STAB: A Complexity-Ladder Framework for Evaluating Local LLMs as Autonomous Agents

João Conrado V. Nogueira
jconrado@gmail.com

Abstract

Running LLMs as autonomous agents on local hardware is harder than it looks. This paper reports STAB: a four-level complexity ladder (S1→S4) combined with a 12-hour temporal stability battery, built to expose failure modes in on-premise tool-calling agents. We tested eight models from four families (Qwen, Gemma, Llama, Mistral) on a single NVIDIA RTX 3060 (12 GB VRAM). The central finding is that most failures trace back to the instruction and contextual decision layer—not to raw model capacity. Two phenomena stood out: *tool-skipping* triggered by descriptive filenames (H12), and the orthogonality between format compliance and semantic accuracy (H10). `gemma4:e4b` was the only model to pass all four levels without a single failure.

Keywords: Local LLMs, Autonomous Agents, Tool Calling, Instruction Following, Small Language Models, On-Premise Benchmark.

Resumo

O uso de Large Language Models (LLMs) em sistemas agênticos locais enfrenta desafios críticos de consistência e confiabilidade, especialmente em hardware de consumo. Este trabalho apresenta o protocolo STAB, uma metodologia de avaliação em escada de complexidade (S1→S4) combinada a um protocolo de estabilidade temporal para diagnosticar falhas em agentes on-premise. Avaliamos oito modelos de quatro famílias (Qwen, Gemma, Llama e Mistral) em servidor com GPU NVIDIA RTX 3060 (12 GB VRAM). Os resultados revelam que as falhas agênticas originam-se predominantemente na camada de instrução e decisão contextual, e não na capacidade bruta dos modelos. Identificamos o fenômeno de *tool-skipping* por nomeação semântica (H12) e demonstramos que conformidade de formato e acurácia semântica são dimensões ortogonais (H10). O modelo `gemma4:e4b` destacou-se como referência de maior confiabilidade, mantendo desempenho perfeito em todos os níveis de complexidade.

Palavras-chave: LLMs Locais, Agentes Autônomos, Tool Calling, Instruction Following, Small Language Models, Benchmark On-Premise.

1. INTRODUÇÃO

Fazer com que um LLM local invoque ferramentas de forma confiável—e não apenas responda perguntas—é mais difícil do que parece. Modelos que se saem bem em benchmarks padrão frequentemente falham em contextos agênticos: pulam invocações de ferramentas, alucinam resultados ou retornam JSON correto encapsulado em markdown que o script chamador não consegue parsear. Avaliações como BFCL e AgentBench não expõem esses problemas porque operam com *compute* praticamente ilimitado e permitem re-tentativas—condições que não existem em hardware restrito com $T = 0$ e execução one-shot.

Este artigo apresenta o STAB para preencher essa lacuna. As principais contribuições são:

—**Framework STAB:** Metodologia de quatro níveis (S1→S4) que avalia desde o uso básico de ferramentas até o planejamento autônomo de múltiplas etapas, validada por um protocolo de estabilidade temporal.

—**Taxonomia de Falhas:** Classificação de erros agênticos em cinco camadas: protocolo, decisão, formato, grounding e instrução.

—**H12 — Tool-Skipping Semântico:** Identificação empírica da omissão de invocação de ferramenta induzida por nomes de arquivo semanticamente descritivos.

—**H10 — Ortogonalidade:** Evidência de que conformidade de formato e acurácia semântica são dimensões de avaliação independentes.

O artigo está organizado da seguinte forma: a Seção 2 revisa os trabalhos relacionados; a Seção 3 descreve a metodologia STAB; a Seção 4 apresenta resultados e análises; a Seção 5 discute os achados; e a Seção 6 conclui o trabalho.

2. TRABALHOS RELACIONADOS

Benchmarks de agentes e uso de ferramentas. O Berkeley Function Calling Leaderboard (BFCL) [Patil et al. 2025] é o benchmark de referência para avaliação de ferramentas, com foco predominante em modelos em nuvem e LLMs de grande porte. O AgentBench [Liu et al. 2024] propõe avaliação em múltiplos ambientes em oito domínios, identificando o seguimento de instruções como principal obstáculo à autonomia. O SLM-Bench [Pham et al. 2025] introduz métricas de impacto ambiental e configurações de hardware controladas, aproximando-se de nossa metodologia de avaliação local.

Modos de falha e seguimento de instruções. Em [Cemri et al. 2025] é proposta uma taxonomia de 14 modos de falha em sistemas multiagente, destacando a desobediência a especificações. O desempenho de SLMs para function calling em dispositivos de borda é mapeado em [Kavathekar et al. 2025], enquanto [Jhandi et al. 2026] demonstra que SLMs com até 350M parâmetros podem superar modelos maiores no uso de ferramentas com ajuste fino direcionado. Uma queda na conformidade com instruções em modelos menores é documentada por meio do dataset KCIF em [Murthy et al. 2024], e pipelines de treinamento para cenários corporativos que exigem uso determinístico de ferramentas são explorados em [Zeng et al. 2024].

Padrões fundacionais de agentes com ferramentas. O paradigma ReAct [Yao et al. 2023] intercala raciocínio e ação em um mesmo loop de geração, base conceitual da maioria dos agentes de tool-calling atuais. O Toolformer [Schick et al. 2023] demonstrou que LLMs podem aprender a decidir *quando* invocar uma ferramenta de forma auto-supervisionada; nenhum dos dois avalia essa robustez sob restrição de hardware ou execuções repetidas. Uma linha ortogonal à engenharia de prompt ataca a conformidade de formato na camada de decodificação: [Geng et al. 2023] restringe o espaço de tokens gerados a saídas estruturalmente válidas via gramática formal—técnica discutida como mitigação ao anti-padrão AP2 na Seção 5.

Diferenciação. O STAB tem um posicionamento diferente: foca em hardware on-premise, introduz uma bateria de estabilidade temporal e sonda explicitamente vieses contextuais como nomeação semântica—nenhum desses aspectos aparece nos benchmarks acima.

Table I. Ambiente experimental.

Componente	Especificação
CPU	Intel Core i5-10400F (6c/12t)
RAM	32 GB DDR4
GPU	NVIDIA RTX 3060 (12 GB VRAM)
SO	Ubuntu 24.04
Runtime	Ollama (REST `/api/chat`), $T = 0$, `stream=false`

3. METODOLOGIA

3.1 Configuração Experimental

3.2 Framework STAB — Escada de Complexidade

O framework STAB organiza a avaliação em quatro níveis de complexidade, cada um exigindo maior capacidade cognitiva e operacional do agente:

Table II. Escada de complexidade STAB.

Nível	Nome	Ferramenta	Objetivo	K	Métrica
S1	Protocolo	`run_bash`	Invocação + retorno de ferramenta	5	TOOL-ACC
S2	Extração	`run_bash`	Saída JSON estruturada	5	JSON-ACC
S3	Análise de Log	`read_file`	Classificação semântica	4	CLASS-ACC
S4	Multi-log	`read_file`	Triagem e ranking multi-etapa	3	TRIAGE-ACC

O framework utiliza fixtures sintéticas em vez de logs de produção para garantir reprodutibilidade e controle de variáveis. Cada fixture consiste em um arquivo de entrada (.log) e um arquivo JSON de referência (.expected.json). A referência emprega vocabulário controlado para identificação de problemas (e.g., `high_cpu_usage`, `database_unavailable`), transformando a avaliação aberta em classificação e extração rigorosas.

Cada teste segue um protocolo de execução em dois turnos via API REST do Ollama (`/api/chat`). No Turno 1, o modelo recebe o prompt e o JSON Schema das ferramentas, produzindo uma `tool_call` estruturada. O script executa a operação no host e captura o `stdout`. No Turno 2, esse resultado é retornado ao modelo, que gera a resposta JSON final. O determinismo é garantido configurando a temperatura como zero e `stream=false`.

3.3 Modelos Avaliados

Table III. Modelos avaliados.

Modelo	Família	VRAM (MiB)	Tipo	Status Final
`gemma4:e4b`	Gemma4	9,821	Zero-shot	**Referência**
`qwen3.5:9b`	Qwen3.5	7,953	Zero-shot	Aprovado (S2B)
`qwen3:14b`	Qwen3	9,437	Zero-shot	Aprovado (S1–S3)
`llama3.1:8b`	Llama3.1	4,868	Zero-shot	Aprovado (Parcial)
`llama3-groq-tool-use:8b`	Llama3	4,754	Fine-tuned	Aprovado (Parcial)
`mistral:7b`	Mistral	4,458	Zero-shot	Descartado (H14)
`phi4-mini`	Phi4	2,500	Zero-shot	Descartado (H13)
`qwen3.5:2b`	Qwen3.5	3,829	Zero-shot	Descartado (H16)

3.4 Métricas

A avaliação emprega uma escala ordinal de 0 a 4, onde 4 representa uma execução perfeita (invocação correta da ferramenta, saída JSON válida e conteúdo semântico preciso) e 0 representa falha completa.

Definição de falha. *Falha*: execução que não atinge escore máximo (4) em alguma métrica aplicável ao nível. *Falha crítica*—descarte do modelo, interrompe a progressão—é um escore 0 em métrica bloqueante do nível (TOOL-ACC em S1; JSON-ACC em S2), já que o nível seguinte pressupõe o anterior como pré-condição. Falhas não críticas (escore 1–3) mantêm o modelo na progressão e são registradas na taxonomia (Seção 3) e na Tabela VI.

As métricas estão agrupadas em três categorias: métricas de consistência (CON, TOOL-CON) avaliam a estabilidade entre execuções; métricas de instrução e acurácia (INST, TOOL-ACC, JSON-ACC, CLASS-ACC, PARSE-ACC, CHAIN-ACC, TRIAGE-ACC) medem a correção em cada nível STAB; e o escore composto STAB reflete a estabilidade temporal ao longo do protocolo completo de bateria de 12 horas.

Table IV. Métricas de avaliação (escala ordinal 0–4).

Métrica	**Descrição**
CON	Consistência: estabilidade estrutural e decisória entre execuções
TOOL-CON	Consistência de ferramenta: invocações válidas repetidas
INST	Aderência a instrução: conformidade de formato e restrições
TOOL-ACC	Correção funcional: ferramenta correta, execução bem-sucedida
JSON-ACC	Acurácia JSON: validade do schema e precisão de valores
CLASS-ACC	Acurácia de classificação: severidade global correta
PARSE-ACC	Acurácia de extração: host, problema e timestamp precisos
CHAIN-ACC	Acurácia de encadeamento: número correto de chamadas (e.g. 3/3)
TRIAGE-ACC	Acurácia de triagem: ranking de prioridade e regra de escalada
STAB	Estabilidade temporal: manutenção de escores em baterias de 12 h

3.5 Protocolo de Estabilidade Temporal

A métrica STAB (Tabela IV) exige um protocolo operacional específico. Cada modelo aprovado executa o conjunto completo de cenários STAB duas vezes—Bateria 1 e Bateria 2—com intervalo mínimo de 12h, hardware, prompts, temperatura e ordem de cenários idênticos entre elas. O intervalo isola variância técnica temporal (aquecimento da GPU, cache do SO, drift de estado do Ollama entre reinicializações) da variância intrínseca ao modelo. O escore STAB de um cenário é a diferença absoluta entre Bateria 1 e Bateria 2 nessa métrica; STAB=4 indica escore idêntico nas duas. Divergências são investigadas inspecionando os *traces* de raciocínio interno para distinguir instabilidade real de variação estrutural (ver `qwen3:14b` na Seção 4).

3.6 Taxonomia de Falhas

As falhas foram classificadas em cinco camadas com base em sua causa raiz. Falhas na camada de Protocolo envolvem violações estruturais do formato de chamada de ferramenta. Falhas na camada de Decisão indicam erros de raciocínio contextual, como inferir resultados sem executar a ferramenta necessária. Falhas na camada de Formato produzem saídas sintaticamente corretas, mas não conformes. Falhas de Grounding geram respostas estruturalmente válidas, porém factualmente fabricadas. Falhas na camada de Instrução decorrem de interpretação ambígua de ações. Cada falha recebe um código único: códigos com prefixo H indicam achados orientados por hipóteses confirmados por experimentos controlados, enquanto códigos AP marcam padrões anômalos identificados durante execuções padrão.

Table V. Taxonomia de falhas identificadas.

Camada	Tipo	Cód.	Exemplo Concreto	Modelo
Protocolo	Alucinação de Protocolo	H13	JSON em `content` em vez de `tool_calls`	phi4-mini
Protocolo	Echo-tool	H16	Entrega JSON via `run_bash echo`	qwen3.5:2b
Decisão	Tool-Skipping Semântico	H12	Leitura omitida; severidade inferida do nome do arquivo	qwen3:14b
Formato	Markdown Wrapper	AP2	JSON correto encapsulado em ```` ```json ````	qwen3.5:9b
Grounding	Grounding Silencioso	H14	JSON perfeito com dados de hardware fabricados	mistral:7b
Instrução	Ambiguidade de Ação	AP1	Usou `cat` em vez de executar o script	qwen3.5:9b

4. RESULTADOS

4.1 Comparação Geral

Três modelos não passaram pelos níveis iniciais: `phi4-mini` (H13, S1), `qwen3.5:2b` (H16, S2) e `mistral:7b` (H14, S2)—cada um por um motivo estrutural diferente, detalhados a seguir.

O `phi4-mini` falha na camada de *protocolo*: alternou entre serializar o JSON da ferramenta no campo `content` (em vez de `tool_calls`) e ignorar a ferramenta, alucinando um resultado plausível no texto. Por não reconhecer a estrutura esperada da API independentemente da instrução, nenhuma correção via prompt foi tentada—essa classe de falha não é candidata natural à engenharia de prompt.

Para `mistral:7b` e `qwen3.5:2b`, correções *foram* tentadas, com resultados instrutivos. O `mistral:7b` recebeu instrução reforçada em S2B ("use `run_bash` com o comando exato")—continuou sem invocar a ferramenta, mas o prompt refinado eliminou o wrapper de markdown que antes acionava o validador, produzindo um **falso positivo**: JSON perfeito (13/13 campos) com dados fabricados (timestamp de três anos no passado, GPU e RAM inexistentes no servidor). Isso motiva a distinção entre H14 e as demais falhas de protocolo (Tabela V): o validador de schema sozinho não capturaria a falha. O `qwen3.5:2b` recebeu instrução igualmente direta, testada via chamada direta à API—o wrapper de markdown persistiu, sugerindo causa no template de chat ou treinamento, não ambiguidade corrigível em inferência.

Dos cinco que prosseguiram, o `gemma4:e4b` foi o único a passar em todos os níveis sem falha (S1→S4; ver Tabela VI). A queda mais acentuada ocorreu no S4: modelos que haviam tratado S1–S3 sem problemas colapsaram nas tarefas de triagem multi-etapa, apontando para um teto na camada de decisão que tarefas de etapa única simplesmente não revelam.

Table VI. Escores STAB por modelo e nível (0–4; **negrito** = execução perfeita).

Modelo	S1	S2	S3	S4-A	S4-B	S4-C	S1→S4
`gemma4:e4b`	**4**	**4**	**4**	**4**	**4**	**4**	**Sim**
`qwen3.5:9b`	4	4 (S2B)	2[a]	4	2	2	Não
`qwen3:14b`	4	4	4	0	0	0	Não
`llama3.1:8b`	4	4	2	4	2	3	Não
`llama3-groq-tool-use:8b`	4	4	4	4	2	2	Não
`qwen3.5:2b`	4	desc. H16	—	—	—	—	Não
`mistral:7b`	4	desc. H14	—	—	—	—	Não
`phi4-mini`	desc. H13	—	—	—	—	—	Não

[a] `qwen3.5:9b` atingiu CLASS-ACC=4 no S3-mixed, mas CON=INST=2 devido ao markdown wrapper persistente, ilustrando a ortogonalidade das dimensões (H10).

4.2 Principais Achados

H12 — Tool-Skipping Semântico. O modelo `qwen3:14b` omitiu a leitura de arquivos em cenários S4 quando os nomes eram semanticamente descritivos (e.g., `critical-001.log`), inferindo a severi-

dade sem executar a ferramenta. No experimento controlado H12 com nomes de arquivo opacos, o modelo invocou corretamente a ferramenta, confirmando a hipótese de viés contextual. **Mitigação:** utilizar nomes de arquivo opacos ou instruções explícitas de leitura obrigatória.

H10 — Ortogonalidade de Formato e Acurácia Semântica. Conformidade de formato e acurácia semântica mostraram-se dimensões independentes. Um modelo pode produzir saída semanticamente perfeita encapsulada em formato inválido (markdown), ou um JSON estruturalmente válido contendo dados alucinados. Isso foi exemplificado pelo `qwen3.5:9b`, que manteve alta acurácia semântica enquanto falhava nas restrições de formatação.

gemma4:e4b como Modelo de Referência. O `gemma4:e4b` foi o único modelo a atingir escores perfeitos em todos os níveis (S1→S4) com STAB=4 e throughput de 73,0 tok/s. Seu raciocínio interno (*thinking*) foi ativado seletivamente para cenários de desambiguação de alta complexidade (S4-C, como desempate baseado em timestamp), indicando escalonamento cognitivo adaptativo.

4.3 Throughput e Latência

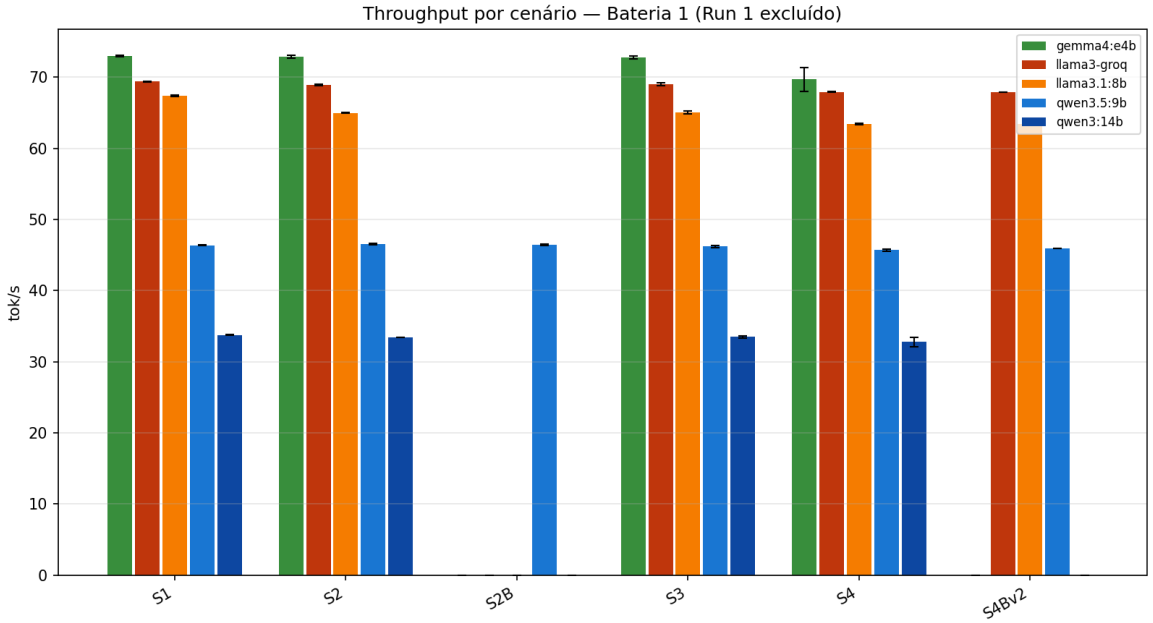

Fig. 1. Throughput médio por modelo e cenário (tok/s), Bateria 1. Execução 1 excluída (cold start). Variação $< 0{,}2\%$ entre baterias confirma inferência limitada pelo hardware.

A variação entre baterias ficou abaixo de 0,2% para todos os modelos, confirmando que o throughput é limitado pelo hardware, não pelo prompt. A Execução 1 de cada bateria foi excluída das médias (penalidade de cold start). Três clusters de desempenho emergiram: `gemma4:e4b` e a família Llama no topo, `qwen3.5:9b` no meio e `qwen3:14b` na base—sua latência no S2 inflada por um extenso trace de raciocínio interno.

4.4 Estabilidade Temporal

Todos os cinco modelos aprovados completaram a Bateria 2 sem degradação: decisões agênticas e acurácia semântica mantiveram STAB=4 ao longo do protocolo. A única anomalia foi o `qwen3:14b`, cujo volume de raciocínio interno variou 7% entre baterias—mas a saída final foi idêntica em ambos os

casos, de modo que isso representa uma peculiaridade estrutural do chain-of-thought, não instabilidade real.

5. DISCUSSÃO

Falhas são instrucionais, não baseadas em capacidade. Dois experimentos corretivos neste estudo contam a mesma história. Uma instrução de formato explícita recuperou o escore S2B do `qwen3.5:9b` de 2 para 4. Uma diretiva de leitura obrigatória eliminou completamente o tool-skipping do `qwen3:14b`. Nenhuma das correções exigiu trocar o modelo ou fazer retreinamento—apenas o prompt. O tamanho do modelo é portanto um proxy fraco para confiabilidade agêntica: modelos acima de 8B claramente possuem a lógica subjacente; o que os quebra é o desalinhamento de contexto, e isso é corrigível na camada de instrução.

Fine-tuning vs. zero-shot em ambientes on-premise. O confronto entre `llama3-groq-tool-use:8b` (fine-tuned) e `llama3.1:8b` (zero-shot) produziu um resultado dividido. No S3-mixed, que exige aplicar regras de precedência binária, o modelo fine-tuned venceu com folga (Groq=4 vs. Llama=2)—especialização funciona para conformidade estrutural em etapa única. Mas no S4-B e S4-C, ambos falharam. Isso coloca uma pergunta concreta: o fine-tuning para uso de ferramentas melhora a aderência estrutural sem tocar no raciocínio multi-etapa necessário para decisões encadeadas? Os dados aqui sugerem que sim—e isso é uma limitação real dos pipelines de ajuste fino atuais.

Recomendações práticas. Em hardware com 12 GB de VRAM, `gemma4:e4b` é a escolha óbvia: consistência perfeita, 73 tok/s, sem necessidade de engenharia de prompt. Abaixo de 6 GB, Llama3.1 funciona—mas precisa de uma camada fina de pós-processamento para tratar falhas de formato. O resultado H12 tem também um ângulo de segurança: nomes de arquivo descritivos permitem que o modelo infira respostas e pule a chamada de ferramenta, o que significa que a ferramenta nunca é de fato executada. Usar identificadores opacos, ou acrescentar `[LEITURA OBRIGATÓRIA]` à instrução, fecha essa brecha.

Decodificação com restrição gramatical como mitigação agnóstica ao modelo. O AP2 (wrapper de markdown) e a resistência à instrução do `qwen3.5:2b` (Seção 4) são falhas de *formato*, não de conteúdo—o modelo "sabe" a resposta, mas a emite de um jeito que o parser não aceita. Engenharia de prompt resolve esse problema só parcialmente (recuperou o `qwen3.5:9b`, não o `qwen3.5:2b`). Geração restrita por gramática formal [Geng et al. 2023]—via GBNF (`llama.cpp`), Outlines ou `format=json` do Ollama—ataca a camada de *decodificação* em vez da de entrada, tornando a violação de formato impossível independentemente da cooperação do modelo. É candidata natural exatamente para os casos resistentes a instrução direta, como o `qwen3.5:2b`. O STAB não testou essa técnica (Seção 6), mas os dados aqui apontam o caso de uso onde ela seria mais valiosa.

Limitações. O estudo rodou em uma única configuração de hardware e utilizou apenas fixtures sintéticas. São restrições reais, mas que também tornaram possível a reprodutibilidade total e o isolamento de variáveis—um trade-off deliberado. O número de execuções por nível também é reduzido (K=5/5/4/3; Tabela II), limitando o poder estatístico e favorecendo a leitura dos resultados como diagnóstico qualitativo, não como taxa de erro com significância estatística. O resultado mais relevante operacionalmente a levar adiante é o H14 (grounding silencioso no Mistral): saída estruturalmente perfeita que é factualmente errada não será capturada por um parser JSON. Validadores semânticos externos não são opcionais em produção—são a única proteção contra esse modo de falha específico.

6. CONCLUSÃO

A confiabilidade de agentes locais é um problema de três variáveis: robustez do modelo, clareza de instrução e neutralidade de contexto. As três precisam funcionar simultaneamente—a falha em qualquer uma delas quebra a cadeia. O `gemma4:e4b` foi o único modelo neste estudo a manter as três

em equilíbrio em todos os níveis, mostrando que arquiteturas MoE quantizadas cruzaram um limiar prático para trabalho agêntico estruturado em hardware de consumidor.

As falhas nos modelos de 8B–14B foram, quase sem exceção, corrigíveis apenas com engenharia de prompt—sem retreinamento. Esse é um resultado encorajador para quem constrói agentes sob restrições de hardware.

O que este estudo não cobriu: logs de produção reais, perfis de hardware diversificados e decodificação com restrição gramatical como alternativa ao enforcement de formato baseado em prompt. Fixtures sintéticas (Seção 3) garantiram reprodutibilidade, mas não capturam o ruído de logs reais—timestamps malformados, encoding inconsistente, entradas truncadas, múltiplos problemas concorrentes. O nível S4 (triagem multi-log), já o de queda mais acentuada mesmo com fixtures limpas (Seção 4), é o candidato mais provável a se degradar ainda mais sob esse ruído—validação direta para trabalho futuro. A bateria temporal de 12 horas também deixa em aberto se deployments mais longos expõem padrões de degradação que não foram visíveis aqui. Essas são as extensões naturais—não por convenção, mas porque os dados deste estudo levantam essas questões específicas.

REFERENCES

Cemri, M. et al. Why do multi-agent LLM systems fail? arXiv preprint arXiv:2503.13657, 2025.

Geng, S., Josifoski, M., Peyrard, M., and West, R. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*. Association for Computational Linguistics, Singapore, 2023.

Jhandi, P. et al. Small language models for efficient agentic tool calling: Outperforming large models with targeted fine-tuning. In *Proceedings of the 40th AAAI Conference on Artificial Intelligence*. AAAI Press, Singapore, 2026.

Kavathekar, I. et al. Small models, big tasks: An exploratory empirical study on small language models for function calling. In *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering (EASE 2025)*. ACM, Istanbul, Turkey, 2025.

Liu, X. et al. AgentBench: Evaluating LLMs as agents. In *Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)*. OpenReview.net, Vienna, Austria, 2024.

Murthy, R. et al. KCIF: Knowledge-conditioned instruction following. arXiv preprint arXiv:2410.12972, 2024.

Patil, S. G. et al. 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 (ICML 2025)*. PMLR, Vancouver, Canada, 2025.

Pham, N. T. et al. SLM-Bench: A comprehensive benchmark of small language models on environmental impacts. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Association for Computational Linguistics, Suzhou, China, 2025.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*. Curran Associates, Inc., New Orleans, USA, 2023.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*. OpenReview.net, Kigali, Rwanda, 2023.

Zeng, G. et al. Adaptable and precise: Enterprise-scenario LLM function-calling capability training pipeline. arXiv preprint arXiv:2412.15660, 2024.