Tool-Skills vs. Workflow-Skills: Routing and Composition in a Multimodal Agent Harness

Kevin Dela Rosa
Cloudglue
USA
kdr@cloudglue.dev

Abstract

Multimodal agents have a much larger and more fragmented tool surface than coding agents, and on top of it real users want *deliverables* (meeting breakdowns, sales-coaching dashboards, captioned reels) rather than raw tool outputs. In Tinycloud, a production multimodal video agent harness, we observe that the skills shipped to cover this gap separate into two recurring categories: *tool-skills* that wrap a single external API or system tool and serve as primitive vocabulary, and *workflow-skills* that orchestrate tool-skill calls plus a template render to produce one named deliverable. Template count helps distinguish the two in this harness: workflow-skills carry exactly one deliverable template, tool-skills carry zero or a reusable template library. We complement this case study with a six-task tool-selection ablation across three loading conditions: full autoload routes all six tasks correctly, autoload-off slows execution and produces hard failures, and the production default misroutes one task because its lexical signal collides with an autoloaded tool-skill that pulls attention away from a listed workflow-skill. The headline lesson is that prompt-exposure of skills is not monotonically helpful; partial exposure can create lexical competition that hurts routing.

Keywords: agent skills, multimodal agents, video understanding, agent harnesses, procedural knowledge, tool-skills, workflow-skills, skill routing

1 Introduction

Agent harnesses for code converged on a small vocabulary of file-system tools: read, write, edit, grep, run-shell. The shape of the agent's tool surface is largely settled. Skills built on this surface, structured packages of instructions and scripts loaded into the agent at inference time, are an emerging answer to the procedural-knowledge gap, and the `SKILL.md` format [2] has become a community-wide convention. Recent work has examined Skills' structure-efficacy relationships, security profile, and self-improvement potential.

Multimodal agents face a different setup [3, 5]. The tool surface is much larger: speech transcription, optical character recognition, vision-language description, structured entity extraction, semantic search, segmentation, transcoding, captioning, and connector ingestion are all separate primitives, with no `grep` equivalent that collapses them. On top of this larger primitive layer, real users typically want *deliverables*: a meeting breakdown of a recorded call, a sales-coaching dashboard with talk-ratio metrics, a YouTube publishing package, a captioned reel. Tools alone do not produce deliverables; the agent has to orchestrate.

We have shipped Tinycloud,[1] a freely-available multimodal video agent harness whose skills cover both layers. In production usage of Tinycloud, the skills cluster into two recurring categories with different loading patterns:

- **Tool-skills** are atomic primitives that wrap an external API or system tool. They are autoloaded into the system prompt, so the agent always has them available as vocabulary.
- **Workflow-skills** are domain-specific recipes that orchestrate multiple tool-skill calls, an LLM reasoning step, and a template render to produce a single named deliverable. They are loaded on-demand: their full body is injected into the prompt only when the user requests their specific deliverable.

We did not design the split up-front; it emerged from observing which skills aged well in production and which paid context-window rent without earning it.

Contributions. This paper presents a case study of one production multimodal video agent harness, with three observations:

- a recurring structural distinction between tool-skills and workflow-skills (Section 3), with template count as a useful marker for the distinction in this harness (Section 4, Table 1);
- a small token-budget ablation showing the autoload split has bounded prompt-budget effect (Section 5);
- a six-task tool-selection ablation showing the load pattern materially affects routing, including a non-monotonic failure where partial exposure routes worse than either full exposure or none (Section 6).

[1] https://tinycloud.sh

The harness source and the cataloged skills are available at https://tinycloud.sh.

2 Background

Skills, tools, and memory. The `SKILL.md` convention and the autoload mechanism trace to Anthropic's Agent Skills launch [2], adopted across multiple harnesses including Claude Code [1]; [11] provides a recent survey. Voyager [10] framed skills as composable code chunks discovered via automatic curriculum; our workflow-skills resemble these but are author-curated, deliverable-shaped, and template-emitting. The tool layer beneath connects to ToolLLM [8] (tool selection at scale), Toolformer [9] (foundational tool-use pattern), and ReAct [12] (interleaved reasoning and acting). MemGPT [7] frames the context window as managed memory; the autoload flag is a static, design-time analog of its runtime paging, and a learned variant is a natural next step.

3 The Skill System

3.1 Format and loader

A skill in Tinycloud is a directory containing three artifacts: a `SKILL.md` file with YAML frontmatter (`name`, `description`, optional `autoload`; default true) and a free-form body the agent reads as procedural knowledge; an optional `scripts/` directory with executable `.ts` or `.sh` files invoked via the agent's bash tool; and an optional `templates/` directory with reference files (HTML, JSON schemas, Markdown) the agent reads then customizes.

3.2 Surface area in the system prompt

The autoload flag controls how a skill is surfaced to the agent at session start. Skills with `autoload: true` appear in a *Skills* block of the system prompt with full per-skill detail (name, description, paths to scripts, paths to templates, plus per-skill usage guidelines maintained by the harness). Skills with `autoload: false` appear in a separate *Additional Skills* block with name and description only; their full instructions are injected on demand when the user (or agent) issues `/skills:<name>`. The asymmetry is deliberate: full skill bodies and per-skill guidelines can be hundreds to thousands of tokens each, so loading every skill at every turn would consume context budget on workflows that fire only a small subset.

3.3 Two emergent categories

In production, the autoload split appears to correspond to a structural distinction in what each skill does:

- **Tool-skills** are atomic primitives that wrap an external API or system tool: speech transcription, semantic search, file management, structured extraction, ffmpeg-style media manipulation. The agent treats them as a vocabulary that composes. They are autoloaded so the agent always has them available.
- **Workflow-skills** are domain-specific deliverable recipes that orchestrate multiple tool-skills, an LLM reasoning step, and a template render to produce something a user explicitly asked for: a meeting breakdown, a sales-coaching report, a blog post, a YouTube publishing package. They are loaded on-demand because each is long, narrow, and fires only when its specific deliverable is requested.

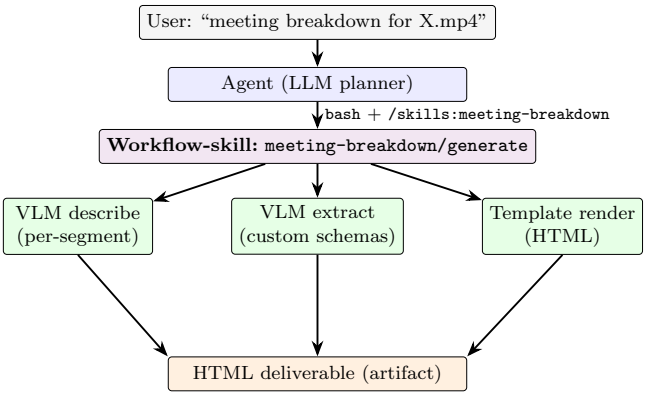

Figure 1. A workflow-skill (**purple**) orchestrates tool-skill calls (**green**): one or more VLM describe / extract jobs plus a template render, producing one user-facing deliverable (**orange**). The agent invokes the workflow-skill once via bash; the underlying tool-skills are orchestrated inside the skill's orchestration script, not step-by-step by the agent. This frames the workflow-skill as a recipe over a tool-skill vocabulary.

3.4 Templates as a structural fingerprint

In our harness the `templates/` directory serves as a useful category marker alongside the `autoload` flag. Workflow-skills consistently contain *exactly one* template: the deliverable shape (an HTML report, a publishing-metadata page, a blog-post layout). Tool-skills contain either zero templates or, in the special case of a primitive-library tool-skill, multiple templates that *other* skills consume via a template-name flag. Thus the same artifact (`templates/`) plays two distinct roles (deliverable shape vs primitive library), and the per-skill template count helps distinguish them.

Figure 1 shows how a workflow-skill orchestrates tool-skills under the hood for a single user request.

4 Catalogue

Table 1 lists the twelve skills currently shipped with the harness, classified along the tool / workflow distinction. The autoload column is the production default; *scripts* counts non-utility scripts (excluding shared helpers);

templates counts files in `templates/`. In this harness, the split is consistent: every workflow-skill is on-demand and contains exactly one template; every tool-skill is autoloaded and contains either zero templates or, in one case, a primitive library of templates that other skills consume.

Table 1. The twelve production skills. *Cat.* is T (tool-skill) or W (workflow-skill); *Auto.* is the production autoload flag; *Sc/Tp* = (non-utility scripts) / (template files). Workflow-skill templates shape the deliverable; `media-artifact`'s seven templates are a primitive library consumed by other skills.

Skill	Cat.	Auto.	Sc/Tp	Role
Tool-skills (atomic primitives)				
`video-analyze`	T	✓	7/0	VLM describe + extract
`cloudglue-files`	T	✓	7/0	Fetch / describe by ID
`data-connectors`	T	✓	6/0	Connector ingestion
`collection-search`	T	✓	4/0	Semantic search + Q&A
`captions`	T	✓	2/0	SRT / VTT / transcripts
`local-search`	T	✓	1/0	Grep cached enrichments
`media-artifact`	T	✓	2/7	Render template library
Workflow-skills (deliverable recipes)				
`meeting-breakdown`	W	✗	2/1	Timeline + chapters
`sales-coaching`	W	✗	2/1	Coaching dashboard
`ad-analysis`	W	✗	2/1	Hook + pacing + CTA
`blog-post`	W	✗	2/1	Long-form post
`youtube-publish`	W	✗	2/1	Title / desc / chapters

Two patterns are worth highlighting. Tool-skills vary in script complexity but uniformly carry no deliverable template (they are vocabulary, not output), while all five workflow-skills share an identical fingerprint — two scripts (an orchestrator and a renderer) plus one deliverable template — because the recipe is the same: orchestrate tool-skill calls, synthesize, render. The one tool-skill carrying templates (`media-artifact`) holds a reusable *library* of seven primitives that other workflow-skills consume by name; the same artifact directory thus plays two distinct roles, and the per-skill template count helps signal which.

5 Empirical

Question. What does the autoload split cost? We measure the system-prompt token budget under three loading conditions, decomposing the cost into (a) the static-rule overhead the harness emits regardless of skill set and (b) the marginal cost of each skill that surfaces as autoloaded.

Conditions. We compare three configurations of the Table 1 skill set:

- **All-on:** every skill has `autoload: true` (the full *Skills* block lists every skill at full detail, including its scripts and template paths).
- **Default:** the production configuration (tool-skills autoloaded, workflow-skills on-demand).
- **All-off:** every skill has `autoload: false` (the entire skill set surfaces in the lightweight *Additional Skills* block as name and description only; full bodies and per-skill paths inject only via `/skills:<name>`).

Two production skills sit outside the Table 1 taxonomy: `yt-dlp` (a utility skill omitted as it doesn't fit either category) is included in the ablation under each condition's loading rule, while a Cloudglue SDK skill is force-autoloaded as a system invariant under all three conditions and excluded from the ablation; it contributes an additional ∼1,862 tokens of body content to every prompt regardless of condition. The ablation thus measures the marginal effect of the remaining 13 skills (the 12 catalogued plus `yt-dlp`). Token counts are computed by running Tinycloud's system-prompt builder under each condition and tokenizing the output with `tiktoken cl100k_base`[2].

Table 2. System-prompt token budget by loading condition. *Skills block* is just the `Skills` + `Additional Skills` sections; *Static rules* is the fixed walkthrough and per-tool guidelines block emitted regardless of skill set.

	All-on	Default	All-off
Skills block (tokens)	1,789	1,411	457
Static rules + tools (tokens)	6,379	6,379	6,379
Total system prompt (tokens)	8,168	7,790	6,836
Skills-block ratio vs. All-off	3.91×	3.09×	1.00×
Total prompt ratio vs. All-off	1.19×	1.14×	1.00×

Token-budget findings. The absolute autoload tax is modest: the entire workflow-skill bundle adds ∼378 tokens to the prompt vs. *Default*, and even *All-on* sits within the same order of magnitude as *All-off*. The skills block itself varies ∼4×, but static-rule overhead (∼6.4k tokens) dominates the total prompt. The autoload split modestly reduces prompt surface area; the larger practical effect, we will show next, appears in routing behavior rather than raw token savings.

Tool selection and task success. We complement the budget measurement with a six-task ablation under each loading condition. Each cell is a single headless run of Tinycloud v0.2.1 (with Claude Opus 4.6 as the planner) that emits a structured JSON trace of the planner's tool calls; n=1 per condition–task pair, no paraphrase or

[2]Used here as a public-tokenizer proxy for Claude Opus's tokenizer, which is not publicly distributable. Token ratios across conditions are insensitive to the tokenizer choice.

seed sweep. The working directory contained ∼30 unrelated videos beyond the fixture targets. Per cell we record (a) which skill the agent invokes to do the task's work and (b) whether the canonical deliverable is produced. The fixtures: *q1* "meeting breakdown for [meeting].mp4" (gold: `meeting-breakdown`); *q2* "coach the rep in [meeting].mp4" (`sales-coaching`); *q3* "analyze [video].mp4 ad: hook structure, pacing, CTA" (`ad-analysis`); *q4* "describe [clip].mp4" (`video-analyze`); *q5* "find every moment across the 4 sales meetings where pricing or fees are discussed" (`collection-search`); *q6* "make a video player for [meeting].mp4" (`media-artifact`). q1–q3 target workflow-skills, q4–q6 target tool-skills. We treat this as a small in-house complement to larger skill benchmarks [6]; Table 3 summarizes the eighteen-cell result.

Table 3. Tool selection per fixture task. ✓/✗ indicates whether the agent invoked the gold skill; the canonical deliverable was produced in every ✓ cell and in no ✗ cell. `wall-seconds/tool-calls` follows. q1–q3 target workflow-skills, q4–q6 target tool-skills.

Task	Default	All-on	All-off
q1	✓ 194/6	✓ 201/7	✓ 262/11
q2	✓ 223/7	✓ 206/10	✗ 58/7
q3	**✗ 231/13**	**✓ 157/13**	**✓ 192/12**
q4	✓ 93/7	✓ 101/5	✓ 147/14
q5	✓ 298/24	✓ 277/16	✗ 1038/83
q6	✓ 168/4	✓ 261/13	✓ 447/23
correct	5/6	**6/6**	4/6
mean s	201	200	**357**
mean calls	10.2	10.7	**25.0**

Routing findings. Three observations. First, All-on is the only condition that hits gold-skill selection on every cell; Default misses q3 and All-off misses q2 and q5. Second, mean wall-clock is roughly tied between Default and All-on but ∼1.8× higher under All-off, driven by the agent's need to filesystem-search for skill scripts when no skill body is in the prompt; q5/All-off in particular escalated to subagent-delegation over raw cached describe-JSON files, costing 17 minutes and 83 tool calls. Third, q3 is the most informative cell: it fails under Default but recovers under both All-on *and* All-off, indicating the autoload knob is not monotonic in helpfulness on the routing axis (Section 6). With n=6 per condition, the 95% binomial CI on per-condition routing accuracy spans ±0.30 (e.g., Default's $5/6 = 0.833 \pm 0.298$); the Default-vs-All-on difference is not statistically significant in isolation, and we rely on the qualitative q3 mechanism evidence in Section 6 to interpret it.

6 Discussion

Prompt exposure is not monotonically helpful. The headline finding of the routing ablation is the q3 cell. The prompt — "Analyze the honda_eleveate.mp4 ad: hook structure, pacing, CTA" — routes to the `ad-analysis` workflow-skill under both All-on (where its body is inlined) and All-off (where it is listed alongside everything else), but routes incorrectly to `video-analyze` under Default. Default is the only condition where `video-analyze` appears with its full body inlined while `ad-analysis` appears only as a one-line listing, so the lexical match *Analyze*→***`video-analyze`*** dominates the listing-only match *ad*→***`ad-analysis`***. We use *lexical competition* here to mean surface-form token overlap between the user query and an autoloaded skill's inlined body that pulls planner attention toward that skill, away from a competing skill whose name matches but whose body is absent from the prompt. Partial prompt exposure can therefore create lexical competition that hurts routing relative to either full or none. As a minimal falsification, we renamed `video-analyze`'s SKILL.md `name` field to `video-describe` (a name without the token "Analyze") and reran q3 under Default; the agent still routed to the renamed skill rather than `ad-analysis`, producing the same inline analysis. We read this as evidence that the lexical pull operates over the full inlined body content (including the description, which still contained "*Analyze* videos using Cloudglue API. . . ") rather than the surface skill name alone, and that skill authors cannot reliably fix this kind of failure by renaming alone. Concurrent work on retrieval-based skill routing at ∼80k-skill scale reports the same body-vs-listing asymmetry as a 31–44 percentage-point routing-accuracy drop [13], complementing prior multimodal routing studies [4]. The autoload knob is a static, design-time signal — tractable at the dozens-of-skills scale we operate in; at thousands-of-skills scale, learned retrieval-based routing becomes necessary, and lexical interference like q3's would remain a risk but could be adapted to via training signal in a way that static autoload assignments cannot. A practical implication for skill authors: if a workflow-skill's name overlaps semantically with an autoloaded tool-skill, leaving the workflow-skill on-demand may actively hurt rather than just save context budget.

Failure modes under All-off. Two of the six All-off cells fail outright in qualitatively different ways. Q2 produces no deliverable: after searching seven plausible skill locations on disk, the agent asks the user to manually run `/skills:sales-coaching`. Q5 bypasses the gold tool entirely, escalating to subagent-delegation over cached describe-JSON files at 17 minutes and 83 tool calls. The other All-off cells succeed but pay a 35–170% wall-clock premium for filesystem search. The autoload knob's role on the routing axis is therefore less about

raising the capability ceiling than about reducing the variance of how the agent gets there.

A practical heuristic for skill authoring. The pattern in Table 1 suggests a practical test in this harness for whether a new skill should be authored as a tool or a workflow: *does it produce a single artifact a user can name?* If yes (a meeting breakdown, a YouTube package, a blog post), we found it useful to treat it as a workflow-skill, on-demand with a deliverable template. If no (a search, an extraction, a transcoding), we found it useful to treat it as a tool-skill, autoloaded with no template. In this harness, the autoload pattern aligns with this distinction: workflow-skills fire only when the user requests the deliverable, while tool-skills are vocabulary that has to be present.

Composition. Workflow-skills in this harness are leaves: they call tool-skills, but no workflow-skill calls another. Higher-level workflows (e.g. "quarterly review report from N sales calls") prefer N tool-skill chains over an inner workflow-skill, since the inner deliverable shape is rarely the right shape for the outer's.

Generality. We described the split for multimodal video agents because that is the domain we have shipped. We hypothesize that a similar distinction may arise wherever a fragmented tool surface meets deliverable-oriented users (audio production, GIS, computational biology), though the present evidence is limited to one harness.

References

[1] Anthropic. 2024. Claude Code. Anthropic Documentation. https://code.claude.com/docs.

[2] Anthropic. 2025. Equipping Agents for the Real World with Agent Skills. Anthropic Engineering Blog. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, October 16, 2025.

[3] Kevin Dela Rosa. 2025. RAVEN: An Agentic Framework for Multimodal Entity Discovery from Large-Scale Video Collections. *arXiv preprint arXiv:2504.06272* (2025).

[4] Kevin Dela Rosa. 2025. Smart Routing for Multimodal Video Retrieval: When to Search What. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW).*

[5] Kevin Dela Rosa, Amy Xiao, and Matt Pua. 2025. VideoMCP: MCP-Enabled Video Intelligence for Multimodal Agent Reasoning and Enterprise Video Collection Analysis. In *Proceedings of the 2nd Workshop on Agentic AI for Enterprise: Emerging Applications and Real-world Challenges (KDD AAE).* Toronto, ON, Canada.

[6] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670* (2026).

[7] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560* (2023).

[8] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations (ICLR).*

[9] Timo Schick, Janvier Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS).*

[10] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research* (2024).

[11] Renjun Xu and Yang Yan. 2026. Agent Skills for Large Language Models: Architecture, Acquisition, Security, and the Path Forward. *arXiv preprint arXiv:2602.12430* (2026).

[12] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR).*

[13] YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. 2026. SkillRouter: Skill Routing for LLM Agents at Scale. *arXiv preprint arXiv:2603.22455* (2026).