Forge: Closing the Agentic Reliability Gap Between Self-Hosted and Frontier Language Models*

Antoine E. Zambelli

***Abstract*—Agentic workflows - multi-step processes where language models call tools, interpret results, and make routing decisions - are a critical capability for deploying AI systems beyond simple question-answering. While frontier API models achieve high accuracy on individual function calls, multi-step workflows expose a compounding reliability problem: even 95% per-step accuracy yields only 77% completion over five steps. We present Forge, an open-source Python framework that adds tool-agnostic guardrails to self-hosted language models running on consumer hardware. Forge's guardrail stack - retry nudges, step enforcement, error recovery, context compaction, and hardware-aware VRAM budgeting - operates independently of the specific tools or workflows being executed. We evaluate Forge across 50+ model/backend configurations using a custom eval harness with 9 agentic scenarios run 50 times each. Our key findings: (1) an 8-billion-parameter model with Forge achieves 99% accuracy on agentic workflows, within 1 percentage point of frontier APIs with the same framework; (2) the same 8B model with Forge outperforms frontier APIs without guardrails - the best configuration a consumer can achieve through API alone; (3) even frontier models drop to 49–87% completion without guardrails, with error recovery scoring 0% universally - an architectural gap, not a capability gap; (4) the serving backend is a hidden variable, with identical model weights producing 0% accuracy on one backend and 78% on another; (5) larger models do not always outperform smaller ones, with 8B models beating 14B variants of the same family across multiple configurations. These findings suggest that the reliability gap between self-hosted and frontier models is primarily a framework problem - guardrails close the mechanical gap, while the intelligence gap between model tiers manifests only in reasoning-heavy scenarios where frontier models excel without assistance.**

I. Introduction

The deployment of language models as autonomous agents - systems that plan, call tools, and act on results [Yao et al.(2023)], [Schick et al.(2023)] across multiple steps - has moved from research prototype to production requirement. However, a fundamental gap exists between single-turn function-calling benchmarks and real-world agentic reliability. Models that score well on individual tool calls often fail to complete multi-step workflows because errors compound: a 90% per-step success rate yields only 59% completion over five steps (0.9^5), and even 95% per-step accuracy drops to 77%. Standard benchmarks, which evaluate single-turn call correctness, do not capture this compounding dynamic [Patil et al.(2025)].

Current approaches to this reliability gap fall into two categories: (1) use frontier API models, which are expensive, raise privacy concerns for sensitive workflows, and - as we show - still require orchestration support for multi-step reliability; or (2) build per-workflow guardrails - hardcoded state machines that ensure correctness for a specific tool set but sacrifice the generality that makes language models valuable.

We present Forge, an open-source framework that takes a third approach: generic, tool-agnostic guardrails that preserve the language model's flexibility to handle arbitrary tool sets while adding the mechanical reliability layer needed for production agentic workflows. Forge supports multiple serving backends (Ollama, llama.cpp / llama-server, Llamafile, and Anthropic's API) [Gerganov(2023)], [Ollama(2023)], [Mozilla Ocho(2023)], [Anthropic(2025)], handles both native function calling and prompt-based tool use, and manages hardware constraints automatically via VRAM-aware context budgeting.

Our evaluation reveals several findings that challenge conventional assumptions: small models with guardrails outperform frontier models without them; the serving backend produces accuracy swings larger than model size differences; and smaller models outperform larger variants within the same family. Notably, frontier models still outperform local models on reasoning-heavy scenarios without any guardrails - suggesting that the gap Forge closes is mechanical reliability, not intelligence.

II. System Architecture

Forge's core design principle is tool-agnostic guardrails: every reliability mechanism operates without knowledge of the specific tools, workflows, or domains being executed. This contrasts with per-workflow approaches where guardrail logic is embedded in the tool definitions or orchestration code.

The framework accepts an arbitrary list of tool definitions (as JSON Schema) and a natural-language task description. The same Forge configuration handles all 9 evaluation scenarios - spanning medical records, incident diagnosis, supplier evaluation, and employee lookups - without any scenario-specific code.

A. The Guardrail Stack

Forge applies five guardrail layers, each independently toggleable.

Retry nudges. When a model produces a malformed tool call (wrong JSON structure, missing parameters, type errors), Forge catches the exception and feeds the error message back to the model as a tool result, prompting self-correction. The nudge includes the specific error (e.g., "TypeError: expected string, got int") but no domain-specific guidance.

*This is a preprint.

Step enforcement. Forge extracts the model's stated plan (e.g., "I will first look up the user, then check permissions, then generate the report") and tracks execution against it. If the model attempts to skip to the terminal tool before completing required intermediate steps, Forge nudges it back. Step tracking is inferred from tool call patterns, not hardcoded per workflow.

Error recovery. When a tool returns an error (via exception or ToolResolutionError; see §2.3), Forge feeds the error back and keeps the corresponding step open. The model sees the error message as a tool result and can retry with different arguments. Without this mechanism, tool errors are invisible to the orchestrator - the step is marked complete with bad data.

Context compaction. Proceeds in tiers: (1) retry and nudge messages are dropped and raw tool results are truncated to their opening lines, (2), tool results are dropped and the model's reasoning traces and tool-call structure are preserved, (3) only the tool-call skeleton survives. Without compaction, workflows on memory-constrained hardware either silently exceed the context window or trigger CPU fallback that degrades inference speed by orders of magnitude.

Hardware-aware VRAM budgeting. Forge queries `nvidia-smi` at startup, calculates available VRAM after model loading, and derives a token budget for the context window. This prevents silent CPU fallback - a failure mode we observed in both Ollama and Llamafile where exceeding VRAM causes inference to silently move to CPU, degrading performance by 10–100x with no warning.

B. ToolResolutionError: A Missing Primitive

We identify a gap in the tool-calling contract used by all current frameworks: there is no distinction between "the tool executed successfully and returned data" and "the tool executed successfully but the requested data does not exist." Both produce a return value; the orchestrator marks the step complete either way.

This is analogous to HTTP's distinction between 200 (success with data) and 404 (valid request, resource not found) - a distinction that does not exist in current LLM tool-calling frameworks. The consequence in multi-step workflows is a silent cascade: a "not found" on step 1 is treated as success, the model fabricates or forwards bad data, and every downstream step is corrupted.

Forge introduces `ToolResolutionError`, a new exception class that signals "valid request, data didn't resolve." The tool author raises it; the orchestrator catches it, feeds the message back to the model, but does not mark the step complete. The model can then retry with different arguments, try an alternative tool, or give up - reasoning about the error naturally. Critically, `ToolResolutionError` does not count toward the crash budget (which tracks malformed calls), preventing conflation of "searching for the right key" with "can't format a valid call."

The name `ToolResolutionError` is deliberate: *resolution* denotes the step between schema validation and successful execution where a syntactically valid argument must resolve against real data. This distinguishes it from Python's `ValueError` (the argument is intrinsically wrong) and `TypeError` (the argument has the wrong type). A query of `"france"` passed to a country lookup is a valid string and a reasonable value - it simply may not match any key in the database. The failure is relational, not intrinsic, and the recovery path (retry with a different key) is fundamentally different from the recovery path for a malformed call (fix the format).

C. Multi-Backend Support

Forge abstracts over four serving backends: Ollama, llama-server (llama.cpp), Llamafile, and Anthropic's API. Each backend exposes different function-calling interfaces - native tool-use APIs, prompt-injection templates, or structured output modes. Forge normalizes these into a common interface while preserving backend-specific optimizations.

This abstraction proved essential: our evaluation shows that the same model weights produce dramatically different results across backends (III), a variable that is invisible in standard benchmarks.

III. Evaluation

A. Eval Harness

We constructed a custom evaluation harness testing 9 agentic scenarios of increasing complexity. Each scenario defines a set of tools, a task, and validation criteria. Scenarios span two difficulty tiers:

Mechanical reliability (4 scenarios): basic 2-step workflows, sequential 3-step chains, error recovery from intentionally malformed inputs, and tool selection from 8 candidates (6 distractors).

Reasoning + reliability (5 scenarios): cross-step argument extraction, 4-step sequential reasoning chains (each step's output feeds the next), conditional routing with optional diagnostic tools, multi-step data gap recovery through dead ends and redirects, and a relevance detection scenario (knowing when not to call a tool).

Each scenario is run 50 times per model configuration. We report six aggregate metrics: score (correct/total), accuracy (correct/completed), completeness (completed/total), efficiency (ideal calls/actual calls), speed (average time per run), and waste (average extra tool calls beyond the ideal). We also report per-scenario score columns to identify which workflow patterns a model struggles with.

Scenarios are implemented in two modes: static (lambda functions returning predetermined results, isolating framework reliability) and stateful (tool state mutates based on arguments, testing model reasoning and framework together).

B. Model Configurations

We tested 50+ configurations spanning:

- Models: Ministral 8B/14B (instruct and reasoning), Qwen3 8B/14B, Llama 3.1 8B, Mistral 7B, Mistral-Nemo 12B, Claude Haiku 4.5, Sonnet 4.6, Opus 4.6 [Jiang

TABLE I

MAIN RESULTS SORTED BY SCORE. N=50 RUNS PER SCENARIO. LAMBDA SCENARIOS (REL–DGR) TEST FRAMEWORK MECHANICS WITH STATIC TOOL RESPONSES. STATEFUL SCENARIOS (REL_S–DGR_S) TEST END-TO-END CORRECTNESS WITH STATE-MUTATING BACKENDS. † = 8B MODEL OUTPERFORMS 14B VARIANT OF SAME FAMILY.

	Aggregate						Lambda Scenarios									Stateful Scenarios								
Model/Backend [Config]	**Scr**	**Acc**	**Cmp**	**Eff**	**Wst**	**Spd**	**rel**	**arg**	**tsl**	**b2s**	**s3s**	**crt**	**srn**	**err**	**dgr**	**rel_s**	**arg_s**	**tsl_s**	**$b2s_s$**	**$s3s_s$**	**crt_s**	**srn_s**	**err_s**	**dgr_s**
Sonnet 4.6 AN/N [reforged]	100.0	100.0	100.0	100	0.1	6.5s	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Opus 4.6 AN/N [reforged]	100.0	100.0	100.0	100	0.1	8.5s	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Haiku 4.5 AN/N [reforged]	99.6	100.0	99.6	96	0.5	4.0s	100	100	100	100	100	100	96	100	100	100	100	100	100	100	100	96	100	100
Ministral 8B-R q4 LS/N [reforged]†	99.3	99.3	100.0	87	0.5	3.7s	100	100	100	100	100	96	100	100	100	100	100	98	100	100	96	100	100	98
Ministral 8B-R q8 LS/N [reforged]	99.2	99.2	100.0	87	0.5	4.6s	100	100	100	100	100	94	100	98	98	100	100	100	100	100	96	100	100	100
Ministral 14B-I q4 LS/N [reforged]	98.8	98.8	100.0	83	0.7	3.5s	100	100	96	100	100	100	100	98	96	100	100	96	100	100	100	100	96	96
Qwen3 14B q4 OL/N [reforged]	96.3	96.3	100.0	85	0.6	19.6s	100	100	100	100	100	100	100	86	82	100	100	100	98	98	100	100	92	78
Qwen3 8B q8 LS/P [reforged]	95.7	95.7	100.0	93	0.3	17.8s	100	100	100	100	100	98	100	72	96	100	100	100	100	100	98	100	72	86
Ministral 14B-R q4 LS/P [reforged]†	95.7	95.7	100.0	98	0.2	3.0s	100	100	100	100	100	90	100	100	80	100	100	100	100	100	82	100	100	70
Haiku 4.5 AN/N [bare+any]	88.9	100.0	88.9	100	0.0	2.7s	100	100	100	100	100	100	100	0	100	100	100	100	100	100	100	100	0	100
Sonnet 4.6 AN/N [bare+any]	88.9	100.0	88.9	100	0.0	5.4s	100	100	100	100	100	100	100	0	100	100	100	100	100	100	100	100	0	100
Opus 4.6 AN/N [bare+any]	88.9	100.0	88.9	100	0.0	7.1s	100	100	100	100	100	100	100	0	100	100	100	100	100	100	100	100	0	100
Opus 4.6 AN/N [bare]	88.6	100.0	88.6	100	0.0	9.0s	100	100	100	100	100	100	100	0	100	100	100	100	100	100	100	100	0	94
Sonnet 4.6 AN/N [bare]	87.2	99.9	87.3	100	0.0	6.8s	100	100	84	100	100	100	100	0	100	100	100	88	100	100	100	98	0	100
Qwen3 8B q8 LS/P [bare]	85.2	97.3	87.6	97	0.1	16.2s	100	100	92	96	100	96	100	0	94	100	100	94	100	94	76	100	0	92
Ministral 14B-R q4 LS/P [bare]‡	81.7	94.0	86.9	100	0.1	3.1s	100	100	100	100	96	78	94	0	56	100	100	96	100	96	74	100	0	80
Ministral 14B-I q4 LS/P [bare]	78.3	95.4	82.1	100	0.0	3.0s	100	86	80	100	84	98	100	0	64	100	88	72	100	86	96	94	0	62
Ministral 8B-R q4 LS/P [bare]‡	67.1	98.7	68.0	99	0.1	2.6s	54	76	6	100	88	90	100	0	90	50	84	10	100	92	92	96	0	80
Haiku 4.5 AN/N [bare]	43.8	100.0	43.8	100	0.0	3.6s	0	92	98	2	100	100	0	0	0	0	96	100	0	100	100	0	0	0

Scenarios: rel = relevance detection (no tool needed); **arg** = cross-step argument extraction; **tsl** = tool selection (8 tools, 6 distractors); **b2s** = basic 2-step; **s3s** = sequential 3-step; **crt** = conditional routing; **srn** = sequential reasoning (4-step chain); **err** = error recovery; **dgr** = data gap recovery (5-step breadcrumb). Subscript $_s$ = stateful variant. **Backends:** AN = Anthropic API; LS = llama-server (llama.cpp); OL = Ollama. **FC mode:** N = native; P = prompt-based. **Model suffixes:** R = reasoning; I = instruct. † = 8B outperforms 14B of same family. ‡ = 14B bare outperforms 8B (reversed from reforged).

et al.(2023)], [Mistral AI(2026)], [Yang et al.(2025)], [Touvron et al.(2023)], [Anthropic(2025)]

- Backends: Ollama (native FC), llama-server (native FC and prompt mode), Llamafile, Anthropic API (native)
- Quantizations: Q4_K_M and Q8_0 for all local models
- Guardrails: Full Forge stack (”reforged”) vs. no guardrails (”bare”) vs. Anthropic's `tool_choice=any`

C. Results

Table I shows the top configurations and key comparison points.

Finding 1: Reforged local models match frontier. Ministral 8B Reasoning with Forge scores 99%, within < 1 percentage point of frontier APIs (Opus, Sonnet, Haiku) at 99-100% with the same framework. The gap between a free, self-hosted 8B model on a $600 consumer GPU and a commercial frontier API is less than a single point.

Finding 2: Reforged local models beat frontier bare. The same Ministral 8B Reasoning with Forge (99%) outperforms Claude Haiku without guardrails (49% completion) and Claude Sonnet without guardrails (87% completion). An 8B local model with a guardrail framework outperforms the best result a consumer can achieve through frontier API alone (Sonnet with `tool_choice=any`: 89%).

Finding 3: Every model needs guardrails - frontier included. Claude Haiku scores drops from 100% to 44% without Forge. Sonnet drops from 100% to 87%. Error recovery scores 0% for every model tested - local and frontier - without the retry mechanism. This is not a model capability gap; it is an architectural absence. No framework feeding errors back to the model means no model can self-correct, regardless of parameter count. Frontier models do, however, maintain high accuracy on reasoning-heavy scenarios (sequential reasoning, conditional routing, data gap recovery) without guardrails - the intelligence gap is real but manifests only where multi-step reasoning, not mechanical reliability, is the bottleneck.

Finding 4: The serving backend is a hidden variable, as highlighted in Table II. The same Mistral-Nemo 12B weights score 7% on llama-server native mode and 83% on llamafile (prompt). Qwen 3 14B scores 96% on Ollama, 93% on llama-server prompt, and 88% with llama-server native. These swings are larger than many model-to-model differences reported in standard benchmarks, yet no published benchmark we are aware of controls for serving infrastructure [Patil et al.(2025)]. Any evaluation of self-hosted model capabilities that does not specify the serving backend may be producing misleading results.

Finding 5: Bigger is not always better. Ministral 8B Reasoning (99%) outperforms Ministral 14B Reasoning (96%) across multiple backend configurations. Qwen3 8B (96%) outperforms Qwen3 14B by up to 5.5% depending on backend. Within the same model family, the smaller variant achieves higher agentic reliability, suggesting that reasoning-oriented fine-tuning at 8B may produce better tool-calling discipline than scale alone at 14B.

D. Compaction Strategy Evaluation

To evaluate context compaction under memory pressure, we designed a 10-step sequential chain where each step's output feeds the next, but data from steps 1–2 is referenced by later steps throughout the workflow. Without token budget constraints, models achieve 96% accuracy on this scenario -

TABLE II
BACKEND COMPARISON FOR THE SAME MODEL WEIGHTS. ALL CONFIGURATIONS USE FORGE GUARDRAILS (REFORGED) AND Q4_K_M QUANTIZATION.

Model	Backend	Scr	Acc	Cmp	Eff	Spd	srn_s	dgr_s
Ministral 8B-R	LS/N	99.3	99.3	100.0	87	3.7s	100	98
	LS/P	98.1	98.1	100.0	92	2.5s	100	88
Qwen3 14B	OL/N	96.3	96.3	100.0	85	19.6s	100	78
	LS/P	93.3	93.3	100.0	91	15.2s	100	74
	LS/N	88.4	88.5	99.9	75	19.2s	78	20
Mistral-Nemo 12B	LF/P	82.6	83.2	99.2	95	4.2s	84	6
	LS/P	75.0	93.5	80.2	84	3.7s	24	74
	OL/N	44.6	62.3	71.6	46	7.9s	44	66
	LS/N	7.2	100.0	7.2	42	2.0s	0	0

Backends: LS = llama-server; OL = Ollama; LF = Llamafile. **FC mode:** N = native; P = prompt-based. srn_s = sequential reasoning (stateful); dgr_s = data gap recovery (stateful).

establishing that failures under compaction are attributable to information loss, not model quality.

We tested three compaction strategies under progressively tighter token budgets (Table III): no compaction (workflow fails when context exceeds budget), sliding window (oldest turns dropped first), and tiered (retry/nudge messages dropped first, then tool results, with tool calls and reasoning traces preserved longest).

TABLE III
COMPACTION STRATEGY COMPARISON ON A 10-STEP WORKFLOW (96% BASELINE ACCURACY WITHOUT BUDGET CONSTRAINTS). BUDGET IS MAXIMUM CONTEXT TOKENS AVAILABLE TO THE MODEL.

Budget	None	Sliding	Tiered
3600 (P1)	0%	58%	**76%**
2200 (P2)	0%	45%	44%
1536 (P3)	0%	15%	18%

Without compaction, every budget-constrained run fails entirely. Tiered compaction recovers 76% of baseline performance at moderate pressure (P1), outperforming sliding window by 18 percentage points. The advantage stems from preserving the model's reasoning traces about early steps - which later steps need for cross-referencing - while discarding the raw tool outputs that informed those conclusions. At extreme pressure (P3), both strategies converge as there is insufficient room for even compressed reasoning. In deployment, compaction is triggered automatically by Forge's hardware-aware VRAM budgeting (II-A), requiring no manual configuration.

IV. CONCLUSION

Forge demonstrates that the reliability gap between self-hosted and frontier models on agentic workflows is primarily a mechanical problem, not a capability problem. Generic, tool-agnostic guardrails bring an 8B model on consumer hardware to within 1 percentage point of frontier APIs - and those same frontier APIs benefit from the same guardrails. The remaining gap between local and frontier models is one of reasoning, not reliability: frontier models excel on complex multi-step reasoning without assistance, while local models match them on the mechanical dimensions that guardrails address.

More broadly, these results suggest that **a production AI solution is not a model - it is a model plus the system around it.** The industry's focus on model capabilities, while important, obscures the degree to which orchestration infrastructure determines real-world performance. Anthropic's Claude Code is not simply Opus with a terminal - it is Opus embedded in a sophisticated harness of retry logic, context management, tool validation, and error recovery. Forge makes the same argument from the self-hosted side: an 8B model that scores 53% bare and 99% reforged is not two different models. It is the same model in two different systems. The 46-point gap is entirely infrastructure.

For practitioners making build-vs-buy decisions, this reframes the question. The choice between self-hosted and frontier should be driven by the reasoning complexity of the target workflow, not by concerns about tool-calling reliability - because reliability is a solved problem at the framework level. All code, evaluation data, and the interactive dashboard are open-source at `https://github.com/antoinezambelli/forge` (public release forthcoming).

ACKNOWLEDGMENTS

Development was assisted by Claude Code (Anthropic) for code generation and iterative debugging.

REFERENCES

[Anthropic(2025)] Anthropic. 2025. Claude Models. https://docs.anthropic.com/en/docs/about-claude/models. Accessed: 2026-03-12.

[Gerganov(2023)] Georgi Gerganov. 2023. llama.cpp. https://github.com/ggml-org/llama.cpp. Accessed: 2026-03-12.

[Jiang et al.(2023)] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023). arXiv:2310.06825 [cs.CL]

[Mistral AI(2026)] Mistral AI. 2026. Ministral 3. *arXiv preprint arXiv:2601.08584* (2026). arXiv:2601.08584 [cs.CL]

[Mozilla Ocho(2023)] Mozilla Ocho. 2023. llamafile: Distribute and run LLMs with a single file. https://github.com/Mozilla-Ocho/llamafile. Accessed: 2026-03-12.

[Ollama(2023)] Ollama. 2023. Ollama. https://github.com/ollama/ollama. Accessed: 2026-03-12.

[Patil et al.(2025)] Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*. PMLR, 48371–48392.

[Schick et al.(2023)] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *Advances in Neural Information Processing Systems* 36 (2023).

[Touvron et al.(2023)] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* (2023).

[Yang et al.(2025)] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388* (2025). arXiv:2505.09388 [cs.CL]

[Yao et al.(2023)] 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)*.