NEWSLETTER

Harnessing the Agent: From 200-Feature JSONs to 8GB VRAM Qwen3.6

April 21, 2026

The common thread across this week's reading: the model is increasingly the easy part. Whether you're running Qwen3.6-35B-A3B on a gaming GPU, fine-tuning it with Unsloth, or turning Claude Opus 4.5 loose on a multi-day coding project, the interesting engineering has migrated outward — into harnesses, Modelfiles, sandbox manifests, sampler recipes, and structured prompt templates. This issue digs into the scaffolding around the model: what Anthropic learned about long-running agents, how Thoughtworks' Birgitta Böckeler frames the feedforward/feedback loop, and the very concrete knob-turning required to get a 35B MoE model to cough up 21 tok/s on 8GB of VRAM.

01

Why Opus 4.5 still can't one-shot claude.ai — and the 200-feature JSON that fixes it

Anthropic's engineering team went public in Effective harnesses for long-running agents (November 2025) with a blunt admission: even a frontier model like **Opus 4.5** running on the Claude Agent SDK in a loop can't build a production-quality clone of claude.ai from the prompt "build a clone of claude.ai." Compaction isn't enough. Agents fail in two ways — trying to one-shot the whole app and running out of context mid-implementation, or, in later sessions, "looking around, seeing that progress had been made, and declaring the job done."

Their fix is a two-agent split. An **initializer agent** runs once, writes `app_spec.txt`, an `init.sh`, a `claude-progress.txt` journal, and — critically — a `feature_list.json` with 200+ granular test cases (e.g. "a user can open a new chat, type in a query, press enter, and see an AI response"), each starting with `passes: false`. Subsequent **coding agent** sessions are prompted to work one feature at a time and are only allowed to flip that boolean. Anthropic explicitly chose JSON over Markdown "as the model is less likely to inappropriately change or overwrite JSON files." The reference implementation is on GitHub with a defense-in-depth bash allowlist and project-dir filesystem jail.

Thoughtworks' Birgitta Böckeler reaches for the same steering-wheel metaphor from a different angle in Harness engineering for coding agent users (April 2026). She defines the harness as everything around the model — `Agent = Model + Harness` — and splits user-facing controls into a 2×2: **Guides** (feedforward, steer before action) vs **Sensors** (feedback, detect after), each either **computational** (linters, ArchUnit tests, codemods) or **inferential** (AGENTS.md, Skills, LLM-as-judge). Her prescriptive advice: "keep quality left" — push fast computational sensors pre-commit and reserve expensive inferential judges for post-integration. The human's job becomes iterating on the harness whenever a

mistake repeats, which is a subtle but important reframe from "prompt the model better."

On the open-source side, HKUDS/OpenHarness (April 2026, currently at v0.1.7 with 10.7k stars) is trying to ship a full agent harness you can actually run on top of your existing Claude Code or Codex subscription — 43 built-in tools, an `anthropics/skills`-compatible plugin system, subagent delegation, and v0.1.6's **Auto-Compaction** that preserves task state across context compression so agents can run "multi-day sessions without manual compact/clear." The bundled **ohmo** personal agent hooks into Feishu/Slack/Telegram/Discord and opens PRs autonomously. *Additional info:* a complementary testing pattern from Ivett Ördög, Approved Scenarios (date unavailable), argues the only way to keep up with agent-generated tests is approval-file fixtures — `.approved.md` files mixing input, service calls, and expected output so review becomes a diff scan rather than reading assertions.

> **FURTHER READING**
>
> Anthropic Engineering — Effective harnesses for long-running agents — November 2025
>
> Martin Fowler — Harness engineering for coding agent users (Böckeler) — April 2026
>
> anthropics/claude-quickstarts autonomous-coding reference
>
> HKUDS/OpenHarness — open agent harness with ohmo — April 2026

02

Codex, OpenCode, and the sandbox-vs-harness boundary

The commercial side of the harness story is consolidating. *Additional info:* OpenAI's Codex page (date unavailable) pitches a single agent that spans app, editor, and terminal, unified by your ChatGPT account, with **parallel worktrees** across cloud environments, **Skills** for team-standard workflows, and **Automations** for background triage. The testimonial reel is specific enough to be interesting: Harvey claims 30–50% iteration-time reduction, Duolingo uses Codex PR reviews to catch backward-compat bugs, and Wonderful reportedly replaced its other harnesses outright.

The open-source counter-offer is *Additional info:* OpenCode (date unavailable), claiming 140K GitHub stars and 6.5M monthly developers with BYO-model support across 75+ providers via Models.dev — including local models. The pragmatic hooks: login with GitHub to drive it off a Copilot subscription, or with OpenAI to use ChatGPT Plus/Pro, and it's LSP-aware so the right language servers load automatically. Their curated "Zen" bundle is a benchmarked set of models tuned for coding-agent consistency.

The more architecturally interesting piece is *Additional info:* OpenAI's Sandbox Agents guide (date unavailable), which draws a crisp line between the **harness control plane** (trusted infra, auth, audit) and the **sandbox compute plane** (filesystem, shell, mounts, credentials). A `SandboxAgent` defines agent behavior + sandbox defaults; a `SandboxRunConfig` supplies the per-run session source. Manifests compose from `File` / `Dir` / `LocalFile` / `GitRepo` / `S3` / `GCS` / `R2` / `Azure` entries, and mount paths must be workspace-relative — `..` escapes are rejected so manifests stay portable across local, Docker, and hosted providers. A "turn" is explicitly a model step, not a shell command, which is the right abstraction: many tool calls can happen inside one reasoning turn.

The shared takeaway across all three: the differentiator isn't the model — it's how cleanly the product separates resumable workspace state from the (stateless) model loop. That's the same insight driving Anthropic's feature-list file, just productized.

> **FURTHER READING**
>
> OpenAI Codex product page
>
> OpenCode — open-source coding agent
>
> OpenAI API — Sandbox Agents guide

03

Qwen3.6-35B-A3B on 22GB of RAM: the sampler recipes that actually matter

Alibaba's new **Qwen3.6-35B-A3B** MoE is small enough to be interesting on consumer rigs — Unsloth's run guide (April 2026) lists hardware budgets of 17GB (3-bit), 23GB (4-bit), 30GB (6-bit), and 70GB (BF16) total RAM+VRAM. Max context is 262,144 tokens, extendable to 1M via YaRN. The standout warning: **"Do NOT use CUDA 13.2 as you may get gibberish outputs. NVIDIA is working on a fix"** — worth pinning your driver. If you're seeing gibberish anyway, they suggest `--cache-type-k bf16 --cache-type-v bf16` before blaming the quant.

The sampler settings differ sharply by mode. **Thinking mode for precise coding** wants `temperature=0.6, top_p=0.95, top_k=20, presence_penalty=0.0`; **thinking for general tasks** cranks temperature to 1.0 and presence_penalty to 1.5. **Non-thinking Instruct** uses `temperature=0.7, top_p=0.8`. Unsloth also flags a Qwen3.5→3.6 improvement in nested-object tool-call parsing and says its uploads now support the **developer role** for Codex and OpenCode. A hard caveat: Qwen3.6 GGUFs *do not work in Ollama* today because of separate mmproj vision files — you need a llama.cpp-compatible backend.

The theory-meets-reality moment comes from user FUS3N on r/LocalLLaMA (April 2026), running Q3_K_S on 8GB VRAM + 24GB RAM with a mmproj-F16 vision adapter, getting ~21 tok/s that drifts down to 19.5 after a few messages. Their llama-server flags are the practical artifact: `-ngl 99 -ctk q8_0 -ctv q8_0 --flash-attn on --n-cpu-moe 38 -b 4096 -ub 1024` with the non-thinking-general sampler. A commenter pushes back that `-b 4096 -ub 1024` actually raises VRAM pressure under tight constraints and recommends letting `--fit on` dynamically adjust context instead — exactly the kind of vendor-rec-vs-hands-on delta you only find in forum threads.

Additional info: for the less adventurous, Codecademy's Qwen 3 + Ollama walkthrough (date unavailable) covers `ollama pull qwen3:0.6b` through 235B, the `--think` / `--think=false` / `--hidethinking` flags and `/set think` chat command, and calling the REST API at `http://localhost:11434/api/generate`. Fine for Qwen 3.x — just remember Qwen 3.6 isn't in that world yet.

> **FURTHER READING**
>
> Unsloth — Qwen3.6 run guide (samplers, hardware, llama.cpp) — April 2026
>
> r/LocalLLaMA — Qwen 3.6 on 8GB VRAM/24GB RAM tuning thread — April 2026
>
> Unsloth Qwen3.6-35B-A3B GGUFs on Hugging Face
>
> Codecademy — Qwen 3 + Ollama setup & fine-tuning

04

Unsloth's notebook buffet: GRPO for sudoku, LoRA from 3GB VRAM, and the Modelfile that saves your export

If you're fine-tuning anything open, *Additional info:* Unsloth's notebook index (date unavailable) is the one-stop map — free Colab/Kaggle notebooks for Qwen3.5/3.6, Gemma 4, gpt-oss, Llama 3, Mistral, Phi-4, DeepSeek, across SFT, GRPO/RL, vision, TTS, and embeddings. The GRPO set alone is worth browsing: **sudoku solving, auto-winning 2048, and automatic CUDA kernel creation** are all shipped as ready-to-run examples. Larger targets (70B Llama, 120B gpt-oss, 31B Gemma-4) need Colab paid or Kaggle's free T4 tier. Embedding tuning is now a first-class citizen: EmbeddingGemma 300M, Qwen3-Embedding 0.6B/4B, BGE M3, ModernBERT.

The *Additional info:* Fine-tuning LLMs Guide (date unavailable) is the conceptual primer worth handing a new teammate. It's frank about the RAG question: "Fine-tuning can replicate RAG capabilities but not vice versa" — weights get modified, context only gets augmented. LoRA trains roughly 1% of params via low-rank A/B matrices; QLoRA adds 4-bit quantization for ~75% memory savings. Unsloth's own claim is **free fine-tuning from 3GB VRAM, 2× faster with 70% less VRAM** than vanilla Hugging Face. The non-obvious tip: train and serve in the same precision — research shows this preserves accuracy better than a mismatched pipeline.

For hands-on multimodal work, *Additional info:* the Qwen3.5-4B Vision Colab (date unavailable) runs on a free T4 and fine-tunes against the `unsloth/LaTeX_OCR` dataset (68,686 handwritten-formula→LaTeX pairs). LoRA config is `r=16, lora_alpha=16, dropout=0`, and the notebook finetunes both vision and language layers plus attention and MLP blocks. User-content chat format is an array of `{type:text}` + `{type:image}`

objects; same approach works for Llama-3.2-11B-Vision, Pixtral-12B, Qwen2-VL-2B/7B/72B, and the Llava family.

The pitfall that burns most newcomers is the export step. *Additional info:* Unsloth's Saving to Ollama guide (date unavailable) auto-generates a Modelfile embedding the exact chat template used during training — the fix for the classic "great on Unsloth, gibberish on Ollama" regression. Default export is Q8_0 (quick), with `q4_k_m` as the popular smaller option. Most post-export quality drops trace back to a wrong chat template, the wrong EOS token, or an extra BOS token getting inserted — the documented conversational notebooks (Qwen-3 14B, Gemma-3 4B, Llama-3.2 3B, Phi-4, Mistral v0.3) enforce correct templates by construction.

> **FURTHER READING**
>
> Unsloth Notebooks index (SFT/GRPO/Vision/TTS/Embedding)
>
> Unsloth Fine-tuning LLMs Guide (LoRA vs QLoRA vs FFT)
>
> Qwen3.5-4B Vision fine-tune on LaTeX_OCR (Colab)
>
> Unsloth — Saving to Ollama (Modelfile + GGUF)

05

The Ollama config surface: num_ctx=2048 by default, and ollama-js has no timeout

Two references every Ollama user should have open in a tab. *Additional info:* the Modelfile reference (date unavailable) documents the seven-instruction DSL — `FROM`, `PARAMETER`, `TEMPLATE`, `SYSTEM`, `ADAPTER`, `LICENSE`, `MESSAGE` — and every runtime knob: `num_ctx`, `temperature`, `top_k`, `top_p`, `min_p`, `repeat_penalty`, `stop`, `num_predict`, `seed`. The trap: `num_ctx` **defaults to only 2048**, which silently truncates anything serious. `FROM` accepts base model names, Safetensors directories (Llama/Mistral/Gemma/Phi3), or raw GGUF files, and `ADAPTER` applies (Q)LoRA adapters as either Safetensors or GGUF. To fork an existing config: `ollama show --modelfile <model>`.

On the client side, *Additional info:* ollama-js issue #103 (date unavailable) surfaces a subtle footgun: there's **no timeout parameter**. A user running Mixtral 8x22B with heavy CPU offload — 86GB required but only 15.6GB VRAM, so only 7 of 57 layers offloaded per `journalctl` — watched long `stream:false` generations fail silently with "Error. undefined." A maintainer clarifies: you build timeouts on top of `ollama.abort()` (global) or `AbortableAsyncIterator.abort()` (per-request), or you flip to `stream:true`, which avoids the problem because chunks arrive incrementally. It's a clean example of the harness/model split from the coding-agent cluster showing up at the client layer.

The practical combination: set `num_ctx` explicitly in your Modelfile to match your actual use case, export Unsloth models with their auto-generated Modelfile so templates don't drift, and if you're driving Ollama from Node for anything slow, default to streaming or wire up an explicit abort controller with a timeout.

> FURTHER READING

> Ollama Modelfile Reference
>
> ollama-js #103 — Timeout for long generation
>
> ollama-js abort examples (any-request / specific-request)

06

EQ4C's 14-tag prompt scaffold: the feedforward control at the prompt layer

If Böckeler's framework puts AGENTS.md and Skills in the "feedforward + inferential" quadrant, the *Additional info:* EQ4C Template Guide (date unavailable) is a maximalist version of the same idea for one-shot prompts. It defines ~14 tagged sections — **System, Context, Meta, Variables, Instructions, Constraints, Definitions, Examples, Output Format, Reasoning, Memory, User Input, Evaluation, Metrics, Tools** — with good/bad examples for each. The `System` tag decomposes further into Role / Expertise / Boundaries / Mode sub-tags, forcing an explicit behavioral contract.

A few of the choices are genuinely useful even if you don't adopt the whole thing. The `Meta` tag splits Tone / Audience / Purpose so the prompter can't fudge "who's this for and why." The `Examples` block requires both a `GoodOutput` and a `BadOutput` so the model steers by contrast, which works noticeably better than positive-only few-shot. And `Evaluation` + `Metrics` let the model self-score output with a Score + Reason pair — effectively a tiny LLM-as-judge bolted onto the same call.

The template ships in three equivalent serializations — XML, YAML, and Markdown — so the same structure can feed an automation pipeline, a database row, and a human-readable doc without rewriting. If you're already investing in harness controls for coding agents, this is the same discipline applied to the content-generation use case: constrain first, inspect output later.

> **FURTHER READING**
>
> EQ4C Template Guide — XML/YAML/Markdown prompt scaffolds
>
> EQ4C Tools homepage

The meta-shift worth watching: harness vocabulary is starting to unify across companies. Anthropic's `feature_list.json`, OpenAI's sandbox manifest, Böckeler's feedforward/feedback quadrants, and OpenHarness's tool+skill+memory separation are all converging on the same abstraction — a durable workspace that outlives any single context window, with explicit controls over what the model can read and change. Expect the next round of agent products to compete less on model choice and more on how elegantly they expose those controls.