimisations](#)
- [Appendix B. Negative results](#)
- [Appendix C. Reproduction and source guide](#)
- [Appendix D. Change history](#)

Executive summary

Krasis was built for a case that ordinary GPU layer offload handles poorly: a large MoE model has far more possible expert weights than available VRAM, but a request still needs fast processing of a long prompt and useful token generation afterward.

The system's central decision is to stop treating prompt processing and token generation as one workload. During prefill, Krasis streams bounded groups of INT4 expert weights through the GPU and reuses each group across many prompt tokens. During decode, it keeps a measured subset of exact experts resident in VRAM and moves only selected cold experts on demand. The prefill phase therefore does not inherit a CPU-oriented decode compromise, while the decode phase does not require the complete model to fit.

HCS turns spare VRAM into an expert cache. An exact-configuration heatmap provides the starting prior; bounded within-request promotion adapts it to the current sequence. A resident tier remains available for decode and a reclaimable tier can be borrowed by prefill. The complete expert collection remains in source-backed or mirror-backed host memory.

On heterogeneous two-GPU systems, the same principle has two measured forms. A speed-aware contiguous split assigns fewer serialized layers to a slower auxiliary card; on QCN, [46, 2] improved decode 11.82% over the capacity split. For GLM, the A4500 was more useful as a disjoint exact expert-service tier than as a serial layer owner. A persistent mailbox served selected experts and returned deadline misses to the canonical cold path without changing output.

Adaptive cold-mass pruning is a separate, approximate extension of this exact cache. It observes a useful asymmetry: a cold expert costs its full transfer even when its router weight is very small, while a resident expert has no demand-transfer cost. The policy therefore considers only low-ranked, demand-cold routes and limits their combined omitted mass for each token and layer. The Ornith 75/8 A/B improved internal decode by 9–10% across 50, 100, and 250 generated tokens. Exact mode remains the default because the available coherence tests do not replace an independent quality witness.

The amount of HCS is measured rather than fixed. Krasis loads the real model and backend, runs short and long calibration workloads without reclaimable HCS, records driver-visible low-water VRAM, and derives a prompt-length decode budget. Live chunked allocation guards the measured plan. Current constrained RTX 5090 runs finish with approximately 856–952 MB minimum free VRAM against a 600 MB safety floor, while fully resident models correctly leave much more headroom.

The token-critical path is native. Rust owns decode, sampling, detokenization, tool-call handling, JSON/SSE streaming, and CUDA dispatch. Prefill computation uses native GPU kernels and bounded staging. Marlin-packed INT4 experts, batched expert GEMVs, CUDA graph segments, pinned demand DMA, native attention sidecars, HQQ attention profiles, and model-geometry-specific kernels reduce the remaining execution cost.

The stateful request path is designed to preserve the selected model state. October counterexamples show that end-to-end equivalence is not yet qualified; see the research update. A versioned compatibility signature binds rendered tokens, templates, model state, GPU ownership, KV layout, and architecture-specific recurrent/compressor state. Committed boundaries can remain active on GPU or be restored from pageable RAM. Save publication, LRU eviction, and cancellation are transactional. Structured tool use is likewise template-derived: Rust implements the native grammars declared by DeepSeek, Qwen, GLM, Gemma, and MiniMax rather than using a permissive universal fallback.

The current benchmark set demonstrates the architecture across several regimes. On one RTX 5090, Qwen3.6-35B reaches 10,670.1 prefill and 117.20 decode tokens per second with all experts resident. Qwen3.5-122B reaches 3,692.5 and 31.80 with 36.0% HCS. Qwen3-235B reaches 1,385.3 and 5.64 with 17.3% HCS. On one RTX PRO 6000, Step-3.7 reaches 5,261.0 and 55.40 with 91.9% HCS, while Ornith-397B reaches 2,354.5 and 23.58 with 42.8% HCS.

The Ornith result provides the clearest external system comparison. The best fitting llama.cpp Q4_K_M configuration found by an OOM-gated sweep used approximately 92 GB of the same RTX PRO 6000 and kept 38 MoE layers on CPU. It reached 185.01 tokens per second at pp8192 and 10.71 at tg128. Krasis's reported values come from a different format and fixed benchmark suite, so the comparison does not isolate kernels or provide a matched-prompt speedup. It does show that the recorded Krasis system handles this memory regime substantially better.

The paper's final contribution is the experimental record. Eighty-six hypotheses are catalogued as rejected, reverted, blocked, or superseded. Speculative expert prefetch could predict routes but did not have a useful copy window. Nested low-bit cold compression reduced isolated bytes but failed once HCS promotion was included. CPU cold-expert execution avoided weight DMA and lost on integrated compute and correctness. Many fusions removed launches and reduced occupancy or left the end-to-end result unchanged. The newest experiments add another recurring lesson: optimisations that remove the same cold route do not compose additively. Peer service, compression, pruning, HCS, and CPU tails must be evaluated in the order they consume the cold population.

This evidence leads to a broader conclusion. For large MoE inference, the important unit of optimization is not a kernel in isolation. It is the complete relationship between phase, residency, data movement, synchronization, allocator state, model quality, and the measurement boundary.

Research update – 19 October 2026

This edition extends the August architecture and performance record with September–October research on retained model capability, quantisation and conversation restoration. The architecture chapters and their experimental inventories retain their dated evidence; they are not a new benchmark of the October repair candidate. This update supersedes earlier broad statements that conversation reuse was established as exact across execution paths.

Correctness is a primary inference requirement

A model can load, produce fluent answers, retrieve information from a very long prompt and sustain good token throughput while failing to complete useful work. An agent can describe a correct change, announce that it will edit a file, and then stop without performing the edit. Other failures include repeatedly using tools without converging or producing a patch that passes visible tests but violates the task's hidden requirements.

We have observed these distinct behaviours during Krasis qualification. The plan-without-action example occurred in a controlled DeepSeek development task; separate exact-500,000-token tasks exposed semantic errors despite successful retrieval and normal request completion. These observations should not be collapsed into a claim that every failure occurred at 500K or had one cause.

Small arithmetic differences matter because inference is a sequence of state-dependent decisions. A changed reduction order can change a logit; a changed selected token can change later reasoning and tool actions. Weight quantisation can also alter those decisions. Neither mechanism establishes the cause of a particular task failure without controlled evidence. Model limits, sampling settings, templates, harness defects and runtime defects must be investigated separately. A BF16 reference can itself fail a task.

For Krasis, useful performance therefore includes retained task capability, repeatability, state correctness and safe operation alongside throughput. The central research question is how much useful capability a specific constrained runtime preserves, at a specified context length and memory cost.

Why we created ADQ

Agentic Development Qualification (ADQ) is our configuration-specific method for asking whether a model actually performs development work. It complements perplexity, first-token comparisons and synthetic retrieval; it does not replace them with one opaque score.

An ADQ identity binds the checkpoint, tokenizer and template, runtime source and binary, expert and attention precision, KV format, heatmap, hardware, sampling contract, context tier, tool harness and task fixtures. A result for one identity does not qualify another quantisation, cache mode or runtime revision. Experimental INT8 and mixed-precision controls in this paper are research treatments, not a change to the production focus on INT4 experts.

ADQ uses four evidence layers:

1. **Numerical fidelity:** controlled comparisons with an independent source-precision llama-witness implementation where available. The standard detailed numerical track extends through 131,072 exact input tokens. Higher-context numerical evidence is reported only when actually obtained.
2. **Cross-runtime behaviour:** frozen cases at increasing exact token counts, including comparisons with hosted implementations. Hosted provider success is a behavioural control; an undisclosed provider stack is not asserted to

be a verified BF16 numerical oracle.

- 3. **Productive work:** frozen repository tasks, real tool calls, completed changes and objective public and hidden tests. A plausible plan or unverified patch cannot pass. These are bounded development tasks, not a claim to represent all software engineering.
- 4. **Operational testing:** context admission, conversation switching and restoration, isolation, cancellation, recovery, determinism and CUDA/HCS health, with request-level low-water VRAM measurements.

Before scoring productive work, fresh-process, cold-first, warmed and replay controls must establish deterministic execution under the declared greedy contract. The actual forwarded sampling parameters are retained; a client label saying “greedy” is insufficient.

The tier names are literal: ADQ-128K exercises 131,072 input tokens, ADQ-256K exercises 262,144 and ADQ-500K exercises 500,000. Rendered templates and tool schemas add tokens, and the server must also reserve continuation capacity. These lengths are measured with the checkpoint tokenizer rather than estimated from characters. A configured one-million-token cache is not evidence of a completed 500K task.

Every report separates productive-task qualification, runtime determinism, numerical fidelity, cross-runtime behavioural coverage, VRAM safety and runtime integrity. No aggregate pass hides an independently failed row. A task pass with unsafe memory use remains a task pass and a safety failure; it does not justify deployment. Missing numerical coverage is explicitly incomplete or untested, rather than silently extrapolated.

Finding 1: simple long-context success can conceal task failure

The September 17 GLM-5.3-Flash revision-7 campaign used routed Marlin INT4 g128 with amax scales, HQQ6 attention, k6v6 KV and INT8 shared/dense/head components, on one RTX PRO 6000, with measured HCS, a 600 MiB reserve, temperature zero and thinking disabled. The checkpoint revision was 3f1971b7b5f7a528c9c4ef6212c8785298a8c24a; configured capacity was 1,048,576 tokens. Its runtime source base was 27054ae4ae388fc056b28db11272bf181323723b, with a separately frozen working-source manifest. This is the archived revision-7 identity, not a verdict on later code.

Independent verdict	Archived result
Productive tasks	FAILED: 2/4 ; Config and Scheduler passed; Ledger and History failed
Runtime determinism	PASS : 20/20 token- and numerically-identical rows across five fresh lifecycles
Numerical fidelity	FAILED_INTEGRITY under the frozen decode-fidelity contract; complete through 131,072 tokens
Cross-runtime behaviour	COMPLETE : seven local cases through exact 500K; 14 aligned provider comparisons
VRAM safety	PASS : global measured request floor 601 MiB against a 600 MiB requirement
Runtime integrity	PASS under that campaign's operational checks

Ledger repeated an incorrect probe hundreds of times without producing the required implementation. History received an exact 500,000-token history, with a first rendered request of 503,186 tokens. Its patch passed public tests but failed two hidden assertions concerning canonical defaults and an error contract. Both hosted references, Together and DeepInfra, completed all four tasks once. This is evidence of a deficit for the tested Krasis identity, not proof that numerical drift was the sole cause.

The independent numerical comparison selected a different first decode token at 3K, 16K, 32K and 64K. Results were finite and retained witness-selected tokens in the top ten; the verdict names a failed fidelity threshold rather than claiming memory corruption. Witness failed to provide 262K numerics because of its own CUDA softmax launch limitation; numerical comparison at 500K was untested. Behavioural coverage at 500K does not fill that numerical gap.

These results show why retrieval, operational robustness and deterministic execution cannot substitute for productive work. They also show why a later, stronger cache counterexample must not be erased by an earlier operational pass.

Finding 2: arithmetic consistency required a real runtime repair

A September DeepSeek INT8 diagnostic held the rendered input, router inputs, expert IDs, route weights, expert payloads and transfer completion constant. Cold-first and graph-warmed decode nevertheless differed after the first routed-MoE layer. The cold path used sequential fused per-expert accumulation; graph replay used a batched expert stack and fixed reduction.

The repair made uncaptured execution use the canonical graph-equivalent path. Post-fix controls matched all 43 hidden/hyper-connection layer boundaries and 129,280 logits, and five fresh lifecycles reproduced their 128-token results. This established a specific arithmetic repair. It did not establish that all task failures had been fixed: three subsequent sampled Ledger attempts yielded FAIL/PASS/FAIL. The sampling contract itself then required correction and separate qualification.

In the later explicitly greedy paired campaign, INT4 completed 3/4 tasks and INT8 4/4. The failing INT4 Config trace described the required change but stopped before editing. That bounded observation supports a precision-related capability effect, while neither a single task matrix nor the earlier sampled runs estimate repeated-attempt reliability.

Finding 3: more precision was not uniformly more useful

A subsequent frozen DeepSeek-V4-Flash-Vision-Exp six-treatment sweep tested plain INT4, four mixed INT4/INT8 payload budgets and full routed INT8. All six completed the same four tasks once, including exact-500K History, with public and hidden tests passing. All passed determinism; all separately failed the unchanged 600 MiB reserve during pre-score checks. Plain and mixed INT4 retained localized numerical warnings; full INT8 was within the inherited envelope, whose reference provenance limitation remains part of the result.

Routed treatment	Archived internal decode, 250-output row
INT4	32.00 tok/s
INT4 +5% payload budget	29.38 tok/s
INT4 +10%	27.22 tok/s
INT4 +15%	25.16 tok/s
INT4 +20%	23.07 tok/s
INT8	11.42 tok/s

These are timing-disabled engine measurements from the sweep's separately recorded performance runs, not HTTP rates or task completion rates. Detailed HTTP, memory and prefill records remain separate. Increasing representation size reduced useful residency and increased cold-transfer traffic in the separate instrumented diagnostics. Mixed precision did not improve numerical agreement monotonically. The sweep supports neither a universal INT4 quality claim nor a mixed-precision production recommendation.

The earlier paired 3/4-versus-4/4 result and the later all-4/4 sweep are both retained. Their identities and experimental contracts differ; choosing only one would conceal the limits of once-only task evidence.

Finding 4: scale selection mattered without adding expert bytes

An October 5 GLM scales-only ablation compared original amax INT4, searched INT4 scales, and searched scales with a small INT8 reserve. It scored 65,448 held-out targets across eight independent approximately 8K sequences, not one 65K context. Two fresh matched cohorts reproduced all 24 response pairs exactly.

Treatment	Pooled negative log-likelihood	Added routed payload
Original INT4	0.72881583	0
Searched scales-only INT4	0.71335846	0
Searched scales plus minimum INT8 reserve	0.71350112	1,799,356,416 bytes

Scales-only reduced pooled NLL by 2.1209% against the original. Adding the reserve slightly worsened pooled NLL relative to scales-only; it helped four documents and hurt four. Seven documents improved over the original in both treatments; one worsened. This isolates scale selection on this corpus. It does not prove long-context task improvement, statistical generality or ADQ qualification. Productive follow-up remains blocked by the separate conversation-equivalence prerequisite.

Conversation caching carries a correctness obligation

Fresh full-history processing, continuation of active GPU state and restoration from pageable RAM are different execution paths. Matching active and restored outputs proves something useful, but both can disagree with full recomputation. A successful copy, cache-hit counter or readable answer does not establish preserved quality.

The August QCN cache benchmark remains a valid historical observation for its fixed workload: six real RAM restores, matching text and counts, 1.132x faster continuation and 1.067x faster end to end. It does not qualify every state family, context length or later runtime. The October investigation retained a GLM counterexample with identical 10,564 input tokens and a different sixth generated token between full and cached execution. Later QCN candidates also exhibited stable full-history controls and active/RAM agreement while differing between full and resumed paths, including complete F32 differences before visible text necessarily diverged.

As of 19 October, the conversation-loading repair is **not qualified**. Component repairs and isolated snapshot, coefficient, ownership and stream-ordering proofs have not closed the end-to-end counterexample. The current successor still requires a diagnostics-off ordinary verdict, complete token/F32 comparisons, cancellation/error recovery and the retained GLM regression. The interrupted native Store sanitizer runs are invalidated evidence, not passes. Both-GPU failure/recovery and zero-leak qualification remains open; component sanitizer coverage is not full-model memory-safety coverage.

The retained engineering acceptance requires genuine reuse and full/repeat, active and RAM comparisons with strict complete numerical checks for the named counterexample. Where changed batch geometry may legitimately change arithmetic, that is characterized explicitly; it cannot be used to waive this gate. A separate output-invariant observer may locate the first differing operation, using matching actual inputs, state, coefficients and geometry before any arithmetic change is accepted. This is stronger than the August envelope-based method described historically in the earlier edition.

Memory and measurement lessons

Memory must be measured through preparation, startup, calibration, prefill, decode and conversation restoration. A report-time resident-set reading is not a startup peak or a system-wide RAM requirement. In one October GLM full-INT8 experiment, report-time RSS was 304.8 GiB while a separate startup sample reached about 515 GiB. Neither measures peak memory for the entire productive workload. These values describe that treatment and host; they are not sizing constants.

Runtime HCS admission remains based on actual short/long calibration and live prompt token counts, with the default 600 MiB margin. Failed safety measurements must lead to a corrected memory model, not a smaller reserve or a fixed budget. A large unneeded gap above the reserve also merits investigation when experts remain cold; spare VRAM has a measurable residency value.

For stalls, GPU utilisation proves activity, not completed work. Submitted layers and completed layers must be distinguished through completion boundaries or nonblocking event observation. Diagnostic instrumentation is enabled to explain behaviour and disabled for ordinary correctness and speed measurements. The October privileged debugger attachment disturbed the sanitizer runs; those runs cannot be counted as clean qualification evidence.

What remains open

The immediate priority is verified conversation loading followed by GLM-5.3-Flash ADQ on the repaired, frozen identity. The earlier GLM 2/4 failure stays visible. A later configuration cannot inherit qualification from an earlier partial success, a scales-only NLL gain or a component proof. Clean speed regression measurements follow correctness; no new speedup is claimed for the pending repair.

Public evidence for the older architecture and benchmark results is linked in Appendix C. September–October ADQ and debugging findings above are Northloom's summaries of retained internal experiments; their complete manifests and raw artifacts are not publicly downloadable with this edition. This limits external reproduction and independent audit. The edition distinguishes those company research findings from linked public evidence and makes no peer-review claim.

1. The problem Krasis is designed to solve

1.1 A large MoE can be small and large at the same time

A dense model uses nearly all of its feed-forward weights for every token. A mixture-of-experts model contains many alternative feed-forward networks and a router chooses a small number of them at each routed layer. This means the total parameter count can be very large while the active parameter count for one token is much smaller.

This is attractive computationally, but it does not remove the storage problem. The router may select any expert. Unless the runtime knows the future route, the complete expert collection must exist somewhere accessible. For a model with many layers and hundreds of experts per layer, exact expert weights can exceed the VRAM of a consumer or workstation GPU many times over.

The central difficulty is therefore not simply how many floating-point or integer operations are required. It is whether the selected weights are already where the GPU needs them.

For a routed layer, a simple approximation of cold traffic is:

```
cold bytes for a token
= sum over layers and selected experts
  (expert is not resident) * expert bytes
```

If h is the hit rate of the resident expert cache, k experts are selected in each of L similar layers, and each expert occupies B bytes, the first-order traffic estimate is:

$$D_{\text{cold}} \approx L * k * (1 - h) * B$$

The transfer-time floor is then at least $D_{\text{cold}} / \text{effective_PCIe_bandwidth}$, before routing, attention, expert computation, synchronization, or serving overhead is included. This explains why another gigabyte of useful expert residency can sometimes matter more than a faster arithmetic kernel.

1.2 Prefill and decode are not the same workload

An inference request has two main phases.

During **prefill**, the model processes the prompt. A prompt may contain ten thousand or more tokens, and the same layer weights are applied to a large token batch. This provides substantial matrix parallelism. A weight transfer can also be amortized across many tokens.

During **decode**, the model produces one new token at a time. The batch is usually one. Kernel-launch overhead, synchronization, routing decisions, expert residency, and cold transfers become much more visible. The runtime cannot amortize a cold expert transfer across the whole prompt because the route is only known as the token reaches that layer.

If one execution policy is imposed on both phases, one of them is likely to be poorly served. A CPU-oriented policy may avoid impossible GPU residency during decode, but it also gives up the large GPU throughput available during prefill. A policy that assumes all weights remain on GPU can make prefill fast, but it cannot run a model whose complete weight set exceeds VRAM.

Krasis begins with the opposite assumption: the best execution strategy for prefill is not expected to be the best execution strategy for decode.

1.3 Long prompts make prefill a user-visible problem

For a short chat prompt, a slow prefill may be tolerable. For an IDE, coding agent, or tool-using session, requests frequently include repository context, tool descriptions, conversation history, and retrieved files. A request can start with thousands or tens of thousands of input tokens before the first new token is generated.

The recorded llama.cpp pp8192 row required roughly 44 seconds of prompt processing. Krasis's current standard benchmark reports a best prefill rate over its own fixed prompt-length set rather than a matched pp8192 row, so a strict same-prompt latency comparison remains follow-up work. The recorded rates are nevertheless far enough apart to make long-prompt processing a central part of the system evaluation rather than a secondary metric.

For this reason, Krasis does not regard prefill as secondary to decode. The two rates are always reported separately, and the complete HTTP result is reported separately again.

2. Design goals and contributions

Krasis is built around the following goals.

2.1 Process every supported prompt on the GPU

The full model need not be resident at once, but prompt computation remains a GPU workload. Weights may be streamed through bounded buffers; computation is not redirected to a CPU prefill fallback.

2.2 Make decode exact by default

The production decode path uses exact cached INT4 expert weights. A cache miss causes a demand transfer of the real expert. Approximation is not silently introduced to improve a benchmark.

2.3 Use VRAM as a dynamic expert cache

Available VRAM after permanent state and measured transient requirements is used for expert residency. This gives every additional unit of VRAM a direct role in reducing cold traffic, without requiring the entire model to fit.

2.4 Keep the measured hot path in Rust and CUDA

Per-token decode, expert dispatch, sampling, detokenization, HTTP formatting, and streaming do not return through Python. Prefill compute is likewise executed through native GPU paths; Python remains appropriate for one-time setup and model-loading orchestration, not token-critical work.

2.5 Measure the loaded system instead of assuming its costs

Memory requirements, transient workspaces, allocator state, prompt length, attention geometry, expert size, and GPU capacity all vary. Krasis measures the actual configuration and uses the result in runtime decisions.

2.6 Prefer end-to-end evidence over local improvements

A smaller transfer, faster microkernel, higher prediction accuracy, or lower instrumented substage is not accepted by itself. It must improve a clean, timing-off execution path and retain the relevant correctness and quality behaviour.

2.7 Keep approximation explicit

Exact execution is the default contract. An approximate optimisation must be selected deliberately, report what it omitted, reject incompatible execution paths, and pass a model-specific quality gate before it can be described as accepted for that model.

“Exact” here means that caching does not intentionally omit or alter selected quantised expert payloads. It does not mean equivalence to BF16 arithmetic or prove full-history/cached execution identity. Those are separate measured gates.

These goals lead to five main contributions:

1. A phase-separated architecture combining fully GPU-executed streamed prefill with residency-aware hybrid decode.
2. A native runtime, memory, and kernel stack that makes this architecture practical across several MoE and attention families.
3. An HCS-aware approximate extension that can exchange bounded low-rank routed mass for less serialized cold traffic without changing exact default mode.
4. A measurement, calibration, and validation process that treats truthful evidence as part of the system rather than as an afterthought.
5. A transactional stateful serving layer designed to reuse conversation prefixes, with end-to-end equivalence under active investigation, and model-native tool grammars without placing Python in the token path or inventing silent compatibility fallbacks.

3. System overview

The complete request can be summarized as follows:

```

request
|
v
template + exact tokenization
|
v
longest exact compatible prefix lookup
|               |
| miss           | active GPU / pageable-RAM hit
v               v
evict reclaimable expert residency for the measured prompt requirement
|
v
GPU PREFILL
  stream bounded layer groups -> process uncached suffix -> build KV/state
  |
  v
  reload only the expert residency allowed by measured decode requirements
  |
  v
  RUST/CUDA DECODE
    route -> resident hit or demand DMA -> expert compute -> attention -> sample
    |
    v
    buffered SSE / native tool-call response
    |
    v
    transactional state snapshot commit on successful completion

```

The architecture has four important forms of state.

Permanent GPU state contains the weights and buffers that the selected backend must keep resident, including embeddings, norms, shared or dense components where applicable, attention representations, KV/state allocations, and native runtime metadata.

Prefill staging state contains bounded expert layer groups and the scratch required for the current prompt chunk. This state is reused as the prompt moves through the model.

HCS expert residency contains the exact routed experts selected to remain in VRAM during decode. Part of this residency may be reclaimed temporarily for prefill.

Host backing contains the complete cached expert representation and other large model data. Depending on available system RAM, HCS can use source-backed or mirror-backed storage without changing expert precision.

Conversation state contains exact stable prefix checkpoints. One compatible conversation may remain active in device state; inactive snapshots use pageable RAM and measured pinned staging for transfer. This state is separate from model weights and HCS. A snapshot is published only after response completion, and a cache miss always returns to ordinary prefill rather than a degraded approximation.

The important point is that these states do not all need their maximum capacity at the same time. Prefill and decode borrow the GPU from each other under a measured contract.

4. Phase one: fully GPU-executed prefill

4.1 Layer-group streaming

Krasis stores routed experts in a cached, Marlin-packed INT4 representation. During prefill it loads a bounded group of layers into GPU staging buffers, processes all prompt chunks through that group, releases the group, and reuses the same VRAM for the next group.

If there are G layer groups and C prompt chunks, the loop is organized with the layer group outside the chunk loop:

```
for each layer group:
    transfer this group's weights to GPU
    for each prompt chunk:
        execute this group for the chunk
    reuse the staging allocation for the next group
```

This order matters. The group is transferred once and reused across the prompt, rather than being transferred again for every token or every small chunk. The complete model does not need to fit in VRAM, but every prompt token still receives GPU computation.

4.2 Prompt chunking

The prompt is divided into chunks because attention state, activations, expert outputs, and backend workspaces grow with prompt length. The safe chunk is not a fixed value. It is selected from the loaded model geometry, measured free VRAM, and calibrated short/long prompt behaviour.

If an allocation nevertheless fails, the chunk can be reduced and the request retried from a clean state down to an explicit floor. This is an error-recovery mechanism around a measured plan, not a replacement for measurement.

4.3 Double-buffered staging

Two staging regions allow preparation or transfer for upcoming work to overlap with the current group where the hardware and dependency graph permit it. Explicit CUDA streams and events preserve ordering, while allocator lifetime is recorded so a buffer cannot be reused while a kernel still consumes it.

The accepted design uses two buffers. A three-buffer experiment increased VRAM and allocator pressure without improving throughput and was reverted. This is a representative Krasis result: deeper pipelining is not assumed to be better without an execution window that can use it.

4.4 Predictable next-layer prefetch

Some architectures provide a real prefill overlap opportunity. In Nemotron Super, the next dense MoE layer is known in advance and preceding Mamba2 work creates a compute interval in which its weights can be staged. This improved prefill from 1,197.8 to 1,370.3 tokens per second, while exposed MoE DMA wait on a representative 5K prompt fell from 1,099.9 to 28.6 milliseconds.

This is different from speculative decode prefetch. Prefill knows which layer comes next and can reuse its weights across many tokens. Decode must predict the next token's routes and often has only one serialized copy engine. The former produced a measured gain; the latter repeatedly did not.

4.5 Attention and recurrent state

Krasis selects the appropriate native attention or recurrent implementation for each layer geometry. Full and sliding attention are selected from model config, not a model-name index. Gated DeltaNet and Mamba2 models use their own guarded native paths.

The benefit can be large. Enabling the correct sliding FA2 path for Step improved long prefill from 1,618.2 to 2,286.5 tokens per second. Guarded Gemma4 head-dimension-512 paths raised HQQ4/k6v6 prefill from 2,268.3 to 3,750.8 tokens per second, and a later wide path raised k4v4 from 4,176.7 to 5,613.0 tokens per second. These changes passed the associated network and model guard tests.

The prefill design is described in more detail in [feature-prefill-pipeline.md](#) (internal record); current performance evidence is in the [benchmark index](#).

5. Phase two: residency-aware Rust/CUDA decode

5.1 Why decode is a residency problem

Once prefill has produced the first token and populated the KV or recurrent state, the computation changes shape. Each new token passes through the model one layer at a time. At every routed layer, the router selects a small top-k set of experts.

If an expert is already in VRAM, the runtime can execute it immediately. If it is cold, its packed weights and scales must first cross PCIe. On a sufficiently large model, this transfer and the synchronization around it dominate arithmetic that is otherwise quite fast.

Krasis therefore does not attempt to make the CPU the normal expert compute device. The CPU and system RAM hold the complete backing, but the selected experts execute on the GPU. Available VRAM is used to prevent as many cold transfers as possible.

5.2 A native token loop

The decode loop is owned by Rust and CUDA. A token passes through embedding, normalization, attention or recurrent layers, routing, expert execution, final normalization, the language-model head, sampling, detokenization, and streaming without a Python callback.

This boundary was not chosen for language preference. An earlier design called back into Python for each token. It measured 6.5 tokens per second while the native Rust MoE/HCS component measured about 40 tokens per second, and it also selected an unsuitable M=1 execution path. Moving the complete loop into Rust removed repeated GIL and FFI crossings and made stream, event, and pointer lifetime explicit.

Python remains useful for setup, configuration, and one-time model loading. It is not placed around the measured token loop. This is especially important when multiple GPUs are intended to work concurrently, because serial host dispatch can make nominally parallel GPU work execute one device at a time.

5.3 Per-token routed expert execution

At each MoE layer the runtime performs the following work:

```
route hidden state
-> select top-k expert IDs and weights
-> look up exact expert pointers in HCS
-> enqueue cold expert DMA when a pointer is absent
-> execute batched INT4 expert projections
-> apply activation/gating
-> reduce weighted expert outputs
```

The hot lookup is a flat array indexed by layer and expert. Resident entries contain GPU pointers; an empty entry means the expert must be transferred. A device-side pointer table allows captured CUDA graph segments to consume changing expert addresses without rebuilding the graph.

Experts are processed in batches rather than with one launch for every selected projection. Early fusion work reduced a QCN MoE layer from 30 launches to four and improved adapted decode from roughly 50–51 to 60–62 tokens per second. The current path uses guarded kernels for the actual activation and top-k geometry; for example, accepted Nemotron Nano sigmoid top-k and ReLU2 work raised its decode path to 119.2 tokens per second at that development gate.

5.4 CUDA graph segmentation

Capturing an entire request as one static graph is not possible because routes, expert addresses, prompt length, and some recurrent state change. Krasis instead captures stable segments.

Fixed upload buffers and pointer indirection supply the changing addresses. The captured segment retains the stable attention, routing, and elementwise launch sequence. This removes repeated launch setup while preserving dynamic routes.

The correct segmentation boundary is model dependent. Adding missing Gemma-specific graph segments raised Gemma4 k6v6 decode from 38.79 to 63.69 tokens per second. Graphing Nemotron Nano's Mamba2 decode reduced a measured timing path from 12.72 to 7.96 milliseconds per token and produced 113.5 tokens per second in the clean timing-off run.

Graph capture is not automatically beneficial. Graphing the DFlash verifier did not improve it because cold expert and verification work dominated. The system therefore graphs stable launch-bound regions, not every region that CUDA permits it to capture.

5.5 Long-context decode attention

A short decode sequence can be handled efficiently by a compact attention kernel. At long context, one block per query head may expose too little parallel work. Krasis uses a tiled, FlashDecoding-style path for long sequences, followed by a log-sum-exp reduction, with the threshold derived from the number of heads, tiles, and available SM parallelism.

On the historical QCN FP8-KV development surface, this changed 18K-context decode from 6.1 to approximately 23 tokens per second and sustained 18.2 tokens per second at 97K context. FP8 is now deprecated, so those absolute rates are not current production claims. The A/B remains evidence for the tiling decision, and tiled dispatch remains part of the current graph and ungraphed kernel stack.

The native decode architecture is documented in feature-decode-loop.md (internal record).

6. HCS: exact expert residency under a changing memory budget

6.1 Resident, reclaimable, and cold experts

HCS divides routed experts into three practical states:

- **resident** experts remain in VRAM across the request;
- **reclaimable** experts are useful for decode but may be evicted to make room for prefill;
- **cold** experts remain in system memory until selected.

The code historically called the two GPU tiers hard and soft. The useful distinction is whether a region survives prefill or can be borrowed by it.

Every cached expert retains its exact packed INT4 representation. HCS therefore changes where an expert is stored, not what it computes.

6.2 Heatmaps as a prior

Before a request begins, Krasis needs an initial resident set. It uses route heatmaps collected with the same model, attention quantization, and execution parameters intended for serving. The most frequently used experts form the initial prior.

Heatmaps are versioned artifacts with compatibility information and checksums. A heatmap made for a different attention profile can change routes and is not silently treated as equivalent. Benchmark prompts are kept separate from heatmap-building prompts so the cache is not optimized for the exact text used to report speed.

The benefit varies by model. A stronger approved Qwen3.5-122B heatmap improved internal decode by about 6.3% over its quick heatmap. On some models the main effect is fewer promotions rather than a headline speed change. Both outcomes are recorded because a heatmap is a prior, not a guarantee.

6.3 Learning during a request

Static global frequency does not capture all local reuse. As a request proceeds, HCS can promote cold experts that are actually used and evict from a bounded recency tail. This keeps an exact global core while adapting to the current prompt and generated sequence.

Promotion must be bounded. An experiment that promoted every cold hit improved Qwen3-235B's apparent hit rate but collapsed decode to 2.7–2.8 tokens per second because cache churn became the workload. Current policy limits promotion and victim movement rather than optimizing hit rate in isolation.

This distinction is important. A cache policy changes which experts will be resident later, so its local decision changes its own future input. A victim rule can appear sensible for one event and still produce a worse residency trajectory over the next hundred tokens.

6.4 Evidence that residency matters

The first HCS A/B cached 41% of QCN experts on an RTX 2000 Ada. MoE time fell from 92.6 to 74.6 milliseconds and MoE throughput improved from 10.8 to 13.4 tokens per second, a 24% gain.

The current benchmark table shows the same mechanism over a much wider range. Qwen3.5-35B holds 100% of routed experts on an RTX 5090 and decodes at 116.43 tokens per second. Qwen3.5-122B holds 36.0% and reaches 31.80 tokens per second. Qwen3-235B holds 17.3% and reaches 5.64 tokens per second. These models differ in active parameters and

architecture, so the rows do not isolate residency as the only variable. They do show why HCS coverage is always reported beside decode speed.

The RTX PRO 6000 results provide a more direct view. QCN rises from 66.6% HCS on the 32 GB 5090 to 100% on the 96 GB card; decode rises more modestly from 87.90 to 91.34 tokens per second because QCN was already highly resident. Step rises from 23.0% to 91.9% HCS and from 23.05 to 55.40 tokens per second. Ornith-397B reaches 42.8% HCS and 23.58 tokens per second on the larger card, compared with 8.9% and 7.89 tokens per second on the 5090. Hardware also changes between these comparisons, but the direction and the measured traffic model are consistent.

6.5 Source-backed and mirror-backed operation

Keeping a second complete host copy can improve reload behaviour when system RAM is plentiful. It can also make a 397B or 550B model impractical on a machine that otherwise has enough RAM for one copy.

Krasis therefore supports source-backed and mirror-backed HCS. The choice is made from the available memory and selected mode; it does not change expert precision. Mirror mode is not assumed to be faster for every architecture. Nemotron Super's mirror experiment left prefill flat and regressed decode from 28.29 to 11.76 tokens per second, so source mode remained the correct choice for that workload.

More detail is available in `feature-hcs-expert-caching.md` (internal record), although the evidence inventories and current code take precedence where older sections of that design history describe superseded policy.

7. Optional approximate decode: adaptive cold-mass pruning

Exact HCS reduces transfers by keeping useful experts in VRAM. It cannot remove every transfer when the complete expert collection is much larger than the available residency. Adaptive cold-mass pruning addresses part of this remaining traffic, but it does so by changing model execution. It is therefore an explicit approximate mode rather than an invisible extension of HCS.

7.1 The cost of a cold route is not proportional to its router weight

For one routed MoE layer, the expert contribution can be written as:

$$\text{MoE}(x) = \sum(\text{weight}[i] * \text{expert}_i \text{ (internal record)}) \text{ for each selected expert } i$$

The router weight scales an expert's contribution to the layer output. It does not scale the number of expert bytes that must be transferred. If two selected experts are cold, a low-weight expert costs the same full packed transfer as a high-weight expert with the same geometry.

This creates a byte-to-mass mismatch. Some routes contribute only a small part of the selected router mass but still place a complete expert transfer on the serialized decode path.

Residency changes the decision. Omitting a resident expert saves no demand DMA, so it would accept approximation without gaining the intended speed benefit. For this reason, adaptive cold-mass pruning is not a static expert blacklist and does not simply reduce top-k. It decides from the routes and HCS state of the current token and layer. The same low-ranked expert is retained when resident and may become eligible when cold.

7.2 The two-part policy

A policy is written as P/M, where P is the percentage of leading router ranks that are always protected and M is the maximum percentage of routed mass that may be omitted in one token/layer routing event.

The runtime applies the following rule:

1. Protect $\text{ceil}(\text{top-k} * P / 100)$ leading routes.
2. Among the remaining routes, consider only experts that would require demand DMA from host memory.
3. Order those candidates from the smallest positive router weight upward.
4. Admit candidates to the drop set only while their combined weight remains at or below M percent of the complete selected routed mass.

For Ornith's top-10 routing, 75/8 protects ranks 1–8 and allows only cold rank-9 and rank-10 experts to become candidates. The 8 is a ceiling for an individual token and layer. It does not mean that the runtime normally discards 8% of the model's routed mass.

Resident routed experts have zero cold bytes and are therefore ineligible. Shared experts use a separate VRAM-resident execution path and never enter the candidate set. The planner reads the real route weights, current HCS/source state, and actual loaded expert byte sizes, so it does not assume that the Ornith expert size or the RTX PRO 6000 residency applies to another model or GPU.

7.3 Why surviving weights are not renormalized

After an expert is omitted, the remaining router weights could be scaled upward so that they sum to the original total. Krasis deliberately does not do this.

Neither choice recreates exact output. Leaving the weights unchanged removes the omitted contribution but preserves the coefficients assigned to every surviving expert by the trained router. Renormalization also removes the omitted contribution, then increases every surviving contribution and changes the scale of the MoE residual branch.

The current evidence does not establish that this second change is safer for Ornith. Krasis therefore makes only the bounded omission that the policy declares. Renormalization would be a separate approximate algorithm and would require its own default-off A/B and quality validation.

7.4 Shadow measurement before changing output

The policy was selected from a shadow-only Rust sweep. The runtime recorded the actual routes, weights, residency, expert bytes, and projected drop plans while leaving model output unchanged. This allowed several protection and mass combinations to be evaluated in one instrumented Ornith-397B run before any approximate speed test.

The final 249-token shadow block contained 19,518 demand-cold routed-expert activations, or 78.39 per token. With the leading 75% of ranks protected, the measured candidates were:

Shadow policy	Projected drops/token	Projected MiB saved/token	Mean omitted routed mass	Maximum event
75/3	0.313	1.938	0.0125%	2.9912%
75/5	3.876	23.980	0.2673%	4.9995%
75/8	18.791	116.270	1.8668%	7.9993%
75/10	21.514	133.118	2.2322%	9.9796%

The table revealed a useful knee. Moving from a 5% to an 8% event cap exposed almost five times as many removable cold routes, while the observed mean omitted mass remained below 2%. Moving from 8% to 10% produced a much smaller additional transfer reduction. The 75/5 and 75/8 policies were therefore selected as the conservative and aggressive runtime A/B cases. This selection came from the measured Pareto surface rather than from treating 8 as a model-independent constant.

7.5 Preserving the native execution shape

In the ordinary Rust decode path, the drop plan compacts the demand-DMA list before transfer and expert execution. Resident experts remain in the batch and cold experts outside the plan are transferred normally.

CUDA graph replay requires a fixed top-k batch shape. Krasis therefore fills each omitted position with an initialized dummy pointer and a zero router weight. The captured shape remains unchanged, while the batched kernels return without performing expert GEMV work for the zero-weight slot.

The mode does not change routing decisions, alter surviving router weights, change HCS membership, or promote an approximate representation. GPU route synchronization and speculative decode do not currently implement the same residency-aware decision, so startup rejects either combination instead of silently using different semantics.

The launcher exposes `Off`, `75/3`, `75/5`, `75/8`, and `75/10`. `Off` is the default. When enabled, the launcher labels the setting as approximate and the runtime reports the actual cold candidates, drops, saved bytes, mean omitted mass, maximum event mass, and affected ranks.

7.6 Ornith-397B results

The runtime A/B used Ornith-1.0-397B on one RTX PRO 6000 with HQQ4/k4v4, Marlin INT4 routed experts, 42.8% HCS, and the same 50/100/250-token internal decode suite.

Mode	50-token decode	100-token decode	250-token decode	Best network	Result
Exact control	23.58 tok/s	21.85 tok/s	20.40 tok/s	41.73 tok/s	Baseline
75/5	24.00 tok/s	21.83 tok/s	20.34 tok/s	42.26 tok/s	+1.8% only at 50 tokens; longer runs flat
75/8	25.73 tok/s	23.81 tok/s	22.46 tok/s	45.43 tok/s	+9.1%, +9.0%, and +10.1% internal

The conservative case demonstrated why projected byte savings must still be tested end to end. It removed roughly 22–25 MiB per steady token, yet the 100-token and 250-token rates did not improve.

The 75/8 result was consistent across all three decode lengths. In the actual 250-token internal request, exact mode would have fetched 19,918 cold routed experts. The policy omitted 4,754 of them, reducing cold work from 79.992 to 60.900 experts per token and avoiding 118.134 MiB per token. This was 23.87% of all cold fetches.

The omitted router mass was much smaller than the fetch count suggests. Across the token/layer routing events, mean omitted mass was 1.884329%; the largest individual event reached 7.999461%, within the declared 8% ceiling. In the corresponding shadow block, ranks 9 and 10 were only 20% of the router slots but contained 31.15% of the cold routes, and the mass rule admitted 76.93% of this cold tail. The gain therefore came from measured concentration of cold traffic, not from removing one quarter of total MoE contribution.

All 14 network and coherence tests passed for both runtime policies. HCS coverage remained 13,161 of 30,720 experts and minimum free VRAM remained 1,346 MB, so the result was not produced by a different residency budget.

7.7 Cross-model guard and evidence boundary

QCN on the same RTX PRO 6000 provided a useful negative control because all 24,576 routed experts were resident. Exact mode reached 90.93 tokens per second. Explicit 75/8 reached 90.86 and reported zero cold candidates and zero drops throughout warmup, internal decode, and network requests.

This is an important property of the policy. Enabling it does not force approximation when no demand transfer can be avoided. It responds to the current residency state.

The Ornith speed and traffic results are strong, but the quality claim is not yet strong enough. Network coherence tests show that the sampled requests remained usable; they do not measure the distributional effect of thousands of bounded omissions. Ornith does not yet have an authoritative independent llama-witness artifact for the required comparison. Adaptive cold-mass pruning therefore remains default-off and experimental even though its runtime mechanism and speed result are validated.

The result should also not be generalized from 75/8 to every model. Different top-k, route distributions, expert sizes, HCS coverage, and residual scaling can change both the transfer benefit and quality cost. Advancement beyond experimental status requires model-specific witness and quality evidence, not only another clean speed run.

Sources: [adaptive planner and runtime](#), [exact Ornith control](#), shadow sweep (internal record), 75/5 runtime A/B (internal record), 75/8 runtime A/B (internal record), QCN exact control (internal record), and QCN 75/8 guard (internal record).

8. Runtime VRAM calibration

8.1 Why a static budget is not portable

Free VRAM is not simply total VRAM minus file size. Permanent tensors, CUDA contexts, attention profiles, KV formats, graph allocations, sidecar workspaces, allocator caches, prompt length, recurrent state, and transient expert buffers all contribute.

A value measured on one model and GPU cannot safely be copied to another. Even an absolute low-water value measured earlier in startup can become invalid after the allocator retains memory.

Krasis therefore calibrates after the exact model, config, attention backend, and native paths have loaded.

8.2 The decode residency model

Startup runs real short and long workloads with reclaimable HCS absent. A native monitor records driver-visible minimum free VRAM. From these measurements, Krasis models the amount of free memory available for decode at the actual prompt length.

In simplified form:

```
allowed HCS(prompt_tokens)
  = measured no-HCS decode free VRAM(prompt_tokens)
    - safety margin
```

The actual implementation also observes live free VRAM while loading HCS in chunks. The measured model is the primary capacity decision; the live check prevents allocator drift or an unmeasured transient from crossing the floor.

8.3 Transient deltas, not stale absolute values

An early budgeting error reused an absolute minimum-free value from calibration. After calibration, PyTorch's caching allocator retained memory, so current free VRAM was about one gigabyte lower on Qwen3.5-122B. The result was a decode OOM.

The correction was to express a phase's requirement as a transient delta:

```
transient requirement = baseline free - measured low-water free
```

That delta is applied to current post-cleanup free VRAM. This preserves the measurement while accounting for the allocator state that exists when the request actually runs.

8.4 Borrowing VRAM for prefill

Before prefill, the server tokenizes the real rendered prompt and evicts enough reclaimable HCS to cover the measured requirement. During prefill, chunk sizing and live guards can reclaim more if needed. After prefill, only the amount of HCS safe for decode at that prompt length is reloaded.

The reload is asynchronous where possible, so early decode may begin while the reclaimable tier is returning. Pointer entries become visible only after the relevant event completes.

8.5 The safety margin as a diagnostic

The default safety margin is 600 MB. It is not a target to exceed casually, and it is not an excuse to leave many gigabytes unused on a capacity-limited model. When expert residency is constrained, the measured low-water should be close to the margin while remaining above it.

Current RTX 5090 HQQ4 rows show minimum free VRAM of 878 MB for QCN, 856 MB for Qwen3.5-122B, 946 MB for Qwen3-235B, and 952 MB for Qwen3.5-397B. Fully resident 35B models retain much more free VRAM because there are no additional routed experts to load. These two outcomes are both correct.

The current budgeting design is described in `feature-vram-capture.md` (internal record).

9. Multi-GPU execution

Krasis assigns contiguous decode-layer segments to the selected GPUs. Each device owns its segment's attention or recurrent state, CUDA graphs, expert backing descriptors, and independently budgeted HCS pool. The hidden and residual state crosses the device boundary; expert weights do not.

This is a pipeline across the depth of the model, not a reduction across every layer. It has two useful properties. First, adding a GPU adds expert residency as well as arithmetic capacity. Second, the inter-device payload per token is the hidden state rather than a set of expert weights.

The historical 2×RTX 5090 AWQ/Polar4 evidence is strong enough to establish the architecture but not to serve as a current production-quality benchmark. On that surface, QCN HCS increased from 68.6% to 100% and decode from 94.0 to 107.5 tokens per second. Qwen3.5-122B HCS increased from 38.7% to 86.1% and decode from approximately 50 to 66 tokens per second.

Each auxiliary GPU is calibrated according to the state it owns. The GPU that runs prefill must participate in the reclaim-before-prefill cycle. A device that only owns a decode segment can devote its remaining measured capacity directly to

resident experts.

The current implementation supports an arbitrary number of selected devices through layer segments and per-device state. Broad current-quality $N > 2$ and multi-model evidence remains a publication gap and is listed in Section 16. The design history is in `feature-multi-gpu.md` (internal record).

The 2026 GLM programme refined this conclusion for mismatched cards. A capacity-converged split is no longer assumed to minimize serialized service time. Startup can measure each selected device's real expert H2D topology and a model-derived layer D2D working set, then evaluate every valid contiguous split. It changes the capacity reference only when the predicted win exceeds observed calibration uncertainty. On QCN with an RTX PRO 6000 and A4500, this selected [46, 2] instead of [40, 8], retained all 24,576 experts, and improved timing-disabled decode from 72.46 to 81.02 tokens per second.

A small auxiliary GPU can also be used without owning a serial layer segment. GLM's exact peer tier admitted a disjoint set of 911 experts on the A4500 and served them through a persistent kernel plus bit-exact pinned-host mailbox. Availability or deadline misses returned visibly to the canonical host-cold path. Peer-only decode improved from 4.75 to 5.07 tokens per second. The gain was real but much smaller than the startup estimate because approved-heatmap averages overpredicted live peer hits.

These results do not yet make multi-GPU conversation reuse valid. Session snapshots contain per-device ownership and topology identity, but collective capture, restore, commit, and invalidation are not implemented. Multi-GPU requests therefore report `multi_gpu_pending` and use full prefill; they do not publish a partial cache hit from one device.

10. Runtime, memory, and kernel optimisations

The phase split and HCS provide the architecture, but they only work well if the token path is kept small and transfers use the hardware effectively. This section covers the accepted lower-level work.

10.1 One cached routed-expert representation

Krasis reads model weights and produces a versioned Marlin-packed INT4 expert cache. The representation is used by GPU prefill and GPU decode. It is not repacked or quantized per request.

This avoids a Python conversion loop and limits expert bytes, while preserving one canonical execution cache. Historically, a separate sequential CPU layout was necessary when CPU decode was the production design. Once exact GPU HCS decode replaced that architecture, retaining a full second CPU execution copy was no longer justified.

10.2 Contiguous pinned host backing

Individual expert objects would require many allocations and registrations. Krasis stores experts in per-layer contiguous arrays and exposes borrowed views for individual experts. For QCN this avoided roughly 40 GB of duplicate RAM and reduced host registrations from approximately 24,576 to 192. For GLM-4.7 the avoided duplicate was approximately 172 GB.

The real backing is pinned before transfer performance is measured. This point changed an important conclusion: an early pre-pin probe reported 13.54 GB/s on the RTX PRO 6000, while the corrected pinned probe measured 25.91 GB/s. The real four-copy expert layout reached 25.91 GB/s and a hypothetical single contiguous copy reached 26.43 GB/s, only about 2% more. As a result, a disruptive one-copy cache redesign was rejected.

10.3 Pipelined demand DMA

Cold transfers use pinned sources, multiple staging regions where appropriate, and explicit events instead of full-stream synchronization. An early integrated MoE development path improved from 3.69 to 1.87 milliseconds per layer and from 5.6 to

11.1 tokens per second, while effective PCIe throughput rose from 5.3 to 9.2 GB/s. Later production-shaped transfers reach approximately 20–26 GB/s on suitable hardware and workloads.

Transfer scheduling is constrained by dependencies and copy-engine topology. The system overlaps work when a real interval exists; it does not infer overlap from asynchronous enqueue alone.

10.4 HQQ attention profiles

Attention and KV state occupy VRAM that could otherwise hold experts. Krasis materializes versioned HQQ4, HQQ6, and HQQ8 attention profiles for native prefill and decode. The selected precision is explicit and is included in heatmap and artifact compatibility.

There is no single best profile for every goal. On QCN, HQQ4 has a +5.33% perplexity delta from its BF16 baseline and HQQ6 +0.04%; both pass the current llama-witness first-token and top-10 gates. HQQ4 often leaves more capacity for HCS and can therefore be faster on constrained cards, while HQQ6 can be much closer to BF16 quality.

Krasis does not claim that selective correction sidecars generally repair HQQ4. A long series of refits, exception groups, and correction searches failed to produce a stable held-out cross-model win. Whole HQQ profiles remain the accepted choice.

10.5 Model-specific fixed residency

Not every large weight family is a routed expert. Nemotron Mamba2 contains dense projection residency that competed with HCS but should not be managed as an HCS expert.

Quantizing those projections separately reduced Nemotron Super's fixed projection footprint from 8,360.0 MB BF16 to 2,220.6 MB INT4. Startup free VRAM rose from 14,430 to 20,232 MB, HCS coverage increased from 4,922 to 7,038 of 20,480 experts, decode improved from 28.94 to 41.96 tokens per second, and the prompt cap rose from 7,422 to 15,191 tokens.

This illustrates a general rule: memory classes should be managed according to their execution role. A fixed dense projection, a routed expert, KV state, and a temporary workspace are not interchangeable merely because all consume VRAM.

10.6 Native attention sidecars

FA2, FLA, Marlin, and generated native kernels are built as explicit sidecars with known compatibility and error propagation. This avoids runtime Python JIT dependencies and allows the release process to ship verified platform artifacts.

If a requested backend is unavailable, the system reports that fact. A launch failure is not replaced by a different backend and then reported as though the requested test succeeded.

10.7 Memory-mapped caches and page shedding

Large caches are memory mapped with source identity. Pages that are no longer needed are released incrementally with the relevant operating-system advice. This prevents repeated startups from retaining an unnecessary combination of application mappings and page cache, and supports current 397B and 480B runs without requiring another full in-memory copy.

10.8 Faster offline cache construction

Some optimisations improve preparation rather than serving. The versioned CUDA search implementation for HQQ4 cache building evaluated the same quantization candidates on GPU. Real QCN q, k, and o projection samples measured 9.75×, 16.50×, and 17.70× speedups with RMSE ratios within 1.000003 of the established builder. A complete QCN cache built in 32.0 seconds and passed the 8/8 witness suite.

This is reported as an offline build improvement. Runtime speed was unchanged, which is the expected result.

10.9 Optional pinned HQQ stage backing

Pinning the real QCN HQQ stage backing reduced a 947 MB stage swap from 151.381 to 112.684 milliseconds. The timing-off speed test improved prefill from 6,892.2 to 7,723.2 tokens per second, while decode changed by +0.5%, HTTP by -1.2%, HCS was unchanged, and the witness passed 8/8.

The option remains explicit rather than universal because it registers about 1.9 GB of host memory. This is a measured high-RAM trade, not a default assumed to help every machine.

10.10 Exact auxiliary-GPU expert service

Serial layer splitting is not the only way to use another GPU. Krasis can keep a disjoint expert tier on one auxiliary card and use a persistent kernel to exchange a hidden vector and weighted partial output through a pinned-host mailbox. The complete host expert cache remains canonical. Directory entries are published only after all four Marlin ranges have arrived, and a deadline or availability miss executes the ordinary exact cold path.

On RTX PRO 6000 plus A4500, the mailbox measured 27.336/29.441/29.665 microseconds p50/p95/p99 and the A4500 held 911 GLM experts without overlapping primary HCS. Llama-witness passed 8/8 generated tokens, including a visible deadline fallback. This is functional and measured exact service, not proof that every spare card helps: the peer-only matrix gain was 6.7%.

10.11 Bit-exact GPU entropy decoding for cold experts

The accepted GLM lossless codec stores an expert-contiguous sidecar bound to a deterministic hash of the exact Marlin source. Startup validates geometry, indices, and source identity, registers bounded mappings, reconstructs sampled production experts byte-for-byte, and measures one-, two-, and four-range plans on the actual GPU.

The lane-256 artifact reduced 373.712 GB to 322.369 GB, a 13.7389% saving. Its selected two-copy/one-decoder plan measured 682.42 microseconds p95 per 19.46 MB expert versus about 748.33 microseconds raw. A previous lane-1024 artifact compressed more but took roughly 914 microseconds and was rejected. This is why the selector minimizes measured p95 rather than compressed size.

The operational cost is substantial. Hashing, mapping, pinning, and verifying both a roughly 375 GB source and 322 GB sidecar adds about ten minutes to the measured startup. Compression also partly competes with peer residency and pruning because an already peer-served or omitted expert has no local transfer left to compress.

10.12 Model-specific state under a common execution architecture

DeepSeek-V4 and GLM-5.2 show the boundary between generic system design and model-specific mathematics. Both retain full-GPU prefill, Rust/CUDA decode, HCS, measured VRAM, canonical host backing, and fail-closed artifacts. Neither is represented as ordinary GQA.

DeepSeek-V4 owns four-stream hyper-connections, raw sliding KV, learned ratio-4 and ratio-128 compression, indexed sparse attention, source FP4 routed experts, and a dedicated final head. GLM owns compact MLA state, DSA query/key/index projections, and one graph-stable resource per IndexShare owner. Workspace, cache, and top-k capacities derive from loaded config and runtime context.

This separation prevented several false milestones. DeepSeek startup once reached a zero-head GQA placeholder and was changed to fail before execution. GLM exact-prefix mode is permitted only when every causal position is within the checkpoint's selector capacity. Sparse/long execution requires the real DSA producer graph rather than being described as dense MLA compatibility.

10.13 Component-attributed Ornith decode tuning

Timing-only graph markers explained 26.96 of 27.33 milliseconds in Ornith's middle graph: routed MoE 12.48, attention 10.02, routing/top-k 4.45, and only 0.37 milliseconds residual. This excluded a large invisible host bucket and made the next work measurable.

Retained HQQ group-cache, wide route-prep, parallel top-k, and measured shared Marlin split changes improved adjacent timing-disabled 50/100/250-token decode from 23.64/21.91/20.27 to 26.23/24.16/22.27 tokens per second. The full 291-window perplexity remained exactly 3.7727 and minimum free VRAM remained 1,314 MiB. Two bit-exact activation/W2 candidates were removed because their apparently large sampled interval was overlapping block work, not serialized headroom.

10.14 Transactional RAM-backed conversation state

Status correction, 19 October: the following describes the implementation and its August benchmark, not a completed end-to-end equality proof. The retained full-history versus resumed counterexample remains open.

The session cache stores exact rendered tokens, absolute positions, cache layout, template identity, topology, and architecture-specific state. Requests select the longest hash-indexed prefix only after full token verification. Compressed KV, recurrent/conv/SSM state, MLA/DSA resources, and DeepSeek compressor/index state each retain their own checked snapshot contract.

The active conversation can continue from device state. Inactive sessions move through reusable double-buffered pinned staging into pageable RAM. Stable writeback preserves prior token rows and copies only appended KV plus complete fixed state. Reservations are non-mutating; a new snapshot and all staged LRU evictions become visible together only after successful SSE completion. A TCP reset or failed admission leaves committed residency unchanged.

The fixed QCN benchmark forced six real RAM restores. Continuation time improved 1.132x and the complete interleaved workload 1.067x. Restore averaged 45.155 milliseconds, while eight transactional saves averaged 231.207 milliseconds; save cost is therefore the largest remaining cache overhead. Default-on speed was neutral within the adjacent v1.0.20 comparison.

10.15 Native model-declared tool grammars

Krasis does not treat tool calls as one universal delimiter. The Rust renderer and response parser implement the grammar declared by the loaded checkpoint: DeepSeek DSML, Qwen JSON/function XML, GLM argument XML, Gemma native calls, or MiniMax native calls. Streaming and non-streaming responses share the parser and emit typed OpenAI `tool_calls`. Malformed or truncated blocks remain visible assistant text.

Live Opencode round trips passed QCN, Ornith, DeepSeek-V4, Step, Gemma, and preview GLM-4.7. Nemotron Super demonstrates the evidence boundary: prompt rendering matched Hugging Face byte-for-byte and the parser was correct, but the quantized model did not reliably emit its declared grammar. Parser permissiveness was not used to conceal the model-quality limitation.

10.16 Experimental sub-4-bit expert storage

The first TileQ-Score uses a distinct KTQ1 cache with packed signed INT3 residuals, group scales, and shared rank-32 two-dimensional tiled BF16 corrections. The artifact records its BF16 source-tree hash, corpus identity, geometry, unresolved algorithm choices, and actual component bytes. It never reinterprets Marlin INT4 storage or falls back to INT4 during execution.

The Ornith v2 artifact is 3.164897 effective bpw and 23.4248% smaller than Q4. It initially admitted 16,956 experts instead of 13,161, then pressure-evicted 54 and retained a 1,302 MiB free floor. PPL was 3.2456: +4.58% versus BF16 and 0.97% worse than accepted Q4. The result proves the storage/residency mechanism and commercially plausible quality, but it fails the

stricter Krasis production replacement gate. Kernel speed was not measured because the current scalar CUDA implementation is a correctness path.

10.17 Reproducible native distribution

Release artifacts are part of the execution contract. Linux wheels ship the native extension plus verified Marlin, FlashAttention, and architecture-specific FLA sidecars. The native Windows installer stages a release-pinned official embeddable CPython and dependencies into a content-verified private runtime, then atomically changes the active pointer only after relocation, ABI, standard library, extension, and sidecar checks pass. It does not search host PATH, registry Python, user packages, or a mutable venv.

The v1.0.21-rc.1 workflows produced and independently verified the Linux and Windows assets. This does not improve token speed, but it makes the measured runtime installable without silently changing Python or native library inputs.

11. Measurement, calibration, and validation methodology

Krasis is sensitive to asynchronous execution, allocator state, prompt length, expert routes, and cache history. A benchmark can therefore be internally consistent and still measure the wrong thing. The project treats methodology as a technical contribution because incorrect measurement repeatedly led to incorrect architectural conclusions.

11.1 Diagnosis and speed measurement are different modes

Instrumentation is enabled when the goal is to discover where time or memory is being spent. CUDA events, extra synchronizations, counters, byte accounting, route traces, and host calls are valuable in this mode.

They are disabled when the goal is to measure speed. The fixed regression surface is `./dev speed -test`; model-specific standard results use `./dev benchmark <config>`. Benchmark logs state whether timing is enabled.

This separation matters because instrumentation can move work between buckets or change total token time. Several candidates improved an instrumented substage and failed to improve the timing-off benchmark. They were rejected rather than being promoted on the diagnostic result.

11.2 Production and benchmark paths

The internal benchmark uses the same prefill, HCS eviction/reload, native decode, and cleanup mechanisms as a real server request. It removes network formatting from the internal engine measurement, then performs separate HTTP requests to measure the user-visible path.

Three values are reported:

- **internal prefill** measures prompt processing inside the engine;
- **internal decode** measures the native generation loop without network I/O;
- **HTTP round trip** measures the client-side streaming request.

These values answer different questions. The labels and raw definitions must be retained; one field must not be substituted for another or described as the same measurement.

The benchmark contract and known differences are documented in `feature-benchmark.md` (internal record).

11.3 Real prompts and exact token counts

Large-prompt tests use six immutable Project Gutenberg texts: Moby-Dick, War and Peace, Les Misérables, The Count of Monte Cristo, Don Quixote, and The Brothers Karamazov. Each target length uses actual model tokenization.

This replaced generated prompts that repeated five paragraphs and estimated tokens from character counts. Repeated text can create unrealistic route and cache behaviour, cannot support meaningful retrieval checks, and changes size when the generator is edited. A benchmark built from it may be repeatable at the string level while still being unrepresentative of the intended workload.

Heatmap prompts and benchmark prompts are distinct. Otherwise, a route heatmap could indirectly overfit the exact documents later used to report HCS speed.

The prompt policy is documented in `feature-test-methodology.md` (internal record).

11.4 Complete raw logs

Long-running model tests are executed in `tmux` and their complete stdout, server, benchmark report, and failure records are preserved. The benchmark summary links to those artifacts rather than retaining only a copied headline.

This is necessary for values such as minimum free VRAM, HCS coverage, copy failures, prompt length, and internal versus client timing. Without the complete run, two superficially identical token rates may represent different memory pressure or a different execution mode.

11.5 Independent correctness evidence

New correctness work uses BF16 llama-witness artifacts where they exist. The artifact stores exact input token IDs, generation and tokenizer contracts, greedy outputs, first-token information, and top-k data. Krasis replays the stored tokens directly so a chat-template difference cannot be mistaken for an execution error.

Hugging Face Transformers was previously treated as the natural source of truth. Qwen3.5 investigations found unstable or invalid diagnostic logits and inconsistencies between generated tokens and recorded rankings. HF artifacts are therefore retained for forensic use but are not the active acceptance authority.

Krasis's own BF16 path is also not an independent oracle. Shared code can make two local paths agree on the same bug, and one QCN BF16 fused expert diagnostic returned zero rows for the exact experts under investigation. A local BF16 run is useful for localization and for provisional PPL comparison when no external witness exists; it is not equivalent to an independent witness.

11.6 Quality metrics

Exact output tokens are useful but not sufficient. Quantized execution can change an argmax while retaining the reference token near the top of the distribution. Krasis therefore combines:

- first-token agreement;
- prefill argmax and top-10 containment;
- decode top-k containment and exact-prefix diagnostics;
- perplexity or a model-appropriate continuation score;
- network, multi-turn, tool-use, and manual coherence tests.

Quality rows state the reference type. Models without a usable independent witness are marked as using Krasis BF16 or as blocked; their evidence is not silently upgraded.

The current quality table is [STATS-QUALITY.md](#), and the witness consumer is described in `feature-reference-test.md` (internal record).

11.7 Structured first-divergence tracing

When an output differs, the final token does not identify the producer of the error. Krasis can emit a request-scoped structured trace covering runtime contracts, prefill and decode stages, layers, components, tensor summaries, routes, and selected values. Stored traces can be compared offline to find the first divergence and the last matching event.

The trace surface is disabled outside explicit trace runs. This keeps the clean speed path free of diagnostic work and avoids ad hoc logging that changes CUDA ordering. The method has localized Qwen3.5, QCN HQQ, Gemma, and Nemotron issues to specific layer and component boundaries.

The procedure is documented in `feature-trace-debug.md` (internal record).

11.8 Measurement failures that changed conclusions

Three examples show why this methodology is part of the system.

First, H2D bandwidth was initially measured before expert memory was pinned. The probe reported 10.9–13.5 GB/s. The corrected production-shaped measurement was 25.91 GB/s. Optimizing against the first number would have targeted a bandwidth problem that did not exist in the same form.

Second, a host "cold DMA" timer measured enqueue time rather than completion. Qwen3-235B appeared to spend only 4.4 milliseconds per token in that bucket, while 211.8 milliseconds was paid later in synchronization for 4.54 GB of real traffic. CUDA events and byte counters were needed to identify the true cost.

Third, per-token VRAM queries and broad server timing once produced QCN rates that could not be reproduced from committed logs. Moving the monitor out of the token loop and ending decode timing before post-processing restored a truthful boundary.

11.9 Stateful-serving evidence requires transactional and same-boundary gates

A session-cache hit is not validated merely because it produces readable text or avoids prefill. Active-GPU and pageable-RAM restoration must agree exactly at the same token/state boundary, including architecture-specific recurrent or compressor state. Compatibility covers rendered tokens, normalized template, model/cache format, positions, topology, and ownership.

The August method compared cached versus full execution against a measured uncached envelope when prompt heights selected different arithmetic. October work supersedes that method as acceptance for the retained conversation-loading counterexample: diagnostics-off full/repeat/active/RAM checks, genuine reuse and strict complete token/F32 comparison are required. Height-dependent arithmetic is measured separately and cannot waive a failing counterexample. Matching active/RAM copies alone is insufficient.

Operational correctness is also part of the gate. Divergent branches, concurrent requests, cgroup-aware RAM pressure, LRU eviction, and a real TCP reset are tested together. A candidate snapshot, its reservations, and any LRU victims become visible only after successful response completion. The first cache benchmark used a debug-logit path and was discarded; the accepted result uses the normal Rust production response path.

12. Experimental evaluation – August baseline

12.1 Evaluation scope

The current public benchmark set uses one RTX 5090 32 GB or one RTX PRO 6000 Blackwell 96 GB with an AMD EPYC 7742 host. The selected rows below use INT4 routed experts with HQQ4 attention and k4v4 KV. Timing instrumentation is disabled.

The table includes model size, active parameters when available, internal prefill, internal decode, HCS coverage, and minimum free VRAM. The complete public table also includes HQQ6/k6v6 and HTTP results.

Hardware	Model	Total / active parameters	Prefill	Decode	HCS	Min free VRAM
RTX 5090 32 GB	Qwen3.6-35B-A3B	35.5B / 3.0B	10,670.1 tok/s	117.20 tok/s	100.0%	10,186 MB
RTX 5090 32 GB	Qwen3-Coder-Next	80B class / n/a	7,028.5 tok/s	87.90 tok/s	66.6%	878 MB
RTX PRO 6000 96 GB	Qwen3-Coder-Next	80B class / n/a	11,211.1 tok/s	91.34 tok/s	100.0%	53,156 MB
RTX 5090 32 GB	Qwen3.5-122B-A10B	122B / 10B	3,692.5 tok/s	31.80 tok/s	36.0%	856 MB
RTX 5090 32 GB	Qwen3-235B-A22B	235B / 22B	1,385.3 tok/s	5.64 tok/s	17.3%	946 MB
RTX 5090 32 GB	Qwen3.5-397B-A17B	397B / 17B	973.8 tok/s	10.04 tok/s	8.9%	952 MB
RTX 5090 32 GB	Ornith-397B	397B / 17B	743.7 tok/s	7.89 tok/s	8.9%	952 MB
RTX PRO 6000 96 GB	Ornith-397B	397B / 17B	2,354.5 tok/s	23.58 tok/s	42.8%	1,346 MB
RTX 5090 32 GB	Step-3.7-Flash	201.4B / 13.9B	2,618.4 tok/s	23.05 tok/s	23.0%	884 MB
RTX PRO 6000 96 GB	Step-3.7-Flash	201.4B / 13.9B	5,261.0 tok/s	55.40 tok/s	91.9%	1,274 MB
RTX 5090 32 GB	Gemma-4-26B-A4B	26B / 4B	5,598.9 tok/s	63.57 tok/s	100.0%	12,084 MB
RTX 5090 32 GB	Nemotron-3-Super-120B-A12B	123.6B / 12.4B	1,852.2 tok/s	41.87 tok/s	34.4%	906 MB
RTX 5090 32 GB	Nemotron-3-Nano-30B-A3B	31.6B / 3.2B	8,583.9 tok/s	151.76 tok/s	100.0%	11,722 MB

Source: [current benchmark table](#).

12.2 What the table establishes

The table establishes that one architecture can operate over several distinct regimes:

- fully resident expert sets on 26–35B models;
- partial residency on 80–123B models;
- low residency on 201–397B models on a 32 GB card;
- much higher residency for 201–397B models on a 96 GB card;
- conventional GQA, sliding/full mixtures, Gated DeltaNet, Gemma-specific geometry, and Nemotron Mamba2.

It does not establish that total parameter count predicts speed by itself. Active parameters, top-k, expert size, attention geometry, HCS coverage, prompt length, and hardware all matter. Nemotron Super, for example, has a similar parameter scale to Qwen3.5-122B but a different recurrent and routing structure.

12.3 Prefill scaling

Prefill declines as models become larger and more complex, but it remains a GPU throughput path. On the RTX 5090, Qwen3.6-35B reaches 10,670.1 tokens per second, QCN 7,028.5, Qwen3.5-122B 3,692.5, Qwen3-235B 1,385.3, and Qwen3.5-397B 973.8.

The exact sequence is not a simple inverse of total parameters because weight reuse, attention family, active experts, prompt chunking, and backend kernels differ. It does show that a model whose full weights greatly exceed VRAM can still process a long prompt at hundreds or thousands of tokens per second.

12.4 Decode and residency

Decode speed generally falls as active work rises and HCS coverage falls. The important result is not that every very large model becomes fast in absolute terms. Qwen3-235B at 17.3% HCS remains 5.64 tokens per second on a 5090. The result is that the runtime degrades continuously with available residency rather than requiring the complete model to fit or redirecting the whole expert path to CPU compute.

The 96 GB card demonstrates the value of this continuity. Step's HCS rises from 23.0% to 91.9% and internal decode from 23.05 to 55.40 tokens per second. Ornith's HCS rises from 8.9% to 42.8% and decode from 7.89 to 23.58. These are not GPU-memory-only ablations because the GPUs differ in other properties, but they show that extra capacity is converted into useful expert residency.

12.5 HQQ4 and HQQ6

The current public table includes two attention/KV profiles. HQQ4 usually uses less VRAM and often improves HCS or prefill; HQQ6 is usually closer to the BF16 quality baseline.

For Qwen3.5-122B, HQQ4 reaches 3,692.5 prefill and 31.80 decode tokens per second with 36.0% HCS; HQQ6 reaches 3,389.2 and 28.83 with 34.1% HCS. Their PPL deltas are +3.25% and -0.73%, and both pass 14/14 first-token and top-10 witness checks.

For QCN, decode is effectively flat: 87.90 for HQQ4 and 88.22 for HQQ6. HQQ6's PPL delta is +0.04% rather than HQQ4's +5.33%. This is why profile choice is exposed rather than reduced to a universal "lower bits are faster" rule.

12.6 Same-card Ornith-397B system comparison

The clearest current external comparison uses one RTX PRO 6000 Blackwell 96 GB and the same EPYC 7742 host.

The llama.cpp run used build 8233, commit c5a778891, Ornith-397B Q4_K_M, 64 CPU threads, FlashAttention, F16 KV, memory mapping, -ngl 999, and n_cpu_moe=38. An OOM-gated search began with maximum GPU MoE residency and increased CPU MoE layers only until the model fit. Settings below 38 did not load; 38 was the fastest passing boundary found and used approximately 92 GB VRAM.

System	Format	Prompt 512	Prompt 2,048	Prompt 8,192	Generate 128
llama.cpp	Q4_K_M, F16 KV, n_cpu_moe=38	131.38 tok/s	176.51 tok/s	185.01 tok/s	10.71 tok/s
Krasis	INT4/HQQ4/k4v4	—	—	best standard prefill 2,354.5 tok/s	best standard decode 23.58 tok/s

The quotients of the reported summary values are approximately 12.7× for prefill and 2.2× for decode. These are not prompt-length-matched speedups: the Krasis values are the best rows from its standard prompt and decode suites, while the llama.cpp values shown are pp8192 and tg128.

This is not a controlled kernel ablation. The quantization formats, KV formats, prompt suites, and benchmark harnesses differ. The Krasis and llama rows were also not collected as one same-day paired harness. The result supports a narrower statement: on this machine, the best fitting llama.cpp GPU-offload configuration found still had to leave 38 MoE layers on CPU and did not approach Krasis’s recorded long-prompt or decode performance for the same model class.

Sources: [llama full-run log](#), [reproducible sweep](#), and [Krasis benchmark index](#).

12.7 Quality evidence

The following selected rows illustrate the current quality surface.

Model	Profile	Reference	PPL delta	Prefill top-10	First token	Status
Qwen3.6-35B	HQQ4/k4v4	llama-witness BF16	+3.28%	8/8	8/8	PASS
QCN	HQQ4/k4v4	llama-witness BF16	+5.33%	8/8	8/8	PASS
QCN	HQQ6/k6v6	llama-witness BF16	+0.04%	8/8	8/8	PASS
Qwen3.5-122B	HQQ4/k4v4	llama-witness BF16	+3.25%	14/14	14/14	PASS
Qwen3.5-122B	HQQ6/k6v6	llama-witness BF16	-0.73%	14/14	14/14	PASS
Qwen3-235B	HQQ4/k4v4	llama-witness BF16	+4.74%	14/14	14/14	PASS
Step-3.7	HQQ4/k4v4	llama-witness BF16	+3.80%	8/8	8/8	PASS
Gemma4-26B	HQQ4/k4v4	Krasis BF16 chat continuation	+9.14%	197/197	14/14	PASS, provisional oracle
Gemma4-26B	HQQ4+10%/k6v6	Krasis BF16 chat continuation	+4.47%	197/197	14/14	PASS, provisional oracle
Gemma4-26B	HQQ6+10%/k6v6	Krasis BF16 chat continuation	+0.29%	197/197	14/14	PASS, provisional oracle
Ornith-397B	HQQ4/k4v4	Krasis BF16	+3.58%	n/a	n/a	PASS, provisional oracle
Nemotron Super	HQQ4/k4v4	Krasis BF16 diagnostic	+16.30% diagnostic	n/a	n/a	BLOCKED
DeepSeek-V4-0731	INT4/BF16 KV	llama-witness + frozen scalar anchor	+0.114% vs scalar anchor	4/4	4/4	PASS
DeepSeek-V4-0731	HQQ4+10%/Native	HQQ8/Native + llama-witness	+0.856%	4/4	4/4	PASS
DeepSeek-V4-0731	HQQ6+10%/Native	HQQ8/Native + llama-witness	+0.203%	4/4	4/4	PASS
GLM-5.2 sparse	HQQ4/k4v4	llama-witness BF16	n/a	8/8 top-10, 7/8 argmax	first 4 contiguous	PASS at production quantization boundary

The rows show why the reference column matters. Ornith’s output and PPL evidence are useful, but there is no independent llama-witness artifact yet. Nemotron’s local BF16 diagnostic produced pathological absolute PPL, so the model’s quality status remains blocked despite strong speed results.

DeepSeek’s delta is against the frozen scalar runtime, not a generic BF16 label, because source FP4 and the target cache contract are part of the model’s quality surface. GLM’s sparse row is accepted with one witness token at native rank two and every witness token in top-10; it is not described as distribution equivalence.

Source: [current quality table](#).

12.8 Adaptive cold-mass pruning

[Section 7](#) explains the policy, shadow selection, graph execution, and evidence boundary in full. The central experimental result is that Ornith 75/8 improved 50/100/250-token internal decode by 9.1%, 9.0%, and 10.1% while HCS coverage and minimum free VRAM remained unchanged.

The traffic result explains the speedup more directly. The 250-token request removed 23.87% of cold fetches and avoided 118.134 MiB per token while omitting 1.884329% mean routed mass. A fully resident QCN control produced zero candidates and zero drops. These results validate the mechanism; they do not close the independent Ornith quality gap, so the mode remains explicit, default-off, and experimental.

12.9 GLM-5.2 cold-bandwidth matrix

GLM-5.2 uses 75 MoE layers, top-8 routing, 256 experts per layer, and a 19,200-expert bank. On one RTX PRO 6000, the approved exact HCS holds 3,887 experts. Timing instrumentation showed a sustained baseline near 210.5 milliseconds/token: approximately 143 milliseconds of cold transfer and 68 milliseconds of graph/compute. Cold traffic therefore remained the largest serialized interval after attention correctness and compute tuning.

The final matrix used one unchanged binary, the same benchmark/config, no timing instrumentation, and pruning off except where named:

Configuration	Internal decode	Change from raw	Prefill
Raw exact baseline	4.75 tok/s	—	44.7–45.1 tok/s
Baseline + 75/8 pruning	5.27 tok/s	+10.9%	flat
Exact peer tier	5.07 tok/s	+6.7%	flat
Peer + 75/8	5.65 tok/s	+18.9%	flat
Peer + exact lossless compression	5.38 tok/s	+13.3%	flat
Peer + compression + 75/8	5.97 tok/s	+25.7%	flat

The last row reduces token time to about 167.5 milliseconds. The remaining cold interval is still about 100 milliseconds, or roughly 60% of the token; Krasis is not compute-bound at that point. If all remaining cold traffic vanished, the measured 68-millisecond graph/compute floor would imply roughly 14.7 tokens per second.

The result also explains why the original 2x target was not reached. The A4500's measured usable capacity covered 1,037 experts, about 15.7% of the post-primary cold tail. Linearized planning suggests roughly 90 GB of useful peer expert VRAM would be required for an idealized 2x result before dispatch and heatmap-shape losses. The peer selector's initial near-2x prediction was therefore not validated by live service.

12.10 DeepSeek-V4 compressed-attention evaluation

DeepSeek-V4-Flash-0731 uses a 304.2B checkpoint/284B main model with about 13B active parameters. Its main runtime has 43 layers, top-6 routing, 11,008 routed experts, four-stream hyper-connections, raw sliding KV, learned ratio-4/128 compression, and indexed sparse attention.

On one RTX PRO 6000 it holds 6,440/11,008 experts (58.5%). Timing-disabled prefill is 152.2/320.6/906.3/1,328.2/1,204.3 tokens per second at 1K/2K/8.6K/23K/62K. Standard 50/100/250-token decode is 29.38/28.20/28.49 tokens per second; 62K-context decode is 19.41. The large- request low-water is 650 MiB against the 600 MiB margin.

Quality is measured against a frozen scalar runtime because the checkpoint's native source format and the Krasis target representation make a simple BF16 label misleading. Llama-witness passes 4/4 prefill argmax, top-10 containment, and first token. Full 281-window PPL is 4.821375, +0.11369% from the 4.8159 scalar anchor and inside the explicit cumulative quality

budget. A faster gathered-GEMM attempt was removed when its fixed PPLscreen regressed +0.66%, despite passing the narrow witness.

Subsequent attention/cache work improved the memory contract without relaxing that quality bar. Phase-one HQQ8 quantizes the four ordinary main attention projections while retaining grouped wo_a and compressor/indexer projections at their existing precision. Its full PPL is 4.816034369 versus 4.821375389 for the accepted BF16-attention/BF16-cache run (-0.1108%), and llama-witness passes 4/4. This establishes HQQ8/BF16 cache as the measured high-quality comparison profile. Fixed HQQ6/Native later measured 4.838535416 perplexity (+0.466% versus HQQ8/Native) with a 4/4 witness pass, and is the current launcher default. Budget-planned mixed attention also passed without weakening that boundary. HQQ4+10%/Native measured 4.857317863, between fixed HQQ6 and fixed HQQ4, while HQQ6+10%/Native measured 4.825864933, between HQQ8 and fixed HQQ6. Each profile promoted 87/172 eligible tensors within 72.5/75.2 MiB and passed the four-prompt witness at 4/4 argmax, top-10 and first token.

DeepSeek’s architecture-owned Native cache stores the checkpoint’s existing E4M3/E2M1 QAT codes, exact scale exponents and BF16 RoPE tail rather than expanding every persistent row to BF16. It is not a conventional k4v4 or k6v6 mode and introduces no second quantizer. At the same 1,000 MiB budget it increases context capacity from 149,808 to 294,432 tokens (1.965x). Full PPL is 4.816071679, only +0.000775% versus HQQ8/BF16 cache.

HQQ8 cache mode	Prefill 1K / 5K / 39,920	Decode 50 / 100 / 250	Min free VRAM
Expanded BF16	151.4 / 534.1 / 1,301.1 tok/s	30.08 / 29.86 / 29.29 tok/s	1,178 MiB
Native	138.6 / 500.9 / 1,296.5 tok/s	30.23 / 30.17 / 29.48 tok/s	1,276 MiB

Native is accepted as the compact architecture-owned mode. It was initially classified as explicit high-context rather than the balanced default because its 1K/5K prefill regresses 8.45%/6.22%, while 10K and longer prefill is effectively flat and decode improves 0.50–1.04%. The real launcher-started Native server subsequently passed 18/18 network checks, including a 62,403-token canonical prompt at a 656 MiB prefill floor and zero HCS copy failures. Product policy subsequently selected HQQ6/Native as the fresh-launch default to prioritize context capacity; expanded BF16 remains explicit.

The current HQQ6/Native launcher default was then benchmarked independently on the same RTX PRO 6000 under its configured 450 W operating cap. Internal prefill at 1K/5K/10K/20K/35K/39,920 tokens was 124.9/532.8/841.9/1,117.9/1,202.0/1,207.2 tok/s. Internal decode at 50/100/250 requested outputs was 30.12/30.26/30.00 tok/s; HTTP round trip was 56.91/40.64/32.13 tok/s. The run retained 6,660/11,008 HCS experts with a 1,205 MiB floor and zero copy failures. Because the earlier HQQ8 rows preceded the 450 W cap, this is reported as the current operating point rather than a controlled attention-precision speed A/B.

12.11 RAM-backed conversation reuse

The fixed session-cache benchmark interleaves two canonical Gutenberg-derived QCN conversations of approximately 10K tokens, four turns each. Cache-off runs bracket the cache-on run. Switching conversations forces six real pageable-RAM restores rather than leaving one convenient conversation active on GPU.

Scope	Cache off	Cache on	Gain
Six continuation turns	20.306 s	17.945 s	1.132x
Complete interleaved workload	27.071 s	25.360 s	1.067x

All response text and token counts matched; six RAM hits and zero restore failures were recorded. Restores averaged 45.155 milliseconds. Eight saves averaged 231.207 milliseconds and committed 3.276 GB, so save cost—not restore or prefix lookup—is the current optimization target. The separate default-on speed gate was neutral within 0.19% decode/HTTP and 0.197% at the only lower prefill point.

12.12 TileQ-S storage and quality gate

The first Ornith TileQ-S result used the complete BF16 expert source, a hash-bound real calibration corpus, signed scalar INT3 residuals, and shared rank-32 tiled corrections. Its actual artifact, not nominal code width, defines the storage result:

Format	Artifact bytes	Effective bpw	HCS admission	WikiText-2 PPL
Accepted Q4	199,702,609,984	4.125	13,161	3.2146
TileQ v2	152,922,701,824	3164897	16,956 before pressure	3.2456

TileQ is 23.4248% smaller and admits 28.8% more experts before pressure. Its PPL is +4.58% versus BF16 3.1034 and 0.97% worse than Q4. That is a strong commercially plausible low-bit result, but the experiment declared no-worse-than-Q4 before execution, so it is not promoted. No speed row is reported because the current native scalar kernels are correctness implementations. A future attempt must first improve residual quality through a measured GPTQ-style or mixed 3/4-bit map; optimizing the wrong quality point would not establish a production backend.

12.13 Community results and independent reports

Public discussion contains real third-party Krasis runs, not only reactions to author benchmarks. A review of ten supplied Reddit links resolved them to six underlying posts plus four crossposts or superseded duplicates. A related GitHub report adds a native-Windows DeepSeek-V4 run. Four successful reports were recoverable with enough information to classify. Historical runs dominated by Krasis bugs that have since been fixed are excluded from the performance set.

Reporter	Hardware	Model	Community-reported result	Classification
Lonely_Drewbear	Ryzen 9900X, 64 GB DDR5-6000, RTX 5080 16 GB, WSL2	Qwen3.5-35B-A3B, early INT4/BF16 architecture	3,302.5 tok/s internal prefill; 5.00 tok/s internal decode	Historical independent run; predates current GPU decode
otietz	ThinkPad P16 Gen 2, RTX 5000 Ada 16 GB, WSL2	Qwen3.5-35B-A3B	approximately 20–30 decode tok/s	Operational report; exact RAM/config/prompt absent
shALKE	RTX 5090; host/profile not reported	Qwen3-30B, Qwen3.5-35B, QCN	7,469.9/89.23, 3,682.7/88.78, and 3,844.2/51.97 prefill/decode tok/s	Strongest multi-model third-party performance report, but not exact-SHA reproducible
sebastian-online-ai	RTX PRO 6000 Blackwell 96 GB, native Windows	DeepSeek-V4-Flash-0731, v1.0.20, INT4 experts/BF16 attention/BF16 KV	First request: 5.4 prefill, 10.4 decode tok/s; second: 7.8 prefill, 14.4 decode; 28-token prompts, 597/390 generated tokens, 84.98/85.78% HCS hit	Successful direct-runtime/API run; launcher could not select the validated BF16/BF16 profile, and the request is not the standard benchmark

The nearest current internal rows are materially newer: on RTX 5090, Qwen3.5-35B reaches 10,700.7 prefill and 116.43 decode tokens per second, while QCN reaches 7,028.5 and 87.90. On the RTX PRO 6000, the controlled DeepSeek-V4 row reaches 29.38/28.20/28.49 internal decode tok/s with 93.89–95.02% timed-leg HCS hits. The community DeepSeek prefill uses only 28 prompt tokens, while the paper’s curve starts at 1,000, so its 5.4/7.8 tok/s values are dominated by a different workload shape. These values are printed beside the community results in the full inventory, but they are not treated as speedup ratios because revisions, hosts, operating systems, HCS hit rates, profiles, and prompts differ.

This changes the external-validity verdict. Krasis is not supported only by its authors’ machines: independent users have installed and run it on RTX 5000 Ada, RTX 5080, RTX 5090, and RTX PRO 6000 systems under WSL2, Linux, and native Windows. The evidence is nevertheless community-reported rather than controlled. No accessible report binds the current release SHA to a complete config, canonical raw log, quality witness, and the current 600 MB safety contract. The correct conclusion is therefore that independent operational and historical performance evidence exists, while exact current-release replication remains open.

13. Case studies

13.1 Qwen3-Coder-Next: the mature production path

QCN is the most mature Krasis model. On an RTX 5090 it reaches 7,028.5 prefill and 87.90 decode tokens per second with 66.6% HCS under HQQ4/k4v4. Its minimum free VRAM is 878 MB, close enough to the 600 MB safety floor to show that the cache is using capacity aggressively without crossing it.

On the RTX PRO 6000, all 24,576 routed experts fit. Prefill rises to 11,211.1 tokens per second and decode to 91.34. The smaller decode change is informative: QCN already had enough residency on the 5090 that remaining token time includes attention, routing, expert compute, and serving work that more HCS cannot remove.

QCN also provides the strongest current witness evidence. HQQ4 and HQQ6 pass all eight first-token, prefill argmax, and prefill top-10 cases; HQQ6 is within +0.04% PPL of the local BF16 scoring baseline.

13.2 Qwen3.5-122B and Qwen3-235B: progressively constrained residency

Qwen3.5-122B keeps 36.0% of experts resident on the 5090 and reaches 31.80 tokens per second. Qwen3-235B keeps 17.3% and reaches 5.64. Both continue to prefill fully on GPU at 3,692.5 and 1,385.3 tokens per second.

These models make the limiting factor visible. Qwen3-235B moves 4.54 GB of cold expert data per token in one diagnostic and pays most of that cost at a later synchronization point. Removing a 0.15-millisecond classification bucket or making route IDs stay on GPU cannot solve a hundreds-of-milliseconds transfer dependency.

The architecture still provides value at this low-residency point, but it does not conceal the result: 5.64 tokens per second is usable for some workloads and slow for others. The path to improvement is greater effective residency, reduced exact cold bytes, a materially different interconnect, or a safe and validated approximation—not a small host-side optimization.

13.3 Ornith-397B: the clearest single-GPU system result

Ornith-397B combines a very large total expert collection with 17B active parameters. On a 32 GB 5090, 8.9% HCS produces 743.7 prefill and 7.89 decode tokens per second. On the 96 GB RTX PRO 6000, 42.8% HCS produces 2,354.5 and 23.58.

The llama.cpp comparison is valuable because it shows the difference between a system that must leave a large group of MoE layers in CPU execution and one that uses CPU memory as exact backing for a GPU-residency cache. The comparison does not prove every Krasis kernel is faster. It shows that the phase-separated architecture handles this memory regime much more effectively in the recorded configuration.

Ornith is also the strongest adaptive cold-mass pruning result. Explicit 75/8 raised the three internal decode lengths by 9–10% by removing 23.87% of demand-cold fetches. The exact 23.58-token-per-second result remains the published baseline, and the approximate result remains experimental until an independent witness exists.

13.4 Step and Gemma: mixed attention geometry

Step combines routed experts with a mixture of attention behaviours. Selecting the correct sliding and full backend per layer improved prefill substantially, while high HCS on the RTX PRO 6000 raised decode to 55.40 tokens per second.

Gemma required a different set of graph and head-dimension-512 attention segments. Many narrow fusions and staging changes were flat or slower. The accepted gains came from changing graph boundaries and attention geometry, raising k6v6

decode from 38.79 to 63.69 tokens per second and current HQQ4/k4v4 prefill to 5,598.9 tokens per second.

These cases argue against a single universal attention kernel. The system needs one architecture for residency and execution ownership, but guarded native implementations for the actual layer geometry.

13.5 Nemotron: extending beyond ordinary transformer attention

Nemotron combines routed experts with Mamba2 state-space layers. Supporting it required separate accounting and quantization for fixed Mamba2 projections, parallel SSD prefill behind correctness gates, and graph replay for recurrent decode.

The August Nemotron performance record reports Super at 1,852.2 prefill and 41.87 decode tokens per second on the RTX 5090 under HQQ4/k4v4, and Nano at 8,583.9 and 151.76. Later HQQ4/k4v4 first-token llama-witness rows passed six prompts each, although reported drift is larger than mature Qwen rows. HQQ6/k6v6 remains blocked in the reviewed quality ledger. Pathological local BF16 corpus PPL is not a valid quality reference. These limited witness results do not establish productive ADQ qualification.

This is an intentional separation of claims: the runtime and kernels execute the model quickly; production quality acceptance requires stronger evidence.

13.6 GLM-5.2: exposing the cold-transfer floor

GLM-5.2 required native MLA, DSA, and IndexShare rather than a config alias. Corrected sparse witness evidence retains 7/8 teacher-forced argmaxes and all eight witness tokens in native top-10; the short exact-prefix control is 8/8. The production quantization boundary is therefore explicit rather than called BF16-equivalent.

Once correctness stabilized, GLM became the clearest cold-bandwidth laboratory. At 3,887/19,200 expert residency, sustained decode moved about 3.82 GB of cold expert payload per token at roughly 26 GB/s. Exact split launch hid about 20.5 milliseconds of resident work beneath demand H2D, the approved heatmap improved live hit rate, an exact peer tier removed a bounded additional route population, and a GPU codec removed 13.7% of remaining bytes.

The case also produced unusually informative failures. 75/10 pruning was fast but moved later log probabilities materially. One CPU tail proved the scheduling concept but remained a small approximate adjunct; a 277 GiB transposed duplicate gained only another 3-4%; a second worker regressed. A universal CPU layout plus GPU repacking was byte-exact and still lost 2.5-3.5% on QCN. Dynamic peer was flat, and deeper entropy streaming was slower than the grouped decoder. The accepted conclusion is a measured portfolio of exact residency and byte-reduction mechanisms, not one universal trick.

13.7 DeepSeek-V4: architecture-specific math without a fallback

DeepSeek-V4 uses source E2M1 experts, E4M3 dense/shared matrices, four-stream hyper-connections, learned compressors, indexed sparse attention, hash routing, and DSML. Several superficially similar Krasis paths were deliberately not reused: the model is neither ordinary GQA nor GLM DSA, source FP4 is not symmetric INT4, and its repository has no generic Jinja template.

Bring-up advanced through fail-closed boundaries: mixed routed/shared source format, architecture-owned KV state, dedicated descriptors, ratio-0 attention, compressor/index equations, complete GPU prefill, llama-witness, full PPL, and only then performance work. A zero-head placeholder and a CUDA grid-dimension failure were caught before being mistaken for valid execution.

The final runtime shows why quality budgets must be explicit. Several BF16-input/FP32-accumulate GEMMs delivered large prefill gains with only +0.11369% cumulative PPL movement. An earlier gathered implementation was even faster and passed four witness prompts, but its +0.66% fixed-screen PPL change violated the then-declared gate and was removed. The project changed numerical policy only through an explicit user-authorized budget, never by silently rebasing after a fast result.

13.8 Agent serving: prefix-state contracts and native tools

Long IDE/agent prompts motivated full-GPU prefill; multi-turn agents add a second opportunity because most of the next prompt is stable history. The session cache treats the rendered prompt and every state producer as one exact compatibility contract. This extends beyond transformer KV to Qwen recurrent state, DeepSeek compressed/index state, MLA/DSA, and other supported families.

The same request can contain native tool schemas. The renderer follows the checkpoint grammar and the response parser translates only complete valid native calls. Live Opencode results show that transport correctness and model quality are separate: several models complete tool/result round trips, while Nemotron's correct renderer/parser cannot compensate for a quantized model that does not reliably choose the tool.

The session result is modest compared with eliminating a 10K-token prefill because transactional save cost remains large. That modesty is useful: it prevents a cache-hit microbenchmark from being reported as agent-level speed when full save, restore, switching, HTTP, and cancellation costs are present.

14. Negative results and what they taught us

The full negative-results catalogue appears in Appendix B. The main lessons are more useful than a chronological list.

14.1 Prediction is not an execution window

Speculative expert prefetch appears ideal for MoE decode. Predict the next layer's experts, move them early, and turn misses into hits.

On the tested consumer and workstation GPUs, there is usually one H2D copy engine. Speculative and demand transfers therefore serialize. A high prediction hit rate can still lose if wrong predictions consume the copy engine, if the prediction arrives too late, or if there is not enough VRAM to retain the staged expert until use.

APFL with ten prefetched experts reached 96% prediction hit in one early test and still regressed 11%. A later Step graph-replay test regressed decode from 21.96 to 14.40 tokens per second. Learned route models, previous-token routes, MTP routes, and preferred-friend planners did not overcome the same execution constraint.

The successful Nemotron prefill lookahead clarifies the lesson. Prediction is not the contribution. A known next layer, reusable batch work, and a real compute interval are the contribution.

14.2 Less data can create more work

Nested INT2/INT3 cold-expert compression reduced transfer bytes in isolated all-cold tests. In steady decode, exact promotion needed the omitted bit planes and restored the transferred bytes. Approximate promotion changed routes and cache residency; preventing promotion destroyed hit rate. The alternatives measured 17.38 and 14.7 tokens per second in the relevant Ornith tests, below the exact design.

Ordinary host codecs did not solve the problem. LZ4 saved only 2.03% on a stratified sample of packed Ornith experts. Zstandard saved 18.94%, but the remaining transfer time implied a decompression requirement above approximately 111 GiB/s to break even, before another serialized host stage was considered.

The later GPU codec succeeded only after changing the complete contract. It trained on the whole expert corpus, kept experts contiguous, reconstructed on a separate CUDA stream, and selected lane/copy grouping from measured live p95. Even then, lane-1024 was slower than raw transfer and deeper streaming was slower than the grouped lane-256 decoder. The accepted result is therefore not "compression works"; it is "one exact GPU codec geometry beats the measured raw critical path on this model and device."

The lesson is not that compression can never work. It is that a useful codec must fit the promotion policy, transfer-call topology, reconstruction cost, and steady HCS trajectory.

14.3 A higher cache hit rate can be slower

Unbounded promotion improved apparent HCS coverage and made Qwen3-235B slower. Repeat-protect victim selection made locally plausible choices and produced a worse future cache. Cross-prompt rebalancing could replace 25% of the global prior after every request and showed no gain over 100 diverse prompts.

HCS is not evaluated on hit rate alone. Promotions, evictions, copied bytes, same-layer feedback, and total milliseconds per token are part of the policy.

14.4 CPU execution does not automatically beat PCIe

Cold-expert CPU execution would transfer only activations and outputs, which are much smaller than weights. In the integrated Step path it regressed to 85.15 milliseconds per token, with CPU cold compute alone at 56.64 milliseconds, and missed one of eight witness first tokens. A later forced scheduling variant was worse.

The standalone CPU kernel was closer to the PCIe floor but did not reproduce the integrated memory, NUMA, routing, and scheduling workload. The accepted design therefore kept explicit pinned DMA and GPU expert compute.

14.5 Fewer launches can reduce occupancy

Skipping W13 k-split work saved launches but made resident experts 7% slower and cold experts 5% slower because the 5090 needed the additional blocks to occupy its SMs. Fusing routing GEMV and top-k into one 512-thread block was 2.2× slower than allowing cuBLAS to distribute the GEMV.

Launch count is not the objective. Token time is the objective, and occupancy, register pressure, memory traffic, and synchronization all contribute.

14.6 Correct local output is not an end-to-end win

Nemotron shared-C passed a bitwise local comparison and regressed prefill. Many Gemma fusions passed narrow correctness checks and were flat or slower in a clean run. DFlash graph capture removed launch work from a verifier dominated by expert and transfer cost.

Krasis therefore promotes a low-level change only after the real model path, timing-off benchmark, and quality gates agree.

14.7 An oracle must be independent

HF output and local Krasis BF16 both produced misleading evidence in specific bring-ups. This does not make them useless. It changes their role. They can localize a problem or provide provisional comparison, but production acceptance requires provenance-checked external witness data where available.

14.8 Optimisations compete for the same cold route

HCS, peer residency, pruning, lossless compression, and CPU tails all appear to reduce cold work. Their effects cannot be added independently. Once an expert is resident on the peer, compressing it saves no local PCIe. Once pruning omits a route, neither peer service nor decompression can improve it. A CPU tail that wins a race also changes which DMA reaches the front of the queue.

The GLM matrix demonstrated this directly. Peer+75/8 reached 5.65 tokens per second. Peer+compression without pruning reached 5.38. The complete combination still improved to 5.97, but much less than the sum of isolated optimistic estimates. Future planners must evaluate marginal exposed wait after preceding decisions, not count the same bytes twice.

14.9 More parallel workers can reduce useful parallelism

One persistent CPU tail could absorb a bounded number of deep GLM routes. A second topology-calibrated team had less queue depth, took about 3.05 milliseconds per attempt, won only 1.48% of claims, reduced H2D bandwidth, and regressed decode. Deeper GPU entropy streaming similarly added progress and polling work and lost to one grouped decoder.

Parallel structure is therefore admitted from measured contention and deadline value. A second worker or finer pipeline is not free merely because it uses a nominally independent CPU core or CUDA stream.

14.10 A commercially good quant can still fail a stricter engineering gate

Ornith TileQ v2 is only 0.97% worse than Q4 perplexity while cutting expert bytes 23.42%. This is a strong efficiency result and plausibly acceptable in a commercial low-bit tier. It still failed the Krasis experiment because the declared goal was no worse than the already accepted Q4.

Keeping both statements prevents two opposite errors: discarding valuable research because it missed an intentionally strict target, or relaxing the target after seeing an attractive result. The artifact, native format, and quality evidence remain useful; production promotion and kernel optimization remain gated.

14.11 Stateful caches fail at commit boundaries, not only copy kernels

The session-cache transfer path was not the hardest correctness problem. Reservation, eviction, concurrent request ordering, divergent branches, template normalization, non-rewindable recurrent state, and cancellation determined whether a snapshot was actually valid.

One live gate found that an uncommitted candidate could evict an older committed conversation before SSE delivery. The fix made reservations non-mutating and staged LRU victims until commit. This is the stateful analogue of CUDA stream ownership: publication order is part of correctness even when every individual copied byte is exact.

15. Related system position

Krasis is aimed at a specific operating point: very large MoE models whose active computation is manageable on one or several NVIDIA GPUs but whose total expert weights exceed VRAM.

If a model fits completely in VRAM, a general GPU inference runtime may offer excellent performance with less host memory and less residency machinery. Krasis's distinctive value appears as the expert collection grows beyond available VRAM and long prompt processing remains important.

Likewise, Krasis does not remove PCIe from the system. Cold experts still cross it. The architecture attempts to amortize transfers during prefill, avoid them through exact residency during decode, and measure the remaining cost honestly.

The Ornith comparison should therefore be understood as an operating-point comparison, not as a claim that every model or GPU configuration will favour Krasis by the same ratio.

16. Limitations and threats to validity

16.1 Hardware coverage

The strongest current controlled performance table is concentrated on RTX 5090 and RTX PRO 6000 Blackwell systems with one EPYC host. The GLM peer, split-planner, repack, and release work adds an RTX A4500 Ampere card, but not a broad Ampere model survey. Community reports add independent operation on RTX 5000 Ada, RTX 5080, RTX 5090, and RTX PRO 6000 systems, including WSL2 and native Windows. These are valuable external evidence, but do not replace an exact-release controlled survey.

PCIe topology, copy-engine count, NUMA placement, driver version, host RAM, and GPU architecture can change the answer. The runtime measurement design is intended to adapt, but every performance ratio in this paper should be read with its hardware row.

16.2 Architecture-wide ablations cross revisions

The historical record contains compelling before/after evidence for HCS, Rust decode, graph replay, pinned DMA, attention kernels, and prefill streaming. Many of these changes were developed sequentially over different revisions. A clean publication should also run major architecture ablations on one pinned release and one fixed model/config so the contributions do not inherit unrelated code changes.

16.3 External comparison format

The Ornith llama.cpp comparison uses Q4_K_M with F16 KV, while Krasis uses its own INT4 expert cache with HQQ4/k4v4. The harnesses and prompts differ. The comparison is reproducible and useful as a complete-system result, but it is not a quantization-equivalent kernel comparison.

A strict publication comparison should rerun both systems adjacent in time, record exact prompts and power/clock conditions, and include additional llama.cpp settings if the implementation changes.

16.4 Quality coverage

QCN, Qwen3.5, Qwen3, Step, Qwen3.6, DeepSeek-V4, and GLM have independent llama-witness evidence at their stated contracts. Ornith and Gemma still use weaker local authority for important rows. Later Nemotron HQQ4 first-token witness rows pass, while HQQ6 remains blocked; neither is productive ADQ. GLM sparse HQQ4/INT4 is 7/8 teacher-forced argmax with 8/8 top-10 rather than a BF16-distribution match. Adaptive cold-mass pruning, CPU tails, and TileQ are therefore kept separate from exact production-quality claims.

16.5 Multi-GPU evidence

Current code supports N layer segments and independent per-GPU HCS pools. The new QCN INT4/HQQ mismatched RTX PRO 6000/A4500 split A/B is current-quality evidence and the GLM peer tier adds an exact alternative topology. Broader multi-model and N>2 evidence is still missing. Session-cache reuse remains single-GPU only and fails visibly as `multi_gpu_pending`.

16.6 Model support is not universal

Krasis supports several transformer, hybrid attention, Gated DeltaNet, Gemma, and Mamba2 surfaces. A new architecture can still require new attention, recurrent, routing, cache, or vision work. GLM-5.2's DSA/IndexShare attention, for example, is not made trivial merely by its MoE shape resembling supported models.

16.7 Startup and host-memory cost

Running a 397B model on one GPU still requires substantial system RAM and a prepared cache. Memory mapping, source-backed HCS, page shedding, and faster cache construction reduce the burden; they do not make it disappear.

Lossless expert compression adds another operational trade. The accepted GLM sidecar is roughly 322 GB and its full source binding, mapping, pinning, and verification adds about ten minutes beyond normal cache load on the measured host. A 277 GiB transposed CPU duplicate improved an experiment but is not a production memory design.

16.8 Sub-4-bit generality and kernel performance

TileQ has one complete Ornith artifact and one complete PPL result. Its builder currently assumes Qwen-style expert layouts and ordinary bias-free SwiGLU, and the scalar correctness kernels have not been benchmarked against Marlin. Extending the cache to every supported family is mostly shared descriptor and kernel work, but every model still needs its own calibration and acceptance. No claim is made that 3.164897 bpw will match Q4 on GLM, DeepSeek, Gemma, Nemotron, or other expert geometries.

16.9 Conversation-cache workload coverage

October correction: full-history/cached numerical and token counterexamples remain open. The pending repair has not completed ordinary, recovery, native Store or final GLM qualification. Earlier cache component passes and the August speed result do not establish end-to-end identity.

The principal speed result is one QCN interleaved two-conversation workload on one RTX PRO 6000. Exact active/RAM identity has broader family coverage, but the measured speedup depends on prompt length, snapshot size, state family, switch frequency, save cost, and host memory. Multi-GPU and image-state reuse are explicitly unsupported. The 1.132x continuation result should not be generalized to every agent workload.

16.10 Launcher capability coverage

The v1.0.21-rc.2 launcher and direct runtime were not equivalent. DeepSeek's validated BF16/BF16 path existed, but the launcher exposed only global HQQ and generic compressed-KV choices. A native-Windows user therefore had to invoke the private server runtime, and the same report showed 133.0 / 0 GB host RAM because pre-launch discovery read Linux /proc/meminfo. Those were genuine product-integration defects in the released candidate, not DeepSeek runtime failures.

Current source resolves that gap. DeepSeek exposes measured fixed HQQ4, HQQ6, HQQ8, budget-planned HQQ4+10% and HQQ6+10%, or BF16 attention and Native or BF16 cache, rejects unqualified multi-GPU modes before load, and derives its nonlinear cache budget from raw-ring, compressed, learned-index and FP32 recurrence state. Fresh launches default to HQQ6/Native. Native-Windows RAM discovery uses GlobalMemoryStatusEx. A real Linux launcher-started HQQ8/Native server passed 18/18 network cases, but a current-source native-Windows model run remains a platform acceptance gap rather than evidence inferred from a unit contract.

The live launcher gate itself found two additional bugs: valid expert DMA group 1 was silently rewritten to 2, and the summary displayed spare-VRAM context potential instead of the configured cache allocation. Both were fixed at the shared launcher contract and covered by saved/CLI/TUI tests before acceptance.

The resulting project policy is stricter than a menu fix: a model is not considered supported until the primary launcher exposes only capabilities the runtime actually implements, selects a validated default, fails before model loading on unsupported combinations, and passes a real launcher-started model gate on each supported host platform. Architecture-owned cache modes must use architecture-accurate names. DeepSeek-V4's packed QAT state is therefore called Native, not k4v4 or k6v6, because it is a shared latent/index state rather than conventional independent key and value tensors.

17. Future work

The next useful work follows directly from the evidence gaps rather than from a list of speculative features.

1. Run same-release ablations for the phase split, HCS, graph replay, HQQ residency, and pinned transfer path.
2. Extend the current mismatched-GPU INT4/HQQ split and peer evidence to more models, matched cards, and $N > 2$ topologies.
3. Calibrate the peer selector against live per-layer hits, bytes avoided, dispatch latency, deadline fallback, and exposed barrier wait rather than approved-heatmap averages.
4. Instrument session snapshot save assembly, copy volume, D2H overlap, and per-state contribution, then test whether save cost can fall without changing transactional publication or uncached speed. Implement multi-GPU session reuse only as a collective all-device contract.
5. Expand authoritative llama-witness coverage for Ormith, Gemma and the unqualified Nemotron precision modes, then reevaluate approximate modes and blocked quality rows.
6. Expand the same-card external comparison to additional model sizes and current llama.cpp revisions while preserving exact commands and raw logs.
7. Continue instrumentation-driven Nemotron work at the SSD dataflow boundary, where current evidence places the real remaining bottleneck.
8. Evaluate new cold-transfer or compression designs only if they address the complete HCS, peer, pruning, codec, and startup lifecycle demonstrated by the GLM matrix. Grouped lossless decode remains the measured control.
9. Keep TileQ paused at the current evidence boundary. If resumed, improve the residual or build a quality-constrained mixed 3/4-bit map before broad model rollout or kernel tuning; preserve actual-bpw and source-bound artifact accounting. NVFP4 is not a current Krasis work item.
10. Extend hardware coverage to Ampere, smaller VRAM cards, and additional PCIe and NUMA configurations.
11. Apply the adopted capability-derived launcher policy to every existing and future model family, and add a current-source native-Windows DeepSeek model run to the platform evidence. Unit coverage now protects Windows RAM discovery, but it is not a substitute for the real installer/launcher/model lifecycle.

18. Conclusion

Updated conclusion, 19 October: phase-separated execution and measured residency are supported by the dated evidence below. Their usefulness also depends on retained task capability. ADQ now makes task, numerical, determinism, behavioural and safety verdicts explicit. The GLM 2/4 task failure and unresolved conversation equivalence are material limitations; the current repair is not qualified. The following architectural conclusion retains its August scope.

Very large MoE inference on limited VRAM is not one problem. Prompt processing needs batch throughput. Token generation needs low synchronization and useful expert residency. Memory planning needs to reflect the exact model, backend, prompt, allocator, and GPUs that are actually running.

Krasis separates those concerns. Prefill streams bounded layer groups through GPU kernels and reuses their weights across the prompt. Decode stays in Rust/CUDA and treats VRAM as an exact expert cache. HCS begins from an exact-configuration route prior, adapts within the request, and borrows memory between phases. Runtime calibration decides how far residency can safely expand.

Adaptive cold-mass pruning shows that the same residency information can also support an explicit approximate mode. It removes only low-ranked routes whose experts are cold, because those are the cases where approximation can eliminate a complete transfer. The Ormith result is promising, but retaining exact mode as the default until independent quality evidence exists is part of the result, not an administrative qualification.

The current results show that this approach can process prompts for 201B–397B models at thousands of tokens per second on one 96 GB GPU and can generate from models that remain far larger than VRAM. The Ornith-397B comparison is the clearest example: the best fitting llama.cpp configuration found still required substantial CPU MoE execution and reached 185.01 prompt tokens per second at 8,192 tokens and 10.71 generated tokens per second, while Krasis recorded 2,354.5 prefill and 23.58 internal decode tokens per second on the same card.

The newer GLM result shows both the value and the limit of continuing this design. Once real CUDA–event transfer time was measured, exact peer service and lossless GPU decoding could remove additional cold work, and explicit pruning could remove a bounded approximate tail. Together they improved decode 25.7%. They did not reach 2x because the measured peer VRAM and codec ratio could not cover enough of the remaining cold population. Dynamic replacement and deeper pipelining did not manufacture capacity that was not present.

The DeepSeek-V4 and GLM bring-ups show that phase separation does not require a single attention implementation. Model-specific hyper-connection, compressed-indexed attention, MLA, DSA, and IndexShare kernels can live under the same full-GPU-prefill, Rust-decode, HCS, and runtime-calibration contracts. The session cache extends the same principle across requests: exact state may move between GPU and RAM, but its compatibility and publication boundaries are measured and fail closed.

TileQ adds a final boundary. Reducing expert bytes can improve RAM, residency, and cold transfer simultaneously, and 3.164897 bpw came within 0.97% PPL of the accepted Q4. That is enough to justify preserving the result and format, but not enough to rewrite the predeclared acceptance gate. Pausing there is an engineering result: the next work is quality, not pretending scalar kernels are production speed.

The result did not come from accepting every plausible optimisation. Many ideas were rejected because they improved the wrong metric, predicted work without an execution window, increased cache churn, weakened quality, or relied on a misleading timer. That experimental record is as important as the accepted path. It shows that the main contribution is not a collection of isolated fast kernels. It is a system that makes the phases, memory classes, evidence, and failure modes explicit.

Appendix A. Accepted architectural decisions and optimisations

This appendix provides a compact map of every accepted, current experimental, or historically useful contribution in the evidence inventory. The numbered entries correspond to the detailed successful-decision ledger (internal record). These identifiers are stable evidence keys, not the appendix's display order; entries remain grouped by contribution when later work extends an older theme.

A.1 System architecture and execution strategy

S01 – Separate prefill and decode execution. Prefill uses batch-oriented GPU streaming; decode uses native HCS-aware token execution. This is the central architectural decision and the basis of the current 26B–397B benchmark range.

S02 – Keep prefill on GPU. The model may be streamed, but prompt computation does not fall back to CPU. QCN reaches 7,028.5 prompt tokens per second on the 5090 and Ornith-397B 2,354.5 on the RTX PRO 6000.

S03 – Own the decode hot path in Rust/CUDA. Per-token model execution, sampling, tokenization, detokenization, and serving avoid Python/GIL re-entry. The superseded Python callback path measured 6.5 tokens per second.

S04 – Use exact HCS residency as the primary decode strategy. Resident experts execute from VRAM; misses use demand DMA. The first controlled HCS test improved MoE throughput by 24% at 41% expert residency.

S05 – Divide HCS into resident and reclaimable tiers. Prefill can borrow a measured portion of decode residency and decode can restore it afterward. This enables aggressive residency without making long prompts unsafe.

S06 – Adapt HCS during a request. A bounded recency tail promotes experts actually used by the current request while protecting the exact global prior.

S07 – Support source-backed and mirror-backed HCS. Machines with enough RAM can optimize reloads; very large models can avoid a complete duplicate host copy. Expert precision is unchanged.

S08 – Seed residency with exact-configuration heatmaps. Heatmaps are built with matching model and quantization parameters, versioned, and kept separate from benchmark prompts. A Qwen3.5-122B approved heatmap improved decode about 6.3% over its quick prior.

S09 – Keep adaptive cold-mass pruning explicit and default-off. Ormith 75/8 improved internal decode roughly 9–10% while removing 23.87% of cold fetches. It remains experimental pending an independent witness.

S10 – Stream prefill by layer group. A bounded set of expert layers is loaded, reused across all prompt chunks, and replaced by the next group. This is the mechanism behind the 397B and 480B single-card prefill paths.

S11 – Keep serving and tool-use streaming native. Rust owns chat-template application, sampling, detokenization, JSON/SSE formatting, tool-call handling, and disconnect propagation during decode.

S12 – Pipeline contiguous layer segments across GPUs. Each GPU owns an independently budgeted HCS pool and native graph set. Historical 2×5090 evidence raised QCN HCS from 68.6% to 100% and decode from 94.0 to 107.5 tokens per second.

A.2 Runtime, memory, and kernel optimisations

S13 – Use a Rust-owned custom CUDA kernel stack. Norm, route, activation, attention, expert reduction, and utilities launch directly from Rust with explicit stream/event control.

S14 – Segment CUDA graphs with mutable pointer indirection. Stable launch regions are captured while changing expert addresses arrive through fixed tables. This raised Gemma k6v6 decode from 38.79 to 63.69 tokens per second and enabled the accepted Mamba2 graph path.

S15 – Use Marlin-packed INT4 as the canonical routed-expert format. One versioned cache serves GPU prefill and decode without runtime quantization or a Python conversion loop.

S16 – Batch routed expert GEMVs and fuse profitable activation/down work. Early work reduced 30 launches to four per layer and moved adapted QCN decode from approximately 50–51 to 60–62 tokens per second.

S17 – Use flat O(1) expert lookup and a GPU pointer table. Token-critical residency lookup avoids hashes, objects, and per-expert host traversal.

S18 – Store experts in per-layer contiguous pinned backing. QCN avoids about 40 GB of duplicate RAM and reduces host registrations from roughly 24,576 to 192; larger models avoid still more duplication.

S19 – Pin the production backing before measuring it. The corrected RTX PRO 6000 probe measured 25.91 GB/s rather than the misleading 13.54 GB/s pre-pin value.

S20 – Pipeline demand DMA with events and multiple staging regions. The early integrated path improved from 3.69 to 1.87 milliseconds per layer and from 5.6 to 11.1 tokens per second.

S21 – Use double-buffered stream-ordered prefill staging. Ping-pong buffers, events, and allocator lifetime annotations overlap safe work without allowing in-flight storage to be recycled.

S22 – Size prompt chunks from runtime measurement and model dimensions. The planner adapts to prompt length, backend, model geometry, and current free VRAM, with bounded retry on an unexpected allocation failure.

S23 – Release redundant tensors after native registration. When Rust owns the production copy and a mode cannot use the Python source tensor, the duplicate is released while diagnostic modes retain what they need.

S24 – Materialize HQQ attention into native prefill/decode representations. Versioned HQQ caches remove repeated projection/dequantization work and leave more VRAM for HCS.

S25 – Offer measured HQQ4/HQQ6/HQQ8 profiles. Precision is explicit and validated per model instead of being assumed globally. HQQ4 often favours residency; HQQ6 often favours BF16 proximity.

S26 – Vendor FA2 and FLA as explicit native sidecars. Known kernels are built, verified, and loaded through native interfaces rather than Python JIT imports during serving.

S27 – Select sliding or full attention per layer geometry. Step sliding FA2 improved long prefill by 41.3%; guarded Gemma HD512 paths produced still larger model-specific gains.

S28 – Quantize Nemotron Mamba2 projections separately from routed experts. Projection residency fell from 8,360.0 to 2,220.6 MB and decode improved from 28.94 to 41.96 tokens per second while HCS expanded.

S29 – Parallelize Nemotron SSD prefill behind shape and correctness gates. The accepted policy moved Super from 192.9 to 1,228.7 prompt tokens per second at its initial gate; later work produced the current 1,852.2 result.

S30 – Memory-map versioned caches and shed unneeded page cache. Large-model startup avoids an unnecessary combination of application copies and retained OS pages.

S31 – Ship native kernels as reusable release sidecars. Linux and Windows packages reuse verified Marlin, attention, and generated-kernel artifacts without moving serving back into Python.

S32 – Tile long-context decode attention across the GPU. Historical QCN development evidence improved 18K decode from 6.1 to about 23 tokens per second and extended useful execution to 97K context.

S33 – Prefetch a known next dense MoE layer during prefill. Nemotron Super prefill improved 14.4% and exposed MoE DMA wait fell from 1,099.9 to 28.6 milliseconds on the representative prompt.

S34 – Build HQQ4 caches with versioned GPU search. QCN projection searches improved 9.75×–17.70× with near-identical RMSE; a complete cache built in 32 seconds and passed its witness.

S35 – Offer pinned HQQ stage backing for high-RAM systems. The stage copy fell from 151.381 to 112.684 milliseconds and timing-off prefill improved 12.1%; the option remains explicit because it pins approximately 1.9 GB.

S45 – Select heterogeneous splits from measured service cost. QCN on RTX PRO 6000+A4500 selected [46, 2], retained full residency, and improved timing-disabled decode 11.82% over the capacity split.

S46 – Serve a disjoint exact peer expert tier. GLM's A4500 tier passed bit-exact witness/fallback gates and improved peer-only decode 6.7%; live hit coverage was materially lower than the heatmap prediction.

S47 – Decode losslessly compressed cold experts on GPU. The accepted GLM lane-256 sidecar saves 13.7389% and reconstructs bit-exactly at lower measured p95 than raw transfer. Startup and composition costs keep it explicit.

S48 – Transactional conversation-state reuse (end-to-end equality open). Active GPU and pageable-RAM prefix checkpoints improved QCN continuation time 1.132x in the fixed interleaved workload while preserving response identity and uncached speed in that historical workload. October full-versus-resumed counterexamples remain unresolved; this inventory entry is not a current correctness pass. Multi-GPU reuse remains pending.

S49 – Derive tool parsing from the loaded template grammar. Rust supports DeepSeek DSML, Qwen JSON/XML, GLM argument XML, Gemma, and MiniMax contracts; malformed output stays visible and unsupported grammars fail closed.

S50 – Tune Ornith from production graph attribution. Retained bit-exact HQQ, route-prep, top-k, and shared-expert changes improved adjacent 50/100/250-token decode roughly 10-11% with identical full-corpus PPL.

S51 – Implement DeepSeek-V4's compressed/indexed attention as its own architecture. The final runtime passes witness/full-PPL gates and reaches 1,204.3 prefill tok/s at 62K plus 19.41 decode tok/s at that context.

S52 – Implement native GLM MLA/DSA and IndexShare owner resources. Exact config-derived sparse state preserves full-GPU prefill and exposes cold expert traffic as the dominant remaining decode interval.

S53 – Treat sub-4-bit methods as source-bound measured backends. TileQ's 3.164897-bpw artifact proves storage, native dispatch, and quality accounting without silently reusing INT4; it remains experimental after missing Q4 PPL by 0.97%.

S54 – Ship verified native runtime artifacts. Linux wheels and the atomic private Windows runtime bind the measured native libraries and Python ABI to the release rather than mutable host state.

S55 – Quantize DeepSeek-V4 attention while preserving checkpoint-native cache state. HQQ8 matches the accepted quality result; Native nearly doubles same-budget context capacity without a second quantizer and remains explicit because short prefill regresses.

S56 – Treat the launcher as a support acceptance surface. Capability-derived choices and a real launcher-started 18/18 network gate caught invalid menu, platform RAM, layer-group and cache-capacity contracts before promotion.

A.3 Measurement, calibration, and validation

S36 – Calibrate VRAM with real short and long workloads. The loaded model/config/backend is measured before residency is selected.

S37 – Express phase requirements as transient deltas. A delta remains useful after allocator state changes; a stale absolute low-water does not.

S38 – Monitor driver-visible low-water VRAM. Native monitoring includes runtime and sidecar allocations that framework counters can omit or delay.

S39 – Separate instrumented diagnosis from clean speed runs. Timing and counters explain a result; timing-off built commands establish the result.

S40 – Keep benchmark and server execution paths aligned. Prefill, HCS, decode, and cleanup are shared, while internal and HTTP measurements remain separately labelled.

S41 – Use canonical real prompts and preserve complete logs. Exact tokenization, distinct heatmap data, immutable literary prompts, tmux, and raw artifacts support reproducible long-context evidence.

S42 – Combine llama-witness, PPL, and real requests. First-token or local tensor agreement alone is not a production quality gate.

S43 – Enforce fail-closed artifact and backend contracts. A mismatched heatmap, reference, native backend, or optional mode cannot silently substitute a different execution surface.

S44 – Locate the first divergence with trace-off-neutral diagnostics. Structured stored traces narrow errors to a layer/component while leaving the normal speed path unchanged.

A.4 Historically successful but superseded techniques

HS01 – CPU-specific expert layouts. Row-major and tiled layouts improved the old CPU decode path by 2.8× and 24% in their respective tests. Native GPU HCS decode later removed the production need for this second execution copy.

HS02 – FP16 CPU KV cache. Replacing FP8 LUT gathers improved the old QCN CPU decode engine from 12.70 to 14.45 tokens per second. The GPU decode/KV design superseded it.

HS03 – A simple sequential INT4 attention GEMV at M=1. In the AWQ era this improved QCN decode from approximately 66 to 78.4 tokens per second. AWQ is now deprecated, but the different-layout lesson for batch GEMM and single-row GEMV remains valid.

HS04 – Resident BF16 attention. This established the GPU-max decode direction by removing per-layer attention streaming. HQQ later reduced the fixed residency and returned more VRAM to HCS.

Appendix B. Negative results

This appendix records the complete set of rejected, reverted, superseded, blocked, or default-off hypotheses in the negative-result ledger (internal record). Ordinary bugs and baseline programming discipline are intentionally excluded. The identifiers are stable evidence keys rather than display-order numbers, so recent results remain beside the experiment family they qualify.

B.1 Rejected system architecture and execution strategies

F01 – One CPU/offload strategy for prefill and decode. This could not provide the long-prompt behaviour Krasis required. The best fitting Ornith-397B llama.cpp result reached 185.01 prompt tokens per second at 8,192 tokens and 10.71 generated tokens per second, compared with Krasis's recorded 2,354.5 and 23.58 on the same GPU.

F02 – A Python callback for every decoded token. The path measured 6.5 tokens per second, selected the wrong M=1 kernel surface, and repeated GIL/FFI work. It was superseded by the native Rust/CUDA loop.

F03 – CPU routing followed by selected-expert-only GPU decode. QCN measured 3.24 milliseconds per layer for the hybrid against 2.0 milliseconds for CPU only; Qwen3-235B gained only about 3% before system overhead. Serial routing-transfer-compute did not create a useful GPU path without residency.

F04 – Interleave whole CPU and GPU layers as the main decode architecture. Measured PCIe/CPU ratios implied only approximately 5–12.5% expected gains and converged on pinning a few whole layers. Fine-grained expert residency had a larger useful range.

F05 – APFLspeculative expert prefetch. Prefetch=10 reached 96% prediction hit and regressed 11% in an early test; a later Step test regressed decode from 21.96 to 14.40 tokens per second. One copy engine and wrong transfers erased the predicted benefit.

F06 – Previous-token route prefetch. QCN reached 86.61 tokens per second against an 88.86 timing control. Ornith's corrected probe could not admit its 10,925 candidates within budget, and the Step matrix did not beat baseline.

F07 – Prompt-conditioned HCS reload as a universal default. QCN retain sweeps were noisy, fully resident 35B was unchanged, and route simulations favoured a global prior plus recency, especially on Qwen3-235B.

F08 – Unbounded HCS promotion. Qwen3-235B's hit rate improved while token speed collapsed to 2.7–2.8 tokens per second from promotion churn. Promotion must be bounded and evaluated on total time.

F09 – Approximate cold-expert swaps. Qwen3.5-122B improved from 25.29 to 27.30 tokens per second but weakened exact/containment quality; QCN was slower than exact controls. The model-dependent policy was not accepted.

F10 – Repeat-protect victim selection without admission gating. Step made 71.61 protected-low-value decisions per token and followed a 416.10 MiB/token, 16.40 millisecond/token worse trajectory. The comparator changed its own future cache state in an adverse way.

F11 – Learned expert prefetch from route history. Offline predictors did not produce a production transfer window after false positives, precision, copy budget, and serialized H2D were included. Runtime APFL evidence independently regressed.

F12 – Preferred-friends and neural request-level HCS planners. Offline rankers did not recover enough oracle value robustly across prompts and models to displace simple exact recency.

F13 – MTP route prediction as a prefetch oracle. Root-guided top-2 improved effective acceptance from 2.223 to 2.600 tokens but clean decode was only 12.97 tokens per second; verifier and second-draft cost remained dominant.

F14 — Standard small-draft speculative decoding. Recurrent-state and KV snapshot/rollback, low acceptance, and expensive target verification prevented a clean gain on the measured large MoE targets.

F15 — Best-of speculation over a hot expert subset. The Qwen3.5-122B hot subset was too different from the target; misses and verification erased the draft advantage.

F16 — A superquantized same-model draft on a 16 GB auxiliary GPU. A hypothetical INT2 Qwen3-235B draft could hold only about 24% of experts and was projected near 4.8 tokens per second, far below the approximately 22.5 tokens per second needed even under favourable acceptance.

F17 — DFlash as the Qwen3.5-122B production verifier. Default DFlash reached 16.25 tokens per second against a non-DFlash baseline around 28.83. Relaxing the threshold increased acceptance but verifier latency reached 298–579 milliseconds per round.

F18 — Nested INT2/INT3 cold-expert compression. Exact promotion fetched the omitted planes and canceled the saving. Approximate promotion destabilized HCS and reached 17.38 tokens per second; preventing promotion reduced hit rate and reached 14.7. The complete runtime experiment was rolled back.

F19 — Prefill expert pinning from an inaccurate prescan. The prescan did not have route-accurate hidden states. After corrections, default no-pinning matched long-prompt performance; the flag remains an evaluation gate only.

F20 — Nemotron mirror host cache. Prefill remained approximately 1,162.7 tokens per second and decode regressed from 28.29 to 11.76. Source backing was retained.

F21 — Python-dispatched split-head tensor parallel attention. The candidate measured 472–473 prompt tokens per second against 476 baseline. Roughly 710 small host-launched kernels per GPU serialized the devices, so the design was removed.

F22 — Replace cold expert DMA with CPU expert execution. The integrated Step path reached 85.15 milliseconds per token, with 56.64 milliseconds in CPU cold compute, and missed one of eight witness first tokens. Forced across-expert scheduling was worse.

F23 — Rebalance HCS between prompts using a sliding history. A 100-prompt test showed flat decode performance while the policy could replace 25% of the approved global prior per prompt. It was removed.

F24 — Reallocate a fixed HCS slot budget across layers from static scores. Step's measured-layer policy reached 47.16 milliseconds per token against 47.08 for control. It changed counters without improving token time.

F25 — Treat Gemma ring-window KV as production long context. k4v4 produced invalid 25K output. k6v6 became correct through 25K and reached 498.2 prefill / 28.4 decode tokens per second, but 100K timed out and memory-saving variants were too slow. The mechanism remains blocked, not accepted.

F70 — Promote GLM 75/10 pruning by speed alone. Decode improved about 17%, but later selected-token log probabilities moved by up to 0.414816 versus exact. The mode remains explicit pending broader quality evidence.

F71 — Make CPU-tail execution the normal cold path. One worker proved the double-ended scheduling idea but added only a few percent and used approximate INT16 activations. It did not approach the 10 tok/s target.

F72 — Keep a 277 GiB duplicate transposed CPU expert tier. It increased CPU tail wins and added 3–4%, but the model-sized duplicate and H2D contention make it an architecture probe rather than a production cache.

F73 — Add a second CPU-tail worker. Worker two won only 1.48% of claims, reduced H2D bandwidth, and regressed all adjacent decode rows. Further workers were not implemented.

F74 — Equate heatmap peer capture with live peer service. The selector predicted a near-2x margin while peer-only delivered +6.7%; live witness peer hits were about half the heatmap estimate.

F75 — Add peer and compression savings independently. They compete for the same cold routes. The full composition remained positive but materially sub-additive.

B.2 Negative runtime, memory, and kernel experiments

F26 – Use Marlin’s GPU-tiled expert layout for CPU decode. Kimi measured 0.55 tokens per second from the Marlin layout and 1.55 from a sequential CPU layout. Tile permutation and cache locality made the shared format unsuitable for the old CPU path.

F27 – Split or reorder hot and cold expert launches. A cold-first pass stayed within noise at 33.4–33.8 tokens per second. A later split-launch QCN path fell to 78.21 from an 88.86 timing control, and Step showed no timing-off win.

F28 – Runtime Marlin W13 k-split autotuning. QCN selected ksplit=1 over the formula’s 2 but reached only 81.94 tokens per second against the 88.86 control. A later robust gate did not change the production conclusion.

F29 – GPU route synchronization and classification. Qwen3-235B route classification was only about 0.15 milliseconds per token while total sync was 211.8 milliseconds. Step’s forced GPU route sync measured 47.39 milliseconds per token against 46.56 control.

F30 – A top-k=10 weighted-accumulation specialization. QCN fell to 78.30 tokens per second from the surrounding 79–80 surface. Removing generic branches did not overcome scheduling and register effects.

F31 – All-row and hybrid grouped INT4 expert kernels. Exact grouped variants measured 124.492 and 109.206 milliseconds against 97.234 for pair-parallel execution. The first multi-row architecture also produced 304–337 millisecond cold loops.

F32 – CUDA graphs for the DFlash verifier. Graphing was slower because expert and cold-transfer work dominated; scratch save and restore was only 0.26–0.39 milliseconds per round.

F33 – Lossless LZ4 compression. A 300-expert stratified Ornith sample retained 97.97% of its original packed size. A 2.03% saving could not justify a runtime decoder.

F34 – CPU Zstandard decompression. Zstd saved 18.94% but the measured residual traffic required more than approximately 111 GiB/s decompression to break even, before the added serialized host stage.

F35 – Plane-major compressed expert backing. A cold expert required eight DMA calls instead of four. Despite fewer bytes, all-cold calibration slowed from 181.45 to 234.30 milliseconds per token.

F36 – Reconstruct every expert from globally bit-sliced backing. Experts without low-bit assignments paid reconstruction cost for no transfer saving; the path remained around 221.5–222.1 milliseconds per token against 181.45 exact.

F37 – Triple-buffered prefill streaming. The third buffer did not improve throughput and increased VRAM and allocator pressure. The two-buffer design was restored.

F38 – Replace broad prefill synchronization with narrow event waits. On an RTX 5080, prefill regressed from approximately 3,500 to 1,500 tokens per second because allocator-pending blocks were not reclaimed as expected. Restoring the broader synchronization recovered approximately 3,100 at longer context.

F39 – Skip allocator cleanup after a prefill group. In-flight Marlin kernels consumed memory that the allocator reused. QCN WikiText perplexity exploded from approximately 4.55 to 2,169, so the change was reverted for correctness.

F40 – Reuse request-scoped prefill pointer tables. The first attempt weakened first-token witness to 6/8 because allocation had accidentally provided ordering. Explicit events restored correctness, but Step prefill still regressed 5.4%, from 1,618.2 to 1,531.3 tokens per second.

F41 – Asynchronous HQQ stage copy. The candidate passed witness but clean QCN speed regressed to 6,656.2 prefill, 88.68 decode, and 143.83 HTTP tokens per second. Lower isolated copy latency did not improve the request.

F42 – cuDNN SDPA as a faster FA2 replacement. Output was close, but QCN prefill reached 5,246.6 tokens per second against 6,191.3 for the nearest FA2 comparison, approximately 15.3% slower.

F43 – FlashQLA on the RTX 5090. Upstream kernels required sm90 and more shared memory than the sm120 card allowed. A smaller tile failed its layout assumptions. A backend that cannot launch was not wired as a fallback.

F44 — Native sm120/extra-architecture FA2 build. The path failed with “no kernel image is available”, a compute_120 PTX fallback did not fix it. Krasis restored the verified compatible build and propagated launch errors.

F45 — FLA recurrence with BV=64. Qwen3.5-35B HQQ8 prefill fell from about 8,103.7 to 5,416.7 tokens per second as state time rose to 2,708.9 milliseconds.

F46 — Non-HD512 tiled k4v4 decode for Step. Decode improved about 4%, from 21.96 to 22.89 tokens per second, but first-token witness fell from 8/8 to 7/8 and top-k diagnostics weakened. The speed signal did not pass correctness.

F47 — INT8 attention as a generic compression mode. QCN perplexity was approximately 5,630 instead of 4.52–4.55 and output was unusable across models. The mode was removed.

F48 — AWQ and FP8 as current production quality paths. Historical speed gains did not satisfy later witness/PPL confidence, and FP8 KV/attention had cross-model quality concerns. Current accepted results use HQQ and INT4 experts.

F49 — HQQ self-calibration sidecars and INT8 exception groups as a default repair. Many staged searches, correction scales, and exception sizes failed held-out/witness acceptance or added cost without a stable cross-model win. Explicit experimental infrastructure remains; the policy was not promoted.

F50 — Reusable Nemotron SSD scratch. The candidate regressed clean prefill from the accepted 1,464.5 path to 1,411.0 tokens per second and reduced prompt capacity in related trials.

F51 — Nemotron SSD and decode shortcut microkernels. Local-dA failed layer-0 correctness; shared-C passed bitwise self-compare and regressed prefill to 1,683.5; state-split variants fell to 1,259.3 or 1,056.7. Later fusions and shortcuts were slower or changed output.

F52 — Read cold weights directly through mapped host memory. QCN mapped reads reached 57.54 tokens per second, approximately 30% below the contemporary DMA control; registering the 98K+ regions with DEVICEMAP reached 54.05. BAR/TLB and non-coalesced PCIe reads stalled compute that explicit bulk DMA avoided.

F53 — Redesign the cache for one copy call per cold expert. The corrected production-shaped measurement reached 25.91 GiB/s for four copies and 26.43 for one contiguous copy, only about 2%. The small gain did not justify breaking the shared layout or adding about 110 GB of host duplication.

F54 — Deduplicate or cluster experts. All 24,576 QCN experts were scanned. There were no exact weight duplicates and no duplicate scale vectors; cosine similarity around 0.918 was too low for safe lossless aliasing.

F55 — Fill prefill layers selectively from HCS. The Python variant issued 976 copies per layer and the Rust variant 2,048 CUDA calls per layer. Both were slower than four large bulk transfers despite moving fewer bytes.

F56 — Compact HQQ slots at every stage transition. Forced QCN HQQ6 compaction produced 14,013 resident experts against 14,256 for control and also worsened prefill. Pointer refresh and graph invalidation consumed the local memory saving.

F57 — Compile an entire recurrent method as one graph. Inductor unrolled 157 iterations into an enormous graph, took approximately 180 seconds to compile, and regressed prefill from 856 to 449 tokens per second.

F58 — Force CPU locality with mmap consolidation, thread pinning, or software prefetch. Contiguous mmap slowed real decode 15%; sequential pinning lost 30%, CCD-spread pinning 266%, and cross-layer prefetch 3.5%. Synthetic locality gains did not carry to the fragmented real workload.

F59 — Offload the LM head on very low-VRAM GPUs. Zephyrus warmup increased from 892.3 to 1,001.8 seconds and soft-HCS safety evictions appeared during prefill. Releasing the redundant source tensor recovered 485 MB without recurring movement.

F60 — Select routed-expert INT4 parameters using static weight RMSE. The objective did not represent activation-weighted expert output. Limited groups improved under activation-aware scoring, but no accepted whole-cache witness established static RMSE as the production policy.

F61 — Optimize Gemma through isolated narrow fusions and staging kernels. A large family of route, W2, QKV, dense, reducer, score, cache, and KV conversion candidates passed narrow gates but were flat, slower, or impractical. The accepted gains came from graph boundaries and attention geometry.

F62 — Remove occupancy-enhancing work merely to save launches. Skipping W13 k-split made HCS experts 7% slower and cold experts 5% slower. A one-block fused gate/top-k kernel was 2.2× slower than distributed cuBLAS execution.

F76 — Use the higher-ratio lane-1024 GPU codec. It reconstructed exactly but took roughly 914 microseconds p95 versus about 748 raw and reduced live decode. Lane-256 was accepted on latency, not ratio.

F77 — Deepen entropy pipelining with a persistent streaming decoder. The bit-exact eight-chunk plan measured 702.01 microseconds p95 versus 682.55 for grouped, and the combined live row was slower. Grouped remains default.

F78 — Claim a dynamic-peer speedup from a flat result. Dynamic measured 5.10 tok/s versus static 5.07 and lacked published swap counts. The transactional implementation remains, but no causal performance claim exists.

F79 — Make CPU-transposed INT4 the universal cache via GPU repack. The repacker was byte-exact and fast in isolation, yet QCN prefill/decode regressed 2.5–3.5% and its scratch displaced HCS.

F80 — Optimize Ornith activation/W2 from overlapping block timings. Two bit-exact candidates were flat or slower; the apparent hotspot was not serialized token headroom.

F81 — Promote DeepSeek's fastest gathered-GEMM prefill. It delivered large speed gains and passed four witness prompts, but its +0.66% fixed-screen PPL change violated the declared gate and the code was removed.

F82 — Promote TileQ v2 as the Q4 replacement. It is 23.42% smaller and only 0.97% worse than Q4 PPL, but the experiment declared no-worse-than-Q4 and the scalar kernels have no accepted speed result.

F83 — Add hot resident compression or a static warm INT2/INT3 tier from planning arithmetic. Measured decoder and route economics predict more critical-path work than cold saving. These ideas were rejected before live runtime promotion, not reported as benchmarks.

B.3 Invalidated measurement, calibration, and validation methods

F63 — Use an absolute calibration low-water as a later budget. Allocator caches made current free VRAM about one gigabyte lower on Qwen3.5-122B and caused decode OOM. Requirements are now stored as transient deltas and applied to current free memory.

F64 — Measure H2D bandwidth before pinning expert memory. The early probe reported 10.9–13.5 GB/s while real pinned transfers reached about 20.5–21.7 GB/s and the corrected probe 25.91 GB/s. Production registration and copy shape must be part of the measurement.

F65 — Interpret enqueue time as transfer completion. Qwen3-235B printed approximately 4.4 milliseconds of host enqueue while 211.8 milliseconds of completion appeared in later synchronization. CUDA events and bytes, not host enqueue duration, describe DMA cost.

F66 — Query free VRAM every token and include post-processing in decode time. Per-token driver calls and vector clones perturbed the loop, while tool parsing extended the timer. Historical 80+ QCN rows could not be reproduced from the committed logs and the measurement boundary was corrected.

F67 — Use repeated synthetic paragraphs and character/token guesses. Five paragraphs were repeated dozens of times and token counts were estimated. The generators were removed in favour of immutable real text and exact tokenization.

F68 — Treat Hugging Face Transformers output as the authoritative witness. Qwen3.5 artifacts did not provide stable, finite, internally consistent diagnostics. HF remains forensic evidence; llama-witness is the active external authority.

F69 — Treat Krasis's own BF16/debug path as an independent oracle. A QCN BF16 fused expert diagnostic returned zero W1 rows, and other local paths agreed with themselves while differing from external witness. Local self-compare is a debugging

tool, not acceptance proof.

F84 – Benchmark conversation reuse through debug logits. The first cache benchmark did not use the normal response path and was discarded. The accepted 1.132x/1.067x result includes production save, restore, decode, HTTP, and response handling.

F85 – Use one cross-height full-prefill output as the cache oracle. Marlin projection bytes can depend on prompt height, and Step needed more controls to expose existing variation. Same-boundary active/RAM identity remains hard; cross-height comparison uses an uncached envelope.

F86 – Publish reservation/LRU changes before response commit. A cancelled candidate could evict committed state. Reservations are now non-mutating and all snapshot/eviction effects publish transactionally on success.

Appendix C. Reproduction and source guide

Public repository links below are pinned to the reviewed source revision. Items labelled “internal record” are retained provenance descriptions, not public downloads. The new ADQ findings are company-reported summaries with the limits stated in the research update; external reproduction of those campaigns is not provided by this edition.

C.1 Current result indexes

- [Krasis benchmark statistics](#)
- [Krasis quality statistics](#)
- [Complete benchmark log index](#)
- [Approved route heatmaps](#)

C.2 Architecture and methodology records

- GPU prefill pipeline (internal record)
- Rust/CUDA decode loop (internal record)
- HCS expert caching (internal record)
- VRAM budgeting and safety (internal record)
- Multi-GPU execution (internal record)
- Mamba2/Nemotron support (internal record)
- GLM-5.2 MLA/DSA and CPU-tail record (internal record)
- DeepSeek-V4 compressed/indexed attention record (internal record)
- Native TileQ-S expert format (internal record)
- Weight quantization history (internal record)
- Rust server, templates, SSE, and tool grammars (internal record)
- Native build and Windows runtime lifecycle (internal record)
- Benchmark contract (internal record)
- Test prompt methodology (internal record)
- Reference testing (internal record)
- Structured trace debugging (internal record)

- Model acceptance process (internal record)

C.3 White-paper claim ledgers

- Accepted decisions and optimisations (internal record)
- Failed, rejected, reverted, and superseded decisions (internal record)
- Internal chronological changelog (internal record)

C.4 Ornith-397B comparison artifacts

- [llama.cpp full-run log](#)
- [llama.cpp OOM-gated GPU-offload runner](#)
- [Krasis benchmark summary](#)

C.5 Adaptive cold-mass pruning artifacts

- [Exact Ornith control](#)
- Shadow policy sweep (internal record)
- 75/5 full test (internal record)
- 75/8 full test (internal record)
- QCN exact control (internal record)
- QCN explicit 75/8 guard (internal record)
- [Rust planner and statistics](#)

C.6 Recent cold-bandwidth, state-cache, and quantization artifacts

- [GLM peer/compression matrix](#)
- [GLM dynamic-peer/streaming follow-up](#)
- [GLM first sparse benchmark and CPU-tail programme](#)
- [DeepSeek-V4 evidence series](#)
- [RAM-backed interleaved agent benchmark](#)
- [Ornith TileQ-S quality gate](#)

C.7 Community evidence

- Community results, canonical Reddit thread register, and DeepSeek-V4 GitHub report (internal record)

C.8 Built commands

Current evidence is produced through the repository's built command surface:

```
./dev speed-test
./dev benchmark <config>
./dev benchmark <config> --timing
./dev reference-test <config>
./dev release-test <model>
./dev test <config>
./dev test <config> --thorough
./dev perplexity <config>
./dev trace-diff <expected-trace> <actual-trace>
./dev session-cache-benchmark <port>
./dev tileq-build <subcommand> ...
```

Timing-enabled runs are diagnostic and are not used as clean speed results. Every publication row should retain the exact config, prompt, hardware, driver, HCS coverage, minimum free VRAM, quantization profile, and raw log.

Appendix D. Change history

Version	Date	Material changes
0.5	19 October 2026	First public living edition; Northloom Ltd authorship and human-curation statement; major ADQ rationale, protocol and split-verdict findings; GLM 2/4 failure; DeepSeek arithmetic and precision studies; GLM scales-only ablation; conversation-cache claim corrections and unresolved gates; memory/progress lessons; refreshed evidence boundaries and Nemotron witness status.
0.4	10 August 2026	Internal evidence edition covering phase-separated architecture, measured expert residency, runtime calibration, performance and quality evidence, conversation-cache benchmarks, TileQ and negative results. Not previously established as a public release.

The latest edition is maintained on Krasis.ai. This appendix records substantive changes; it does not offer every previous edition as a separate public download.