2.3 Reasoning Loops Under Constraint

Chapter 2 closes here. We have a model that fits, weights we can stream, KV state we can manage, and decode at roughly 6–20 tok/s. The question this section answers: given that decode budget, what reasoning architectures actually work? The naive answer — bolt a ReAct loop on top and let the agent think — collides with the latency ceiling in a way worth being specific about.

The Three Reasoning Architectures

Three patterns dominate agent design, and they sort cleanly by NPU compatibility:

Single-shot. One prompt, one response. No loop. The agent reads the input, produces the output, done. Translation is the canonical single-shot task: source sentence in, target sentence out. The cost is one prefill plus one decode. Phi Silica's Click to Do affordances are single-shot. **This is the NPU-native pattern.**

Plan-then-execute. The model produces a plan once, then executes the plan deterministically (often without further model calls, or with a small number of pre-determined model calls). For a translation assistant: "rewrite this paragraph for a teenage audience and translate to French" decomposes to (1) rewrite via Phi-3.5-mini, (2) translate via M2M-100. The plan is one LLM call; the execution is a fixed pipeline. Two model calls total, predictable latency.

ReAct (Reason + Act). The model alternates between thinking and tool-calling in a loop, with each iteration informed by the last. The hallmark is that the *number of iterations is not known in advance*. This is the dominant pattern for cloud agents and the one developers reach for by default. **It's also the pattern that NPU latency budgets cannot afford.**

The ReAct Latency Budget

Let's price out a 5-step ReAct loop on Intel Core Ultra NPU, anchored on Chapter 1.3's two published benchmarks.

Assumptions: 512-token context per step (prompt grows as the loop accumulates), 64-token decode per step (the agent's "Thought / Action / Observation" turn). Using Llama 2 7B at MLPerf's TTFT-1.09s/128-tok-prompt and DeepSeek-Distill-Llama-8B's 163 ms/token decode as the conservative anchors:

Component	Value	Source
TTFT, ~128-token prompt	1.09 s	MLPerf Client v0.6
TTFT extrapolated to 512-token prompt	~4 s	linear-ish
ITL per decode token (8B INT4)	163 ms	OpenVINO Model Hub
Decode 64 tokens	10.4 s	computed
One ReAct iteration	**~14–15 s**	extrapolated
5 iterations	**~70–75 s**	extrapolated

On the same SoC's iGPU (12.8 tok/s, ~78 ms/token): one iteration ≈ 7 s, five iterations ≈ 35 s.

A 5-step ReAct agent at this context size on Intel NPU sits in the 60–90 second range — usable for offline summarization, marginal for chat, infeasible for interactive autocomplete. Stretching the loop to 10 steps doubles it. ReAct's behavior of growing the context monotonically with each step makes it worse over time, not better, because every iteration's prefill takes longer than the last.

These numbers are extrapolations from published single-call benchmarks, not measurements of ReAct loops. We flagged in Chapter 1.3 that Intel and Microsoft have published almost nothing about multi-step agents on NPU. Treat the table as the right order of magnitude, not as a precise SLA.

Why Single-Shot Wins on NPU

The structural reasons single-shot translates to NPU and ReAct doesn't:

Each ReAct step pays full TTFT. The prefill is the compute-bound, MAC-array-heavy phase; on NPU it's relatively fast per-prompt, but you do it `N` times per loop instead of once. A 5-step ReAct burns 5× the TTFT of an equivalent single-shot.

Context grows monotonically. Step 1's prefill is short. Step 5's prefill includes everything that came before. The TTFT cost rises through the loop. Chunked prefill on NPU helps, but doesn't fix the issue: each chunk costs constant time, and step 5 has more chunks.

Cold-cache pressure increases. The KV state from step 1 has to be valid at step 5 — which works fine within `LLMPipeline.start_chat()` but means the state-variable allocation must accommodate the full final context. You commit to the worst-case footprint up front.

Greedy-only hurts most here. On NPU's static pipeline, no beam search. ReAct's "Thought" outputs are exactly the kind of free-form text that benefits from beam-4 sampling diversity. Greedy ReAct tends to fall into repetitive loops.

The cumulative effect: ReAct on Intel NPU magnifies the very constraints that NPUs are worst at. It's the wrong architecture for the hardware.

What to Do Instead

Prefer single-shot. If your task can be reduced to one prompt and one response, do that. Translation is single-shot. Summarization is single-shot. Tone-rewrite is single-shot. "Explain this code" is single-shot. The cloud-agent culture's enthusiasm for ReAct has obscured how many useful tasks don't actually need a loop.

Use plan-then-execute when you need composition. A planning call decides the structure; deterministic code runs the plan. The planning model needs to produce structured output (JSON, XML), which works fine in single-shot. The execution is fixed-cost, and any individual sub-call can hit its own device — the plan can route one sub-task to NPU, another to iGPU.

Use the cascade pattern for triage. A tiny model on NPU decides whether the request needs the heavy model. The cheap path is sub-second; the expensive path is the budget you'd already pay for a single-shot. Worst-case latency is the heavy-model latency, not the heavy-model latency *times the number of ReAct iterations*.

When you genuinely need ReAct, run it on iGPU. The 2.1× speedup from Chapter 1.3 turns 75-second NPU ReAct into 35-second iGPU ReAct. Still slow by cloud standards; in budget for offline workflows like document analysis. The NPU's role becomes drafting and triage; the iGPU does the reasoning loop.

Tighten context aggressively. Every byte you can prune from the running prompt is bandwidth you don't pay for at every step. The Phi Silica architecture's N=64 sliding window over context is an aggressive version of this — most of the time you don't need everything in scope.

Working vs Long-Term Memory

The reasoning loop's *state* — what the agent remembers across steps — splits into two regimes.

Working memory is what's in the prompt this turn. On NPU it's bounded by `MAX_PROMPT_LEN`. Generous on chunked-prefill-capable models (up to 8K validated on Lunar Lake); tighter on encoder-decoder seq2seq like M2M-100. Working memory is fast (it's in the model's attention window) and ephemeral (it doesn't persist across sessions).

Long-term memory lives outside the model — in a SQLite database, a vector store, a key-value cache, a local filesystem. It's persistent and unbounded in size, but accessing it costs an explicit retrieval step before the next prompt. For NPU agents, **long-term memory needs to be local**, which means it's a few milliseconds away and orders of magnitude cheaper than another NPU forward pass.

The pattern that works well on NPU: aggressive working-memory pruning (small context, small TTFT), with retrieval into a local vector store between turns. The vector store is on CPU; the embedding model can be on NPU (which is exactly the kind of single-shot, batch-friendly workload NPU is great at — see Chapter 3.3 for the OpenVINO 2026.1 `TextEmbeddingPipeline` NPU support). The reasoning model gets short, dense context; the agent stays responsive.

Where Intel and Microsoft Have Been Quiet

Honest gaps to flag, because this is the section most likely to invite extrapolation:

No Intel-published guidance on multi-step LLM agents on NPU. The Hugging Face × Intel Qwen3-8B Agent blog is the closest analog, and it explicitly runs on iGPU, not NPU.

Phi Silica is documented as single-turn. Microsoft routes it through Click to Do prompt templates with no learned router and no documented multi-step loop. The Windows Developer Blog extends the Phi Silica stack to DeepSeek-R1-Distill (1.5B at ~40 tok/s, 14B at ~8 tok/s on Snapdragon X NPU) — a reasoning model on NPU — but does not describe an agent architecture around it.

No published ReAct-loop measurements on Intel NPU exist. The 60–90 second budget in the table above is extrapolation from single-call benchmarks. If you build a real ReAct agent on NPU, the data points you collect will be original contributions to the public record.

The chapter's recommendation — prefer single-shot, fall back to plan-then-execute, treat ReAct as the iGPU pattern — reflects the absence of evidence for ReAct working well on NPU as much as it reflects the math. When more data appears the calculus might shift. As of May 2026 it hasn't.

What This Section Bought You

You should now understand:

- **Three reasoning architectures**: single-shot (NPU-native), plan-then-execute (decomposable), ReAct (iGPU pattern, not NPU)
- **A 5-step ReAct loop costs ~70–75 seconds on NPU** vs ~35 seconds on iGPU for an 8B INT4 model — extrapolated, not measured
- **ReAct magnifies the constraints NPUs are worst at**: repeated TTFT, growing context, greedy-only sampling, accumulating KV state
- **Single-shot tasks are more common than the cloud-agent literature suggests** — translation, summarization, tone-rewrite, code explanation all fit
- **Cascade triage is the NPU-native multi-step pattern** — tiny model decides whether the heavy model needs to run

- **Working memory (prompt) is bounded by `MAX_PROMPT_LEN`; long-term memory lives in local stores** with embedding-model retrieval between turns
- **Intel and Microsoft have published almost nothing on multi-step NPU agents** — be honest about the gap when designing for production

Chapter 2 ends here. The reader now has a working mental model of the constraints, the state, and the decision-making patterns. Chapter 3 turns to tools: how does an NPU-bound agent reach the world, what tool designs survive the latency budget, and where does the cloud fit?

Previous: *2.2 KV Cache Engineering* **Next:** *Chapter 3: Tool Use & Integration Patterns*

Revision #4
Created 2026-05-12 15:59:52 UTC by Admin
Updated 2026-05-12 19:18:22 UTC by Admin