1.3 Latency, Throughput, and Hardware-Aware Patterns

The architecture and constraints from Chapters 1.1 and 1.2 set the ceiling. This section is about measuring it: what does a real model's latency profile look like on Intel hardware, how does that latency break down, and what does that imply for the agent loop design patterns Chapter 2 will develop?

We use two published benchmarks as anchors: **Llama 2 7B on MLPerf Client v0.6**, measured by Intel on a Core Ultra Series 1 processor, and **DeepSeek-R1-Distill-Llama-8B INT4 on OpenVINO Model Hub**, both real data points that set the floor and ceiling for what you can expect.

The Two Key Latency Metrics: TTFT and ITL

Model inference latency on accelerators is traditionally quoted as a single number (e.g., "inference takes 50 ms"). That's been obsolete for over a decade in LLM contexts because LLMs have two phases with radically different characteristics.

Time-To-First-Token (TTFT) is the latency of the prefill phase: the time from when you send the prompt to when the model emits the first output token. The prompt is static, potentially long (hundreds of tokens), and the entire computation is on the critical path — you can't generate a second token until the first one exists. TTFT is compute-bound.

Inter-Token Latency (ITL) is the latency of each subsequent token in the decode phase. The decoder sees only the new token slot plus the KV cache, and the computation is roughly constant per new token. ITL is memory-bandwidth-bound on NPU.

On Intel Core Ultra with Lunar Lake, the published benchmarks nail this split:

Llama 2 7B on MLPerf Client v0.6 (Intel internal, Core Ultra Series 1 Meteor Lake):

- TTFT at 128 input tokens: **1.09 seconds**
- ITL (tokens 2+): **~54 ms/token**
- Implied throughput: **18.55 tok/s** sustained

DeepSeek-R1-Distill-Llama-8B INT4 on OpenVINO Model Hub (public benchmark, Intel NUC 14 Pro with Lunar Lake):

- Measured at **6.10 tok/s sustained**, which is **~163 ms/token ITL**
- TTFT is not published; extrapolate from the 8B size and INT4 quantization

The 2.8× gap between Llama 2 (18.55 tok/s) and DeepSeek-Distill-8B (6.10 tok/s) is real. A naive explanation is parameter count: 7B vs 8B is 14% more matmul. But the gap is closer to 3×, not 14%, which means something structural is different. The honest answer: these are measured on different hardware revisions (Series 1 Meteor Lake vs Series 2 Lunar Lake is a 4× MACs gain), different quantization targets (Llama 2 at FP16? INT8?), and different workload assumptions (batch size, prompt length). The benchmarks are not apples-to-apples; treat them as reference ranges.

The Roofline: Hardware Limits

The sustainable throughput on Intel NPU is bounded by the LPDDR5X bandwidth ceiling from Chapter 1.1: **136.5 GB/s platform-wide shared among CPU, iGPU, and NPU**. No device gets the full 136.5 GB/s; the actual per-device quota depends on driver scheduling and competing loads.

For an 8B INT4 model:

- **Weight memory**: 4 GB (8B params × 4 bits/param / 8)
- **Sustained throughput**: 6.10 tok/s (from the published benchmark)
- **DRAM read rate**: 4 GB × 6.10 tok/s = **24.4 GB/s**

This is roughly **18% of platform peak bandwidth**. The NPU is not starving, but it's not saturating the bus either. The gap between 24.4 GB/s and 136.5 GB/s is scheduling overhead, driver latency, and contention from other agents on the SoC (CPU, iGPU). The roofline model says: if you could eliminate all contention and overhead, you'd hit bandwidth saturation at roughly **(136.5 GB/s) / (4 GB model weight) = 34 tok/s** — about 5.5× higher than what's measured. That gap is real and structural.

The practical implication: **you cannot expect sustained decode speeds above 15–20 tok/s on Lunar Lake NPU for reasonable 8B models**. Going faster requires either a smaller model, lower precision (NF4, FP8 on NPU 5), or moving decode to the iGPU.

Comparing to iGPU

The same Core Ultra platform has an Xe2 iGPU (Lunar Lake) or Xe1 iGPU (Meteor Lake). The iGPU is not on the same 136.5 GB/s bandwidth constraint as the NPU — it has its own path to VRAM — and it's substantially faster for decode workloads.

On the same hardware (Core Ultra Series 2), Llama 2 7B typically reaches **~40 tok/s on iGPU** (measured by community benchmarks; Intel does not publish iGPU LLM numbers). That's a **2.1× speedup over NPU** for decode. For prefill (TTFT), the gap is wider: iGPU TTFT is typically **300–400 ms** for a 128-token prompt, vs **1.09 seconds on NPU** — a 3–4× gap.

The hybrid story emerges: if you can split the workload with prefill on NPU and decode on iGPU, you get 2.1× throughput for the large constant-cost phase (decode) and take the NPU's hit only on the one-time prefill. Chapter 3.1 builds the code for this pattern.

What Phi Silica Tells Us

Microsoft's Phi Silica is the closest public reference architecture for an NPU-targeted LLM, deployed on Snapdragon X (Qualcomm NPU, not Intel). The published numbers are **TTFT 230 ms, 20 tok/s sustained** on a 2K context window. The architecture is: **CPU tokenizer + embedding + LM-head, NPU transformer blocks, CPU decode with N=64 KV sliding window**.

This is instructive not because Snapdragon X hardware maps cleanly to Intel NPU (it doesn't), but because it shows what real deployed decisions look like: **encoder on accelerator, decoder split between accelerator and CPU**, because the decode phase's structure (lots of memory, little compute per token) is where the accelerator's architecture breaks down.

Phi Silica also exposes the **sliding-window KV cache technique**: instead of keeping the full context KV in memory, keep only the most recent N tokens (here N=64). This trades recompute (re-running attention over discarded context) for memory bandwidth. For NPU where bandwidth is the constraint, this trade-off wins. The Llama 2 and DeepSeek-Distill benchmarks above use full KV caches. If they switched to sliding-window N=128, ITL would drop materially, but context awareness would degrade after 128 tokens. This is a tuning knob for the agent's working memory size.

Architecture-Specific Wisdom

Three things deserve to be nailed down because they're easy to get wrong:

Batching doesn't help on NPU for decode. On GPU, you can batch multiple independent decode streams and keep the compute pipeline full — token 1 from user A, token 1 from user B, token 1 from user C, all in parallel. On NPU with a fixed-shape pipeline and 136.5 GB/s bandwidth ceiling, batching adds more weight reads without adding more available bandwidth. Batching increases *latency* (because you're now serving multiple users sequentially) without increasing *throughput* (because you hit the bandwidth ceiling with a single-user stream). The practical result: **always use batch size 1 for decode on Intel NPU**.

LPDDR5X speed is shared, not divisible. The 136.5 GB/s includes all traffic: CPU instruction fetches, iGPU reads, NPU reads, system memory traffic. If the CPU is running code and the iGPU is running a concurrent task, the NPU's available bandwidth drops. If you want predictable NPU performance, you need to account for potential contention. The Phi Silica sliding-window approach partly exists to reduce bandwidth hunger, reducing contention sensitivity.

Compile-time overhead is real. The first invocation of a compiled model on NPU takes 30–60 seconds (from Chapter 1.2 cold-start benchmarks). Subsequent invocations take <3 seconds (warm start, cached to disk via `CACHE_DIR`). This cost is amortized over the model's lifetime in production, but for development and short-running agents, it's a gotcha. Always set `CACHE_DIR` to a persistent location; otherwise you pay the cold-start penalty on every process restart.

The Agent-Loop Latency Budget

A 5-step agent loop — where the agent reasons, takes an action, observes the result, and repeats — looks like this in latency terms:

- **Step 1 prefill**: 512-token accumulated prompt, ~4 seconds TTFT
- **Step 1 decode**: 64 output tokens (the agent's "Thought / Action / Observation"), ~10.4 seconds ITL
- **Steps 2–5**: same pattern, context grows each iteration
- **Total**: ~70–75 seconds for 5 steps at this prompt size

On iGPU: ~35 seconds.

This is the roofline for agent patterns on Intel NPU, and it's the why behind the Chapter 2 reasoning-architecture recommendations: ReAct (which is inherently loopy) doesn't fit the latency budget, but single-shot and cascade patterns do.

What This Section Bought You

You should now understand:

- **TTFT and ITL are two distinct metrics** with different hardware bottlenecks — compute vs bandwidth
- **Published benchmarks**: Llama 2 7B at 18.55 tok/s (TTFT 1.09s), DeepSeek-Distill-8B at 6.10 tok/s
- **The roofline ceiling is 136.5 GB/s LPDDR5X** shared across CPU/iGPU/NPU, yielding ~34 tok/s theoretical max for 8B INT4
- **iGPU is 2.1× faster than NPU for decode**, making hybrid prefill-on-NPU / decode-on-iGPU the natural pattern
- **Phi Silica shows real deployment wisdom**: CPU encoder/decoder, NPU transformer, sliding-window KV for bandwidth
- **Batch size 1 for decode on NPU**; batching doesn't increase throughput, only latency
- **Compile-time overhead is 30–60s cold, <3s warm**; set `CACHE_DIR` always
- **A 5-step ReAct loop takes ~70–75 seconds on NPU**, which is the structural reason Chapter 2 recommends single-shot or cascade patterns

Chapter 2 now turns from hardware to software: given these latency budgets, how does model state (KV cache, attention memory) factor into design, and what reasoning architectures actually work within constraint?

Previous: *1.2 Computational Constraints & Model Optimization* **Next:** *Chapter 2: Agent State & Decision-Making*

Revision #4
Created 2026-05-12 15:56:36 UTC by Admin
Updated 2026-05-12 19:16:28 UTC by Admin