Research Technical Report FRITS-TR-2026-03 5 July 2026 TOOL USE ROUTING MISTRAL PRODUCTION LLM SYSTEMS

Every Turn Ends in a Tool Call: Reliable Tool Use for Mid-Size Open Models via Two-Stage Forced Routing

Frits Lyneborg

FRITS AI ApS

ABSTRACT

Production assistants want many tools; mid-size open models want few. Operating a Mistral-based assistant with more than twenty tools, we catalogued what actually goes wrong as tool count grows: the model answers from its priors in prose before deciding to call a tool (producing two contradictory answers on screen), invents tool names remembered from other vendors' ecosystems, leaks tool-call JSON into user-visible text, and — in the case of smaller models — fails silently at rates that reached 86% in our burn-in. We describe the architecture that eliminated the worst of these failure classes structurally rather than by prompting: a first stage that always sees exactly four tools and is forced (tool_choice: required) to end every turn in a tool call — one of which, directAnswer, is an empty-schema signal meaning "no lookup needed" — and a second stage of sixteen specialist categories, each seeing at most three tools, enforced by a runtime assertion. Around the model sits what we call guide wheels rather than guardrails: deterministic repair of hallucinated tool names, fallback chains for empty results, and in-flight stream correction. A validation experiment found the forced-call design eliminated pre-tool hallucination entirely (0/10 sessions), left tool selection at 5/5, cost a median +200 ms to first token, and — unexpectedly — made tool-bound requests up to 24% faster end-to-end. We report the design, the measurements, and the costs.

1. The problem: tools degrade the model that uses them

Tool calling is how an assistant becomes useful — search, code execution, document analysis, image generation, charts, weather [1 (#ref-1)]. The natural architecture is to hand the model every tool and let it choose. With a frontier-scale model this mostly works. With the mid-size open models a European-sovereignty deployment builds on — we operate on Mistral's model family — it degrades in specific, reproducible ways as the tool count grows. Our internal design guidance, arrived at empirically: behaviour is clean up to roughly ten tools, and materially unreliable beyond that. With more than twenty tools in the product, we had a problem.

What "unreliable" concretely means, from our production logs and experiments:

Failure class 1 — answering before deciding (the two-bubble bug). With `tool_choice: "auto"`, the model would sometimes commit to a prose answer from its parametric memory *and then* call the relevant tool. The user watches an answer appear, followed by a second, tool-grounded answer that contradicts it. This is the worst failure class because it is user-visible, trust-destroying, and unfixable by prompt: the instruction "always check the knowledge base first" competes with the model's fluency at answering directly, and loses often enough to matter.

Failure class 2 — tools from other people's ecosystems. The model invents tool names it has seen in training data rather than the ones in its schema [3 (#ref-3)]: `codeInterpreter` and `executePython` (other vendors' conventions) for our `runPython`; format-suffixed inventions like `createDocx` for our document tool; lowercase near-misses like `weather` that collide with nothing and go nowhere. These are not random strings — they are coherent memories of *other* APIs, which is what makes them frequent.

Failure class 3 — schema pressure on smaller models. We tested routing on a small model of the same family for cost reasons. Under the full tool schema its time-to-first-token spiked past client timeouts, and the burn-in produced silent failures — no tool call, no text, no error — in 12 of 14 calls (86%). The capability cliff between "supports tool calling" and "supports tool calling under production schema load" is large and, in our experience, not visible on any public benchmark [5 (#ref-5)].

Failure class 4 — the machinery leaks into the answer. Tool-call JSON fragments, speaker labels, and echoes of prior tool metadata intermittently appear in user-visible prose, especially around malformed calls; occasionally the model enters repetition loops (a sign-off line repeated indefinitely).

2. The architecture: two stages, small tool sets, no way out

The design principle that fell out of a month of fighting this: **never show the model many tools, and never let it not choose.** Concretely:

Stage 1 — a router with exactly four tools. Every user turn first hits a routing call [2 (#ref-2)] whose schema contains exactly four tools: `webSearch` (current-events grounding), `webContext` (fetch/ground a specific source), `routeToSpecialist` (hand off to a category, below) [8 (#ref-8)], and `directAnswer`. The call runs with `tool_choice: "required"`: the model *cannot* answer in prose; it must end the turn in exactly one tool call. The two-bubble bug is thereby not discouraged but *structurally impossible* — there is no code path in which prose precedes the tool decision, because prose cannot occur.

The subtle piece is `directAnswer`, the "no lookup needed" path — most turns in a chat product need no tool at all [7 (#ref-7)]. `directAnswer` is a tool whose schema is essentially empty: it carries no answer payload and functions purely as a routing signal. The actual answer is produced by a second, tool-free call that streams token-by-token. This two-trip design exists because of a Mistral-specific behaviour: tool-call arguments are not streamed incrementally but arrive as one block, so putting the answer *inside* the tool call would mean a long silent wait followed by a wall of text — unacceptable in a product whose responsiveness bar is "immediate feedback, always." The empty-schema signal costs one extra round trip on direct answers and buys back real streaming.

Stage 2 — sixteen specialist categories, at most three tools each. `routeToSpecialist` names a category — code, documents, images, charts, diagrams, geography, weather, news, video, knowledge-base and so on, sixteen in total. Each category maps to a small, fixed tool set with a hard ceiling of three tools, enforced by a runtime assertion

that fails loudly in development if any category grows past it. The specialist call sees only its own tools and owns the final answer; any prose the router produced is dropped. The ceiling is a design constraint we treat as load-bearing: when a category wants a fourth tool, the category is split.

Both stages run on the family's flagship model. After the small-model burn-in of Section 1, we consider router duty a flagship workload [9 (#ref-9)]: it is one cheap call, and everything downstream depends on it.

3. Guide wheels, not guardrails

Around the two stages sits a philosophy we found more effective than instruction-stacking: **accept what the model produces, and build the environment so that what it produces works.** We call these guide wheels — in contrast to guardrails, which try to stop the model from straying and grow the prompt (and the latency, and the weirdness of residual failures) with every rule added.

- **Hallucinated-name repair.** A repair hook intercepts unknown tool names and category values and maps them deterministically to the intended real ones — the other-vendor imports, the format-suffixed inventions, the case variants of Section 1 all resolve to their obvious targets instead of erroring. The mapping is only ever applied where the intent is unambiguous; a genuinely unknown name still fails. The model is never scolded for its vocabulary; the vocabulary is absorbed.
- **Fallback chains.** Empty or failed tool results retry along fixed chains (image generation falls back to image search; the two web tools substitute for each other; knowledge-base misses fall back to grounded web context), with query reformulation, capped at three total attempts. Transient emptiness stops being user-visible.
- **In-flight stream correction.** A stream transformer strips leaked tool-JSON fragments, speaker labels and metadata echoes from user-visible text as it streams, and cuts repetition loops. It operates only on patterns that cannot occur in legitimate output.

The unifying rule: corrections must be *deterministic and certain*. Where we know what the model meant, the environment absorbs the difference silently. Anything fuzzy remains a prompt problem — this is not a licence to guess.

4. Validation

Before making forced calling the production default, we ran a controlled comparison: five representative prompts (greeting, trivial arithmetic, a knowledge-base-dependent question, a current-events question, an image request), each under the incumbent `tool_choice: "auto"` configuration and the forced two-stage design, two runs each — small by benchmark standards [6 (#ref-6)], designed as a go/no-go gate on five pre-registered criteria. All five passed:

Criterion	Result
Pre-tool prose hallucination	Eliminated: 0/10 forced-mode sessions (the failure reproduced in auto mode)
Tool selection accuracy	5/5; the forced design chose knowledge-base grounding *more* often, and one answer improved as a result
Time-to-first-token cost	Median +200 ms (range +188 to +301 ms) — within our responsiveness budget
Answer correctness	Equal or better on every prompt
Model breakage under `required`	None: 0 malformed calls, 0 schema errors across all sessions

One result we did not predict: **image requests completed 24% faster** (10.4 s vs 13.6 s end-to-end). Under forced routing, generation halts immediately after the routing call is emitted; in auto mode the model habitually produced post-decision filler tokens that added latency to every tool-bound request. Forcing the call did not just remove a failure mode — it removed a systematic waste we had not noticed we were paying for.

The routing battery has since become reusable infrastructure: a 100+-prompt harness across the four stage-1 tools and all specialist categories, in nine languages, that we re-

run whenever routing-relevant text changes (a later change to steer financial questions toward grounded web answers validated at 18/18 with zero over-steering onto unrelated prompts) — and which doubles as the routing-fidelity phase of the entry exam that any *new* model must pass before serving in the product (see FRITS-TR-2026-01).

5. Costs and limitations

Honesty about the bill. The two-trip `directAnswer` design roughly doubles LLM calls on no-tool turns and adds its round trip to their total latency; we consider this the price of both streaming and the structural no-hallucination guarantee, and it is the design's main standing cost. The +200 ms first-token overhead applies to every turn. The sixteen-category taxonomy is maintained by hand — when tools are added, a human decides the category split; nothing auto-partitions. The specific numbers (ten-tool ceiling, 86% small-model failure rate, latency deltas) are measurements of one model family at one point in time, not constants of nature — expect the ceilings to move with model generations, and re-measure. And the validation experiment was a deliberately small pre-registered gate, not a large-sample benchmark; it earned the production rollout, whose logs have since done the real validating.

What we believe transfers beyond our stack: (1) cap the tools any single call can see, aggressively — the ceiling is lower than the model card implies; (2) if a failure class is intolerable, make it structurally impossible rather than instructionally discouraged — `tool_choice: "required"` plus an explicit "just answer" tool is the cleanest example we know; (3) route hierarchically so tool breadth lives in a taxonomy, not in one schema [4 (#ref-4)]; (4) absorb the model's vocabulary with deterministic repair instead of fighting it with prompt rules. Teams building on mid-size open models — which, for European-sovereignty deployments, is the realistic model class — will meet these failure modes; we hope the catalogue and the recipe save them the month it cost us.

Corrections and prior art pointers are welcome: contact (/contact/).

References

1. Schick, T., et al. — *Toolformer: Language Models Can Teach Themselves to Use Tools.* Meta AI, 2023. arXiv:2302.04761.
2. Yao, S., et al. — *ReAct: Synergizing Reasoning and Acting in Language Models.* Princeton / Google, ICLR 2023. arXiv:2210.03629.
3. Patil, S. G., Zhang, T., Wang, X., Gonzalez, J. E. — *Gorilla: Large Language Model Connected with Massive APIs.* UC Berkeley, 2023. arXiv:2305.15334.
4. Qin, Y., et al. — *ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs.* Tsinghua University (THUNLP) et al., ICLR 2024. arXiv:2307.16789.
5. Li, M., et al. — *API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs.* Alibaba DAMO Academy, EMNLP 2023. arXiv:2304.08244.
6. Patil, S. G., Mao, H., Yan, F., Ji, C. C.-J., Suresh, V., Stoica, I., Gonzalez, J. E. — *The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models.* ICML 2025, PMLR v267, pp. 48371–48392.
7. Chen, L., Zaharia, M., Zou, J. — *FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance.* Stanford University, 2023. arXiv:2305.05176.
8. Ong, I., et al. — *RouteLLM: Learning to Route LLMs with Preference Data.* UC Berkeley / LMSYS, ICLR 2025. arXiv:2406.18665.
9. Ding, D., et al. — *Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing.* Microsoft Research, ICLR 2024. arXiv:2404.14618.

How to cite

Frits Lyneborg (2026). Every Turn Ends in a Tool Call: Reliable Tool Use for Mid-Size Open Models via Two-Stage Forced Routing. FRITS AI ApS, Technical Report FRITS-TR-2026-03. https://frits.ai/research/two-stage-tool-routing/

```
@techreport{lyneborg2026every,
  title       = {Every Turn Ends in a Tool Call: Reliable Tool Use for Mid-Size Open Models via
  author      = {Lyneborg, Frits},
  institution = {FRITS AI ApS},
  number      = {FRITS-TR-2026-03},
  year        = {2026},
  month       = {jul},
```

```
  url         = {https://frits.ai/research/two-stage-tool-routing/}
}
```