BABELARENA: A LARGE-SCALE MULTILINGUAL BENCHMARK FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly execute multi-step workflows through tool use and interaction with users and environments. However, current agent evaluations are largely English-centric, limiting our understanding of agent capabilities in multilingual settings. We introduce BABELFLOW, a benchmark-general agentic workflow that adapts existing agent benchmarks to new languages by analyzing runtime dependencies, coordinating structure-preserving translation, and combining multi-layer verification with human review to preserve task and evaluation semantics. Using BABELFLOW, we construct BABELARENA, a large-scale multilingual benchmark comprising 16,146 instances derived from 702 canonical tasks across four benchmark families, 13 domains, and 23 languages. Experiments with five frontier models show that no single model dominates across benchmark families and that cross-language disparities extend well beyond task success. Lower-resource languages exhibit distinct failure patterns, with larger shares of tool-use and control-flow errors rather than answer-quality errors alone, pointing to gaps in reliable task execution across the resource levels of these languages. On the same tasks, agents in low-resource languages also consume substantially more tokens than in English (up to roughly twice the input) without proportional increases in interaction length, and language consistency degrades further on tasks requiring structured output, where switches are directed overwhelmingly toward English. We believe BABELARENA provides a foundation for advancing research on reliable and efficient multilingual agents.

1 INTRODUCTION

Large language models (LLMs) are increasingly used as agents that go beyond answering questions to execute multi-step workflows through tool use and interaction with users and environments (Kuang et al., 2026). To serve users in different languages, agents need to do more than produce fluent responses (Liu et al., 2024; Zhou et al., 2024; Patil et al., 2025). They must interpret requests correctly, act on tool feedback, and maintain task constraints throughout the workflow (Luo et al., 2026; Nguyen et al., 2026). Therefore, evaluation must assess whether agents can complete these workflows reliably across languages, rather than judging response fluency alone.

However, existing agent benchmarks provide limited support for such multilingual end-to-end evaluation, as most remain concentrated in English or Chinese (Hofman et al., 2026; Kim et al., 2026; Caciolai et al., 2026). Recent multilingual benchmarks attempt to address this gap, but most adapt only selected components of the source benchmarks and cover a limited range of languages and domains (Hofman et al., 2026; Kim et al., 2026; Caciolai et al., 2026; Luo et al., 2026; Nguyen et al., 2026; Li et al., 2026) (Table 1). Extending this coverage is particularly challenging because agent benchmarks are not self-contained input–output datasets, but executable environments composed of multiple interacting components (Zhou et al., 2024; Yao et al., 2024; Patil et al., 2025). Across these components, task instructions, tools, databases, simulated users, and evaluators are connected through shared entities and constraints that must remain consistent. Translating these components independently can break cross-component consistency, causing execution or scoring failures even when the agent behaves correctly (Kim et al., 2026; Luo et al., 2026; Nguyen et al., 2026). Reliable cross-language comparison therefore requires coordinated adaptation that preserves these dependencies while keeping task and evaluation semantics aligned across languages.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Comparison of multilingual agent benchmarks. ✓/✗ denote adapted/unadapted components: system, simulator, and evaluation prompts (*Prompt*), task inputs (*Task*), tool descriptions (*Tool*), and environment content (*Env.*). *Tasks* counts distinct tasks; *Dom.*, domains; *Lang.*, languages; and *Inst.*, total instances across languages. Boldface marks BABELARENA.

	Adapted Components				Statistics			
Benchmark	Prompt	Task	Tool	Env.	Tasks	Dom.	Lang.	Inst.
MAPS (2026)	✗	✓	✗	✗	805	4	12	9,660
GAIA-v2-LILT (2026)	✓	✓	✗	✗	165	1	6	990
OmnilingualGAIA2 (2026)	✓	✓	✗	✓	640	1	11	7,040
MLCL (2026)	✗	✓	✗	✗	200	1	4	800
SEATauBench (2026)	✓	✓	✓	✓	278	3	6	1,668
PolyWorkBench (2026)	✓	✓	✗	✓	67	5	10	67
BABELARENA	✓	✓	✓	✓	**702**	**13**	**23**	**16,146**

To address these challenges, we introduce BABELFLOW, a benchmark-general agentic workflow for adapting existing agent benchmarks to new languages while preserving task and evaluation semantics. A coding agent first analyzes runtime dependencies to identify language-bearing fields and distinguish translatable content from execution-critical values. Guided by these dependencies, a translator agent translates fields consistently across components and reuses canonical translations for shared entities, while the workflow reconstructs the translated fields in their original structures. Independent verifier agents then check structural integrity, cross-component consistency, semantic fidelity, and behavioral equivalence, with human reviewers adjudicating their findings and guiding revisions before a language variant of the benchmark is accepted.

Applying BABELFLOW to VITABENCH (He et al., 2025), DEEPPLANNING (Zhang et al., 2026), τ^3-BENCH (Yao et al., 2024; Barrès et al., 2025; Shi et al., 2026), and CLAWEVAL (Ye et al., 2026), we construct BABELARENA, a multilingual benchmark covering interactive services, long-horizon planning, knowledge-grounded tool use, and heterogeneous autonomous tasks. With 702 distinct tasks across 13 domains and 23 languages, BABELARENA offers more tasks than most prior benchmarks, the broadest language and domain coverage, and the largest total of 16,146 task instances among the benchmarks compared in Table 1. Task identities and source evaluation criteria remain fixed across languages, enabling controlled comparisons of agent performance.

We first validate BABELFLOW through a controlled ablation on 40 tasks adapted into Chinese, finding fewer broken references and behavioral mismatches than with independent field translation (Section 4.1). We then evaluate five frontier models on BABELARENA and find that no single model leads across all four benchmark families, while all five models have lower average task success and language consistency in the low-resource group than in the high-resource group (Section 4.2). Among sampled VITABENCH failures, Thai and Tamil show larger shares of tool-use and control-flow errors than English and Chinese, suggesting that cross-language disparities extend beyond answer quality to execution reliability (Section 5). On the same VITABENCH tasks, low-resource trajectories consume 1.69–$2.06\times$ as many total input tokens as their English counterparts with similar numbers of turns, while shopping tasks in DEEPPLANNING also incur more turns and tool calls. English accounts for 91.2% of annotated switches among sampled language-inconsistent DEEPPLANNING and CLAWEVAL trajectories, with inconsistency concentrated in assistant responses and textual prefixes before tool calls, respectively.

Our contributions are threefold:

- We introduce and validate BABELFLOW, a benchmark-general agentic workflow for adapting agent benchmarks across languages while preserving the semantics (Section 3.2 and Section 4.1).
- We construct BABELARENA, a multilingual agent benchmark with aligned tasks and evaluation criteria across languages (Section 3.3).
- We evaluate five frontier models, revealing language-dependent differences in task success, language consistency, failure patterns, and execution costs (Section 4.2 and Section 5).

2 Related Work

Agent Evaluation Agent benchmarks increasingly measure closed-loop behavior in executable environments rather than isolated final answers (Liu et al., 2024; Zhou et al., 2024; Patil et al., 2025). Our four source benchmarks cover interactive services, constrained planning, knowledge-grounded tool use, and autonomous tasks (He et al., 2025; Zhang et al., 2026; Yao et al., 2024; Barrès et al., 2025; Shi et al., 2026; Ye et al., 2026).

Multilingual Evaluation Conventional multilingual benchmarks use self-contained input–output pairs (Bandarkar et al., 2024; Kulkarni et al., 2025). Agent benchmarks also expose language through prompts, tools, environments, and evaluators. MAPS and MLCL mainly translate task inputs (Hofman et al., 2026; Luo et al., 2026), while GAIA-v2-LILT also adapts evaluation (Kim et al., 2026). PolyWorkBench and OmnilingualGAIA2 additionally localize environments and evaluators, with the latter calibrating its multilingual verifier (Li et al., 2026; Caciolai et al., 2026). SEATauBench adapts all components but covers fewer domains and languages than BabelArena (Nguyen et al., 2026).

Software Localization Software localization separates translatable resources from program logic (Pirrone & D'Ulizia, 2024; Xia et al., 2013; Wang et al., 2013). Its practice further treats validation as more than a fluency check: internationalization testing examines whether software works properly in a specific language and region (Couto et al., 2025; Felipe et al., 2024).

Ours BabelFlow adapts interdependent benchmark components while preserving execution and evaluation semantics. Applied to four benchmark families, it produces BabelArena, with the broadest language and domain coverage and the most instances among benchmarks in Table 1.

3 Adapting Agent Benchmarks From English to Multilingual

We first outline the challenges of adapting agent benchmarks across languages (Section 3.1). We then introduce BabelFlow, an agentic workflow that combines structured translation with multi-layer verification and human review (Section 3.2). Finally, we apply BabelFlow to construct BabelArena (Section 3.3).

3.1 New Era, New Challenges

Multilingual chatbot benchmarks are commonly built by translating self-contained input–output pairs to reduce authoring costs. However, agent benchmarks link execution-critical text across tasks, tools, databases, simulated users, and graders. Adapting these executable environments introduces a few new challenges:

- **(C1) Executability Preservation**: Like software localization, agent benchmark adaptation must preserve executability. Therefore, we must examine their code, data, and evaluators to determine what can be safely translated.
- **(C2) Translation Consistency**: Shared entities and values, such as product names and tool names, must remain consistent across databases, tool responses, and ground truth. Otherwise, execution may fail, or correct agent behavior may be scored as incorrect.
- **(C3) Evaluation Validity**: Output parsers designed for English may fail when other languages use different word forms or word order. Translating instructions for LLM-simulated users can change their behavior, while LLM judges vary in accuracy and strictness across languages.
- **(C4) Evaluation Efficiency**: Multi-turn rollouts, tool execution, environment simulation, and model-based judging make each evaluation costly. Multiplying this cost across languages can make exhaustive evaluation infeasible.

C1–C3 motivate the adaptation framework (Section 3) to preserve each language variant's fidelity, while C4 guides benchmark composition (Section 3.3) to control multilingual evaluation cost.

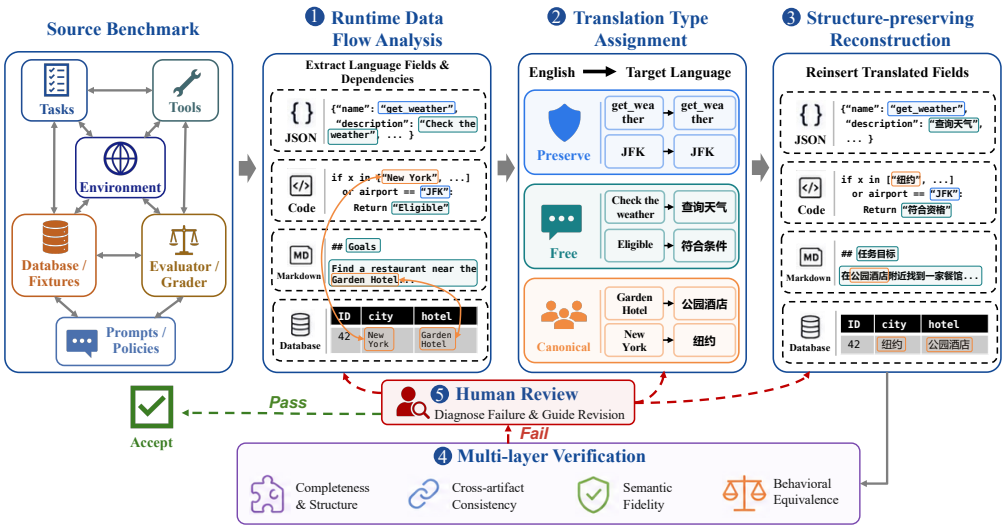


Figure 1: Overview of BABELFLOW. Runtime data-flow analysis identifies text fields and dependencies for translation by type and reconstruction in their original structures. Multi-layer verification checks adaptation quality. Human review guides revisions to analysis and translation.

3.2 BABELFLOW: AGENTIC WORKFLOW ADAPTING BENCHMARKS BEYOND ENGLISH

Given a source benchmark $\mathcal{B}^s$ in language ℓ_s and a target language ℓ_t, BABELFLOW constructs an independently executable variant $\mathcal{B}^t$ in ℓ_t. As shown in Figure 1, the analyzer $\mathcal{A}$ (Claude-Opus-4.8) identifies text fields and cross-artifact dependencies and assigns translation types (Steps 1 and 2), while the translator $\mathcal{T}$ (Qwen-3.7-Plus) translates the fields for reconstruction in their original structures (Step 3). The verifier $\mathcal{V}$ comprises three heterogeneous agents (GPT-5.6-Sol, Qwen-3.7-Max, and Claude-Opus-4.8) that independently check the variant across multiple layers and pool their findings (Step 4). Human reviewers triage these findings and guide revisions by $\mathcal{A}$ and $\mathcal{T}$ (Step 5). The variant $\mathcal{B}^t$ is accepted only after every layer passes and a reviewer signs off. Steps 1–3 establish C1 and C2 by construction, and Steps 4 and 5 re-test them together with C3 by execution.

Step 1: Runtime data flow analysis The analyzer $\mathcal{A}$, a coding agent, reads and runs the source benchmark to trace its evaluation data flow and inventory every text field accessed at runtime, whether in data files or code: task specifications, policies and system prompts, simulated-user instructions, tool documentation and schemas, databases and fixtures, tool observations, output parsers, rubrics, judge prompts, and grader constants. It then links fields that must agree across artifacts, such as product names shared by requests, database rows, tool responses, and ground truth, or category labels shared by task prose and evaluator lexical matchers. These dependencies determine which fields can be safely translated. The analysis yields a typed intermediate representation $\mathcal{I}^s = \mathcal{A}(\mathcal{B}^s) = \{e_i^s\}_{i=1}^N$, where each element e_i^s represents a source-language text field and records its source string, artifact location, and the fields it must agree with.

Step 2: Translation type assignment The analyzer $\mathcal{A}$ assigns each element $e_i^s \in \mathcal{I}^s$ a translation type t_i to guide the translator $\mathcal{T}$ when building the target variant:

- **PRESERVE:** Keep byte-identical, with a declared reason that is either *executable*, covering tool and argument names, schema keys, IDs, enum codes, dates, and evaluator control tokens, or *source asset*, covering reference documents, images, audio, video, and other task-defining source media.
- **TRANSLATE-FREE:** Translate in local context, covering instructions, policies, personas, descriptions, and rubric prose.
- **TRANSLATE-CANONICAL:** Translate once and reuse, covering cities, products, display labels, weekdays, and evaluator-matched terms.

The PRESERVE reasons distinguish execution-critical values, which must remain unchanged for the benchmark to run, from source assets whose translation would change the task. Recording the

reason makes exemptions auditable and distinguishes retained assets from incomplete translations. Only source assets count against a variant's localization coverage. These types guide $\mathcal{T}$ in Step 3: translate ordinary prose, reuse canonical translations across artifacts, and retain execution-critical values and declared source assets.

Step 3: Structure-preserving field extraction and reconstruction To protect artifact structure and out-of-scope content such as JSON keys and code, BABELFLOW passes only extracted fields to the translator $\mathcal{T}$. It copies the source artifact, extracts fields marked for translation, and uses placeholders to protect identifiers, templates, markup, and exact-format examples. The translator renders free prose in ℓ_t and reuses shared-entity translations from the canonical translation map $\mathcal{C}$. The results are written back to their original positions in the copied structure, preserving keys, nesting, and executable values. Translating each element e_i^s according to its type t_i yields its target-language counterpart e_i^t and the representation $\mathcal{I}^t = \mathcal{T}(\mathcal{I}^s, \ell_t) = \{e_i^t\}_{i=1}^N$. Reconstruction following cross-artifact dependencies produces an adapted benchmark $\mathcal{B}^t$ that executes in ℓ_t.

Step 4: Multi-layer verification Translation fluency alone does not establish benchmark validity. Three agents backed by GPT-5.6-Sol, Qwen-3.7-Max, and Claude-Opus-4.8 form the verifier $\mathcal{V}$: each independently executes $\mathcal{B}^s$ and $\mathcal{B}^t$ side by side and checks four layers:

- **Completeness and structure.** Required artifacts exist, parse correctly, and retain source keys, types, nesting, and placeholders.
- **Cross-artifact consistency.** Protected values remain unchanged, shared entities use consistent translations, and references resolve across tasks, tools, databases, parsers, and graders.
- **Semantic fidelity.** Translations preserve intent, constraints, and information, with particular attention to fields affecting task difficulty or scoring.
- **Behavioral equivalence.** The verifier compares tool results, state changes, terminal states, parser outputs, and grader decisions under identical controlled actions in both environments.

The agents do not see one another's reports. Their findings are merged into a failure set $\mathcal{F} = \mathcal{V}(\mathcal{B}^s, \mathcal{B}^t)$, retaining every issue flagged by any agent. Each entry records the offending field, verification layer, reasoning, and side-by-side evidence. The first two layers re-check C1 and C2 on the built variant, where mistyped fields may still parse and read fluently. The last two target C3.

Step 5: Human review Reviewing $\mathcal{B}^t$ line by line is infeasible because its interdependent text fields span tasks, prompts, tools, databases, and evaluators, and many defects surface only at runtime. The verifier group identifies candidate defects and provides reasoning. Human reviewers confirm findings before revision because verifiers can misjudge. Reviewers inspect every failure in $\mathcal{F}$, discard false alarms, and trace confirmed defects to their source stage to produce revision guidance $\mathcal{R}$: mistyped or missing elements return to the analyzer $\mathcal{A}$ (Steps 1 and 2), while fidelity or canonical-consistency errors return to the translator $\mathcal{T}$ with updates to the canonical translation map $\mathcal{C}$ (Step 3). Analysis, translation, verification, and review repeat until $\mathcal{F} = \emptyset$, after which reviewers spot-check a random sample of tasks end to end and sign off on the variant.

3.3 BABELARENA: AGENT BENCHMARK FOR MULTILINGUAL AGENTS

Benchmark Overview We instantiate BABELFLOW on four source benchmarks with complementary interaction structures and target capabilities: VITABENCH for versatile interactive tasks in food delivery, in-store consumption, online travel and cross-scenario; DEEPPLANNING for long-horizon travel and shopping planning; the text component of τ^3-BENCH for simulated-user interaction, tool calling, knowledge retrieval, and domain-policy compliance; and CLAWEVAL for service orchestration, multimodal perception and generation, and professional dialogue. To control the evaluation cost (C4), we randomly select 50% tasks from each domain of the benchmark families. The resulting BABELARENA release covers 702 canonical task identities and 16,146 instances (Table 1), each benchmark spanning the same 23 languages (Table 2). Our selected languages cover 15 scripts and high-, medium-, and low-resource levels, capturing variation in writing direction, word delimiters, and glyph shaping.

Evaluation We retain each source benchmark's scoring procedure, translating evaluation prompts as needed, and run three independent trials per task. From task-completion rewards, we report

Table 2: The 23 languages in BABELARENA, grouped by LLM resource level. Codes combine the ISO 639-3 language tag with the ISO 15924 script tag.

High resource (7)		Medium resource (8)		Low resource (8)	
English	eng_Latn	Korean	kor_Hang	Belarusian	bel_Cyrl
Chinese	zho_Hans	Indonesian	ind_Latn	Eastern Panjabi	pan_Guru
Japanese	jpn_Jpan	Hindi	hin_Deva	Kazakh	kaz_Cyrl
French	fra_Latn	Thai	tha_Thai	Khmer	khm_Khmr
Russian	rus_Cyrl	Hebrew	heb_Hebr	Burmese	mya_Mymr
Arabic	arb_Arab	Turkish	tur_Latn	Lao	lao_Laoo
Spanish	spa_Latn	Vietnamese	vie_Latn	Tamil	tam_Taml
		Malay	zsm_Latn	Telugu	tel_Telu

Pass^1, the average single-trial success rate, and Pass^3, the fraction of tasks solved in all three independent trials. For each trajectory τ, Gemini-3.7-Flash assigns $\mathrm{LC}(\tau, \ell_t) = 1$ if the agent's *user-visible text* consistently uses the target language ℓ_t, and 0 otherwise. This check covers assistant responses, textual prefixes before tool calls, and natural-language content in deliverables (Appendix A). We average LC across trials and report all three metrics separately.

Quality Assurance We assess translation quality by randomly sampling 200 translatable texts per language, yielding 4,600 records across the 23 languages. Each translation is scored on a scale from 1 to 5 by human language experts for the seven high-resource languages and by GPT-5.6-Sol for the sixteen medium- and low-resource languages. The mean scores are 4.28, 4.02, and 3.91 for high-, medium-, and low-resource languages, respectively. Further details are provided in Appendix B.

4 EXPERIMENTS

We first validate BABELFLOW by measuring adaptation defects and construction token costs (Section 4.1). We then evaluate five frontier models on BABELARENA, comparing task performance and language consistency across benchmark families and language resource groups (Section 4.2).

4.1 VALIDATING BABELFLOW

Setup We adapt 40 canonical tasks (10 per benchmark family) into Chinese, comparing *independent field translation*, *canonical translation* reusing $\mathcal{C}$, and BABELFLOW, which adds multi-layer verification and human-guided revision. We report the percentage of tasks affected by each defect type: *broken references* (inconsistent entities or unresolved references) and *behavioral mismatches* (inconsistent tool, state, parser, or grading outcomes under semantically equivalent controlled actions across languages).

Table 3: Ablation of BABELFLOW on 40 tasks adapted into Chinese. *Br. ref.* and *Be. mis.* report percent of tasks with broken references and behavioral mismatches; token costs are relative to independent translation. Lower is better.

Method	Br. ref.	Be. mis.	Cost (tok.)
Independent	20.0	5.0	1.00
+ Canonical	7.5	2.5	1.78
BABELFLOW	2.5	0.0	3.35

Results Table 3 highlights the effectiveness of BABELFLOW: compared with independent translation, it lowers the broken reference rate from 20.0% (eight failed tasks) to 2.5% (one failed task), with no behavioral mismatches observed among the 40 tasks. Canonical translation alone leaves higher defect rates (7.5% for broken references and 2.5% for behavioral mismatches), supporting the value of verification and human-guided revision. These gains come at a higher construction cost of $3.35\times$ the baseline token usage. The remaining reference errors highlight the challenge of preserving consistency across benchmark components during multilingual adaptation.

Table 4: Model performance by benchmark family on BABELARENA. We report Passˆ1, Passˆ3, and language consistency (LC) in percent. Higher is better. Best results are in **bold**.

	VITABENCH			DEEPPLANNING			τ^3-BENCH			CLAWEVAL		
Model	Passˆ1	Passˆ3	LC	Passˆ1	Passˆ3	LC	Passˆ1	Passˆ3	LC	Passˆ1	Passˆ3	LC
Claude-Opus-4.8	36.9	20.1	98.5	36.2	17.2	61.8	62.6	49.5	**95.0**	**55.8**	**43.0**	96.3
GPT-5.6-Terra	29.6	15.1	99.1	22.5	10.7	**90.0**	69.2	38.5	82.5	46.1	33.5	98.7
Gemini-3.1-Pro	34.3	17.7	**99.9**	18.6	5.6	74.3	**71.4**	46.9	86.2	41.5	29.3	**98.8**
Qwen-3.7-Max	36.1	18.9	98.6	41.6	23.8	57.5	59.0	49.1	92.8	32.4	24.6	98.5
Qwen-3.8-Max	**42.3**	**25.0**	94.7	**46.0**	**26.5**	75.8	67.6	**56.1**	87.8	40.8	27.1	88.7

Table 5: Model performance by language resource group on BABELARENA. We report Passˆ1, Passˆ3, and language consistency (LC) in percent. Higher is better. Best results are in **bold**.

	High resource			Medium resource			Low resource		
Model	Passˆ1	Passˆ3	LC	Passˆ1	Passˆ3	LC	Passˆ1	Passˆ3	LC
Claude-Opus-4.8	**51.3**	34.3	90.4	**48.1**	**33.9**	87.2	**44.8**	**29.6**	86.7
GPT-5.6-Terra	45.3	26.3	**98.8**	43.9	25.1	**90.9**	36.8	21.9	**89.6**
Gemini-3.1-Pro	45.9	27.4	98.6	42.4	25.8	87.5	36.4	21.0	83.8
Qwen-3.7-Max	45.8	32.4	89.0	42.9	29.5	86.9	40.1	27.6	84.9
Qwen-3.8-Max	51.2	**34.8**	89.9	48.0	33.5	86.2	44.7	29.0	84.1

4.2 EVALUATING MULTILINGUAL AGENTS

Setup We evaluate five frontier models on BABELARENA: Claude-Opus-4.8, GPT-5.6-Terra, Gemini-3.1-Pro, Qwen-3.7-Max, and Qwen-3.8-Max. All five use medium reasoning effort to control costs. We use Gemini-3.5-Flash as the simulated user agent on all four benchmarks and as the convert agent for DEEPPLANNING, mapping free-form plans to the required structured format. GPT-5.6-Sol serves as the LLM judge for VITABENCH, τ^3-BENCH, and CLAWEVAL, while DEEPPLANNING uses deterministic rules without an LLM judge. All auxiliary agents run with thinking disabled, and all models use temperature 0.

Model Performance per Benchmark Family Table 4 reports per-family results on BABELARENA. Qwen-3.8-Max generally achieves the best overall performance across the benchmark families, leading on accuracy for both reasoning-heavy tasks, VITABENCH and DEEPPLANNING, but no single model wins on all the benchmarks, suggesting the diversity of our benchmark. However, Qwen-3.8-Max leads on accuracy, yet it falls behind on language consistency. We also observe that language consistency on DEEPPLANNING is significantly worse than on the other benchmarks, and we dive into this in Section 5.

Model Performance per Language Group Table 5 shows that agent performance generally decreases as the language resource level declines. From high- to low-resource languages, both Passˆ1 and Passˆ3 decrease substantially across the five models. Language consistency also declines for every model, with LC dropping by 3.7–14.8 points. While Claude-Opus-4.8 leads both task-completion metrics across all three groups, GPT-5.6-Terra achieves the highest LC, indicating that strong task performance does not necessarily imply consistent use of the target language. These results reveal disparities across resource groups in both task completion and language adherence, motivating stronger support for agents operating in lower-resource languages.

5 ANALYSIS

To better understand the multilingual performance gaps, we analyze failure patterns, resource use, and language consistency. We examine how errors vary across languages and models, how execution costs depend on language and task type, and where agents switch away from the target language. More analysis is in Appendix D.

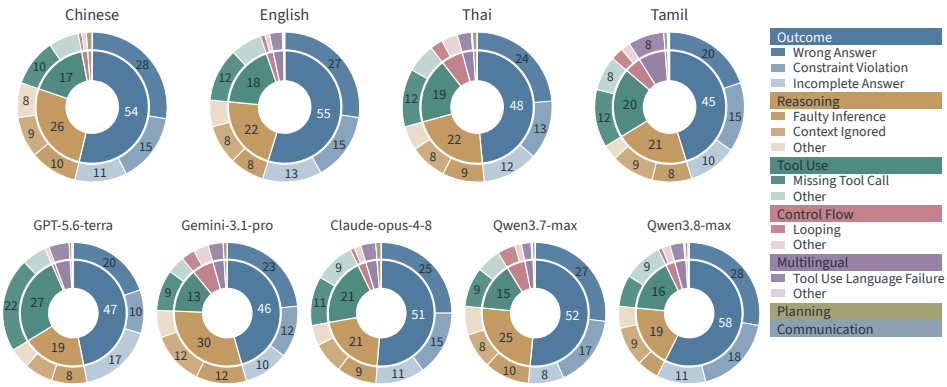


Figure 2: Primary error distributions for 1,600 failed VITABENCH trajectories by language (top, 400 per panel) and model (bottom, 320 per panel). Inner rings show broad categories; outer rings show common subtypes, pooling rarer ones within each category. Each trajectory contributes one primary error. Percentages in both rings are computed over all failures per panel and rounded to integers. See Appendix D for additional results.

Table 6: Resource use on VITABENCH by model and language resource group, relative to English on the same tasks. Turns count agent responses. Constant (Const.) input counts tokens in translated fixed text: system instructions, tool definitions, and the initial user message. Total input and output count cumulative tokens per trajectory. Darker shading indicates greater deviation from English (1.00×): red for higher usage, blue for lower. The high-resource group excludes English.

	High-resource				Medium-resource				Low-resource			
Model	Turns	Const. input	Total input	Total output	Turns	Const. input	Total input	Total output	Turns	Const. input	Total input	Total output
GPT-5.6-Terra	0.98	1.31	1.20	1.13	0.97	1.41	1.26	1.21	0.97	2.51	1.92	1.60
Gemini-3.1-Pro	1.05	1.25	1.23	1.09	1.08	1.34	1.41	1.09	1.16	1.89	1.90	1.15
Claude-Opus-4.8	1.03	1.24	1.21	1.07	1.03	1.53	1.45	1.27	1.05	2.20	2.06	1.66
Qwen-3.7-Max	0.86	1.15	0.78	0.94	0.95	1.23	0.96	1.05	1.03	2.10	1.69	1.65
Qwen-3.8-Max	0.98	1.14	1.10	1.15	0.99	1.23	1.20	1.26	1.01	2.09	1.80	1.54

Lower-resource languages show larger shares of tool-use and control-flow failures. Using Gemini-3.7-Flash and the hierarchical taxonomy in Appendix C, we label 1,600 failed VITABENCH trajectories balanced across four languages, five models, and four domains. Outcome errors dominate all languages, followed by reasoning and tool-use errors (Figure 2, top). English and Chinese have similar profiles, while Thai and Tamil show smaller outcome-error shares and larger shares of tool-use and control-flow errors, particularly unproductive loops. Tamil has the largest explicitly language-related error share, driven by tool misuse such as arguments in the wrong language.

Outcome errors dominate across models, but reasoning and tool use reveal distinct failure profiles. Outcome errors dominate all models (46–58%; Figure 2, bottom). GPT-5.6-Terra has the largest tool-use error share (27%); missing required calls are its most common error subtype (22% of failures), ahead of wrong answers. Gemini-3.1-Pro has the largest reasoning-error share (30%), mainly ignored context and faulty inference, while Claude-Opus-4.8 has similar reasoning and tool-use error shares (about one-fifth each). Compared with Qwen-3.7-Max, Qwen-3.8-Max has smaller reasoning (19% vs. 25%) and control-flow error shares (3% vs. 6%), but a larger outcome-error share (58% vs. 52%). Explicitly language-related errors account for 3–5% of failures across models, primarily involving language-induced tool misuse.

Low-resource languages incur higher token costs with a similar number of conversation turns. On the same VITABENCH tasks (Table 6), every model shows its highest constant-input, total-input, and total-output ratios in the low-resource group. Constant input rises from 1.14–1.31× English in the high-resource group to 1.89–2.51× in the low-resource group. Furthermore, low-resource

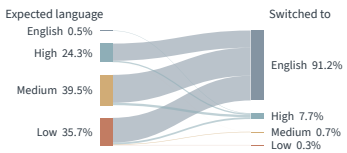
total input and output reach 1.69–2.06× and 1.15–1.66× English, respectively, while turns remain at 0.97–1.16×. Together with the failure analysis, these results motivate more efficient and effective multilingual agents for low-resource languages.

Low-resource conversation overhead varies across benchmarks and task types. On VITABENCH, turn ratios remain at 0.97–1.16× English and tool-call ratios at 0.93–1.13×, indicating relatively small changes in conversation length and tool-use volume. In contrast, on DP-SHOP, low-resource trajectories use 1.14–1.71× as many turns and 1.31–1.80× as many tool calls as English trajectories. We hypothesize that this contrast partly reflects task type: DP-SHOP requires agents to search a product catalog, where language-sensitive query matching can lead to repeated searches and query reformulation. This search requirement may amplify low-resource conversation overhead, extending multilingual resource gaps beyond token costs to longer conversation sequences and more tool use.

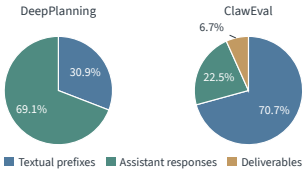
Table 7: Low-resource conversation costs on VITABENCH and DP-SHOP (DEEPPLANNING shopping), as multiples of English usage on the same tasks. Turns count agent responses and calls count tool invocations.

	VITABENCH		DP-SHOP	
Model	Turns	Calls	Turns	Calls
GPT-5.6-Terra	0.97	0.93	1.14	1.31
Gemini-3.1-Pro	1.16	1.13	1.71	1.59
Claude-Opus-4.8	1.05	1.03	1.49	1.66
Qwen-3.7-Max	1.03	1.09	1.67	1.80
Qwen-3.8-Max	1.01	1.00	1.68	1.79

Language inconsistency is dominated by switches to English, but the affected component differs across benchmarks. All five models are less language-consistent on DEEPPLANNING than on VITABENCH or CLAWEVAL (Table 4). We annotate switch destinations and affected components in 2,000 randomly sampled DEEPPLANNING and CLAWEVAL trajectories flagged as language-inconsistent (Figure 3). English accounts for 91.2% of annotated switches; medium- and low-resource languages are rare destinations. Inconsistency mainly occurs in textual prefixes before tool calls on CLAWEVAL and assistant responses on DEEPPLANNING. Manually inspecting 20 trajectories per benchmark, we find that CLAWEVAL agents commonly add English prefaces before tool calls, such as "*Let me check the weather now*". Similarly, 13 of the 20 inspected DEEPPLANNING responses preface the required `<plan>...</plan>` block with English text such as "*Let me compile a travel plan now*". We hypothesize that these prefaces stem from training data and leave further study to future work.



(a) Switching directions across language groups.



(b) Switching components by benchmark.

Figure 3: User-visible language inconsistency in 2,000 random DEEPPLANNING and CLAWEVAL trajectories.

6 CONCLUSION

We introduce BABELFLOW, a benchmark-general agentic workflow for adapting existing agent benchmarks to new languages. Applying BABELFLOW to four complementary benchmarks yields BABELARENA, covering a wide range of languages and domains. We validate the effectiveness of BABELFLOW and evaluate five frontier LLM agents on BABELARENA and find that no model dominates across benchmark families and languages. For lower-resource languages, failures shift from answer quality toward tool-use and control-flow errors, while token consumption increases substantially even on successful tasks. We also observe that language inconsistency is most visible in textual prefixes preceding tool calls. These findings highlight the need to evaluate task completion alongside reliability and efficiency. BABELARENA provides a broad, controlled testbed for advancing agents that serve users reliably and efficiently in their own languages.

AI USE STATEMENT

In preparing this manuscript, we used AI tools (Codex) only to polish the language and improve the clarity of the presentation. Their use was limited to correcting grammar, rephrasing author-written sentences, and improving readability and consistency of wording. The research questions, core ideas, methodology, experimental design, analyses, and scientific conclusions of this work were developed by the human authors, and no claims, results, or references were generated by these tools. Every AI-assisted edit was reviewed for correctness by at least two human authors. We take full responsibility for the final content of this work.

REFERENCES

Lucas Bandarkar, Davis Liang, Benjamin Muller, Mikel Artetxe, Satya Narayan Shukla, Donald Husa, Naman Goyal, Abhinandan Krishnan, Luke Zettlemoyer, and Madian Khabsa. The belebele benchmark: a parallel reading comprehension dataset in 122 language variants. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 749–775, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.44. URL https://aclanthology.org/2024.acl-long.44/.

Victor Barrès, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *CoRR*, abs/2506.07982, 2025. doi: 10.48550/ARXIV.2506.07982. URL https://doi.org/10.48550/arXiv.2506.07982.

Andrea Caciolai, Pere-Lluís Huguet Cabot, Chierh Cheng, Albert Ventayol-Boada, Gabriel Mejia Gonzalez, Christophe Ropers, Lucas Bandarkar, Sebastian Ruder, Darlene Sakakihara, Elliot Yun, Pierre Andrews, Grégoire Mialon, Romain Froger, and Marta R. Costa-jussà. Omnilingualgaia2: Evaluating the multilingual gap in frontier AI agents, 2026. URL https://doi.org/10.48550/arXiv.2608.08775.

Maria Couto, Breno Miranda, and Kiev Gama. A tool-assisted training approach for empowering localization and internationalization testing proficiency. In *IEEE Conference on Software Testing, Verification and Validation, ICST 2025, Napoli, Italy, March 31 - April 4, 2025*, pp. 711–720. IEEE, 2025. doi: 10.1109/ICST62969.2025.10988966. URL https://doi.org/10.1109/ICST62969.2025.10988966.

Lais Felipe, Breno Miranda, and Maria Couto. Tstring: a tool to locate the target string's screen based on automatic exploration. In Everton L. G. Alves and Márcio Ribeiro (eds.), *Proceedings of the 9th Brazilian Symposium on Systematic and Automated Software Testing, SAST 2024, Curitiba, Brazil, September 30 - October 4, 2024*, pp. 66–73. Sociedade Brasileira de Computação, 2024. doi: 10.5753/SAST.2024.3886. URL https://doi.org/10.5753/sast.2024.3886.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. Vitabench: Benchmarking LLM agents with versatile interactive tasks in real-world applications. *CoRR*, abs/2509.26490, 2025. doi: 10.48550/ARXIV.2509.26490. URL https://doi.org/10.48550/arXiv.2509.26490.

Omer Hofman, Jonathan Brokman, Oren Rachmil, Shamik Bose, Vikas Pahuja, Toshiya Shimizu, Trisha Starostina, Kelly Marchisio, Seraphina Goldfarb-Tarrant, and Roman Vainshtein. MAPS: A multilingual benchmark for agent performance and security. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Findings of the Association for Computational Linguistics: EACL 2026, Rabat, Morocco, March 24-29, 2026*, Findings of ACL, pp. 821–845. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.FINDINGS-EACL.42. URL https://doi.org/10.18653/v1/2026.findings-eacl.42.

Yunsu Kim, Kaden Uhlig, and Joern Wuebker. Gaia-v2-lilt: Multilingual adaptation of agent benchmark beyond translation. *CoRR*, abs/2604.24929, 2026. doi: 10.48550/ARXIV.2604.24929. URL https://doi.org/10.48550/arXiv.2604.24929.

Peng Kuang, Haibo Jin, Xiaoyu Han, Yanli Wang, Xiaopeng Yuan, Ye Yu, Kaidi Xu, and Haohan Wang. Kv-prm: Efficient process reward modeling via kv-cache transfer for multi-agent test-time scaling, 2026. URL https://arxiv.org/abs/2607.09153.

Mayank Kulkarni, Vittorio Mazzia, Judith Gaspers, Chris Hench, and Jack FitzGerald. MASSIVE-agents: A benchmark for multilingual function-calling in 52 languages. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 20193–20215, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1099. URL https://aclanthology.org/2025.findings-emnlp.1099/.

Hongliang Li, Yijin Liu, Zhiwei Zhang, Zihe Liu, Xinyue Lou, Jinan Xu, Fandong Meng, and Kaiyu Huang. Polyworkbench: Benchmarking multilingual long-horizon LLM agents. *CoRR*, abs/2607.06008, 2026. doi: 10.48550/ARXIV.2607.06008. URL https://doi.org/10.48550/arXiv.2607.06008.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Zheng Luo, Thirulogasankar Pranav Kutralingam, Ogochukwu N. Okoani, Wanpeng Xu, Hua Wei, and Xiyang Hu. Lost in execution: On the multilingual robustness of tool calling in large language models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 44059–44077. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.2039. URL https://doi.org/10.18653/v1/2026.acl-long.2039.

My Chiffon Nguyen, Aulia Adila, Saksorn Ruangtanusak, Kittiphat Leesombatwathana, Vissuta Gunawan Lim, Patomporn Payoungkhamdee, and Samuel Cahyawijaya. Seataubench: Adapting tool-agent-user evaluation into low-resource southeast asian languages. *CoRR*, abs/2606.28715, 2026. doi: 10.48550/ARXIV.2606.28715. URL https://doi.org/10.48550/arXiv.2606.28715.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Marco Pirrone and Arianna D'Ulizia. The localization of software and video games: Current state and future perspectives. *Inf.*, 15(10):648, 2024. doi: 10.3390/INFO15100648. URL https://doi.org/10.3390/info15100648.

Quan Shi, Alexandra Zytek, Pedram Razavi, Karthik Narasimhan, and Victor Barrès. τ-knowledge: Evaluating conversational agents over unstructured knowledge. *CoRR*, abs/2603.04370, 2026. doi: 10.48550/ARXIV.2603.04370. URL https://doi.org/10.48550/arXiv.2603.04370.

Xiaoyin Wang, Lu Zhang, Tao Xie, Hong Mei, and Jiasu Sun. Locating need-to-externalize constant strings for software internationalization with generalized string-taint analysis. *IEEE Trans. Software Eng.*, 39(4):516–536, 2013. doi: 10.1109/TSE.2012.40. URL https://doi.org/10.1109/TSE.2012.40.

Xin Xia, David Lo, Feng Zhu, Xinyu Wang, and Bo Zhou. Software internationalization and localization: An industrial experience. In *2013 18th International Conference on Engineering of*

Complex Computer Systems, Singapore, July 17-19, 2013, pp. 222–231. IEEE Computer Society, 2013. doi: 10.1109/ICECCS.2013.40. URL https://doi.org/10.1109/ICECCS.2013.40.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *CoRR*, abs/2406.12045, 2024. doi: 10.48550/ARXIV.2406.12045. URL https://doi.org/10.48550/arXiv.2406.12045.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-eval: Toward trustworthy evaluation of autonomous agents. *CoRR*, abs/2604.06132, 2026. doi: 10.48550/ARXIV.2604.06132. URL https://doi.org/10.48550/arXiv.2604.06132.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. Deepplanning: Benchmarking long-horizon agentic planning with verifiable constraints. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 7377–7407. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.335. URL https://doi.org/10.18653/v1/2026.acl-long.335.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

A Language Consistency Check

We use Gemini-3.7-Flash to assess all user-visible text across the complete trajectory τ against the target language ℓ_t. The check covers assistant responses, user-visible textual prefixes preceding tool calls, and natural-language content in deliverables. Deliverable text is included even when delivered through tools; internal reasoning, user turns, tool-call syntax, and other tool-result content are excluded. Language consistency is evaluated independently of task completion, and a single inconsistent passage yields $\mathrm{LC}(\tau, \ell_t) = 0$ even if later responses return to ℓ_t. A trajectory can therefore be language-consistent but fail the task, or language-inconsistent but succeed; its task reward is unaffected by the LC score. The prompt used for checking language consistency is presented in Figure 4.

B Quality Assurance

We conduct a text-level translation audit to assess whether the multilingual variants preserve the source meaning and remain clear and natural in the target language. For each of the 23 languages, we randomly sample 200 translatable texts, yielding 4,600 annotation records in total. For the seven high-resource languages, each record is annotated by two human language experts, while GPT-5.6-Sol assesses records in the eight medium-resource and eight low-resource languages. Each annotation assigns an integer score from 1 to 5 according to the rubric in Table 8.

The mean scores are 4.28, 4.02, and 3.91 for high-, medium-, and low-resource languages, respectively. The inter-annotator agreement (IAA) between the human experts, measured by Cohen's κ, is 0.71. These results demonstrate the high quality of the multilingual translations in BabelArena.

C Error taxonomy and annotation

To characterize how agents fail across languages and models, we annotate failed VitaBench trajectories using a hierarchical error taxonomy. The taxonomy captures both general agent errors and failures tied to multilingual interaction. This section presents the sampling procedure, annotation rules, complete taxonomy, and fixed prompt used to produce the error distributions in Figure 2.

```
You are auditing whether an AI assistant replied in the language it
was supposed to use.

Target language: {language}

Below is everything the assistant said during one task, with its tool
calls and the user's turns removed. Judge ONLY the language of the
assistant's own prose.

Treat these as CORRECT, never as violations:
- Identifiers echoed from the environment: reservation/order/user
IDs, flight, train and product codes, SKUs, phone numbers, addresses,
dates, times, prices and currency symbols.
- Tool names, function names, argument names, JSON keys, code and
file paths.
- Structural scaffolding the output format requires. deepplanning's
travel plans, for example, are emitted inside <plan> tags with
English field labels ("Day 1:", "Current City:", "Accommodation:",
"Transportation:") because the scorer parses them; only the values
around those labels should be in the target language. Markdown
headings, table headers and bullet markers are likewise formatting,
not prose.
- Proper nouns that the benchmark deliberately keeps in their
original form: brand names, product model names,
hotel/restaurant/attraction names, airline and city names. These
benchmarks preserve such names on purpose so that tool calls still
match, so their presence is expected and correct.
- Standard loanwords and technical terms that a fluent speaker of the
target language would normally leave untranslated.

Count as a VIOLATION only prose the assistant composed itself --
sentences, explanations, questions, apologies, summaries -- written
in a language other than the target. An empty or near-empty sample
is not a violation.

Answer with a single JSON object and nothing else:
{"consistent": true|false, "share_in_target": <0.0-1.0>,
"observed_language": "<dominant language of the assistant's
prose>", "evidence": "<at most 200 characters quoting the strongest
off-language prose, or empty>"}

share_in_target is your estimate of how much of the assistant's own
prose was in the target language. Set consistent=false when
meaningful prose is in the wrong language, not merely because
identifiers or proper nouns appear.

--- ASSISTANT OUTPUT ---
{sample}
--- END ---
```

Figure 4: The prompt used for trajectory-level language consistency evaluation with Gemini-3.7-Flash. {language} is the target language name and {sample} is the concatenation of the assistant's own messages for one trajectory, with tool calls, tool results and user turns removed. The judge returns a JSON object; the binary consistent field is the label used for scoring, and share_in_target is recorded for auditing but does not enter the metric.

Table 8: Five-point translation quality rubric for the text-level audit. The same criteria apply to human experts and the LLM judge.

Score	Level	Criteria
5	Excellent	Fully faithful, complete, fluent, and natural. All relevant terminology and protected content are handled correctly, with no identifiable translation issues.
4	Good	Faithful and complete, with only minor grammatical or stylistic imperfections. These do not affect meaning, clarity of requirements, or protected content.
3	Fair	The main intent remains clear, but localized, noncritical inaccuracies or noticeably awkward wording require revision. Task-critical requirements and protected content remain intact.
2	Poor	Substantial mistranslations, omissions, or ambiguity distort the intended meaning, or an error changes a task-critical requirement or protected value. Some source meaning is still recoverable.
1	Unusable	The translation is largely incorrect, incomprehensible, or missing, or the content requiring translation is left in the wrong language. The intended meaning cannot be reliably recovered from the target-language text.

Sampling and labeling We analyze 1,600 failed VITABENCH trajectories, sampling 20 trajectories for each combination of five models, four languages (English, Chinese, Thai, and Tamil), and four domains. Each trajectory has task reward 0, although it may still receive partial rubric credit. Gemini-3.7-Flash annotates the complete interaction trajectory with the model's hidden thinking removed; the input retains user messages, assistant responses, tool calls, tool results, and rubric verdicts.

Annotation prompt The system prompt includes the complete error taxonomy in Table 9, covering eight broad categories and 39 leaf subtypes. Annotation is multi-label: every supported error is assigned a taxonomy id together with a short quote or turn reference, and exactly one assigned label is selected as the *primary* error most directly responsible for failure. Leaf ids are used whenever possible, with a top-level category used only when no leaf fits. The main-text error distributions use only the primary label, so each trajectory contributes once.

The prompt also separates agent errors from benchmark and evaluation issues. In non-Chinese tasks, Chinese text generated by the agent can be labeled as a multilingual failure, while untranslated Chinese surfaced by tools or the environment is recorded separately as a benchmark artifact. Suspected evaluator mistakes are recorded separately, and a novel pattern is noted when the dominant failure does not fit the taxonomy. The complete fixed prompt is shown in Figure 5; at runtime, {TAXONOMY} is replaced by the taxonomy in Table 9.

Table 9: Complete error taxonomy used in the failure analysis. Bold rows define broad categories and their scope: Single-turn covers one response and its input, Multi-turn covers a trajectory, and Both covers either. Subtypes inherit the scope of their parent category.

Category / subtype	Definition
Outcome	**The final answer is wrong, incomplete, or violates the task contract.**
Single-turn	
Wrong Answer	Final answer does not match the reference / success criterion.
Incomplete Answer	Answer omits required parts of the expected response.
Format Violation	Output does not follow the requested format or structure.
Constraint Violation	Answer breaks an explicit task constraint (length, scope, rules).
Unfaithful Answer	Answer is not supported by the provided input or context.
Reasoning	**The reasoning that produced the answer is flawed.**
Single-turn	
Factual Hallucination	States a fact that is false or unsupported.

Continued on next page

Table 9 continued

Category / subtype	Definition
Logical Inconsistency	Reasoning contradicts itself.
Faulty Inference	Draws a conclusion that does not follow from the premises.
Calculation Error	Makes an arithmetic or computational mistake.
Unfounded Assumption	Relies on an assumption with no support.
Context Ignored	Fails to use relevant information present in the input.
Planning *Multi-turn*	**The plan or decomposition of the task is poor.**
Missing Plan	Acts without forming a plan when one was needed.
Poor Decomposition	Breaks the task into ineffective or wrong subtasks.
Goal Drift	Drifts away from the original goal over the trajectory.
Scope Creep	Expands the task beyond what was asked.
Inefficient Path	Takes a needlessly long or wasteful route to the goal.
Tool Use *Both*	**Errors in selecting, calling, or interpreting tools.**
Wrong Tool Selected	Chooses a tool inappropriate for the subtask.
Hallucinated Tool	Calls a tool that does not exist.
Hallucinated Arguments	Invents argument values not grounded in context.
Malformed Arguments	Tool-call arguments violate the declared schema.
Missing Tool Call	Fails to call a tool that the task required.
Tool Output Ignored	Ignores a tool result it should have used.
Tool Result Misinterpreted	Misreads or misuses a tool result.
Control Flow *Multi-turn*	**Errors in how the trajectory progresses or terminates.**
Looping	Repeats the same step(s) without progress.
Oscillation	Alternates between states without converging.
Premature Stop	Stops before the task is complete.
Non Termination	Continues past the point of completion.
Stuck No Progress	Makes no measurable progress over several steps.
Memory *Multi-turn*	**Errors in tracking state or earlier context.**
Context Forgotten	Forgets information established earlier in the trajectory.
Self Contradiction Over Time	Contradicts an earlier statement of its own.
State Tracking Error	Loses track of intermediate state.
Communication *Single-turn*	**Errors in how the output is communicated.**
Unclear Output	Output is confusing or ambiguous.
Fabricated Citation	Cites a source that is fabricated or wrong.
Overclaiming	Asserts more certainty or capability than warranted.
Multilingual *Single-turn*	**Failures tied to operating in a non-English or multilingual context.**
Wrong Language Output	Responds in a different language than the user/task language (e.g. replies in English or Chinese to an Arabic task).
Language Mixing	Unwanted code-switching — mixes languages within a single response when it should stay in one.
Script / Encoding Corruption	Garbled characters, mojibake, or wrong script — especially in low-resource scripts (Burmese, Khmer, Tamil, Telugu).
Locale Convention Error	Wrong date/number/currency/unit format, name order, honorifics, or register/politeness for the locale.
Tool Use Language Failure	Language-induced tool misuse — passes a localized/non-canonical value into a tool argument that expects a canonical key/ID, or mishandles an English tool result while reasoning in the task language.

```
You are an expert annotator for a multilingual tool-using agent
benchmark.

You will read ONE failed agent trajectory.  The task was performed in
a specific language (which may be Chinese, English, Thai, or Tamil).
The benchmark's SOURCE language is Chinese; every non-Chinese variant
is a translation of the same underlying task, so any Chinese text
appearing in a non-Chinese variant is notable and you must decide
whether the AGENT produced it or the ENVIRONMENT surfaced it.

The trajectory scored reward = 0, so at least one error label is
REQUIRED.

Label it against this taxonomy.  Use LEAF ids (e.g.
tool_use.tool_output_ignored); use a top-level id ONLY when the error
is real but no leaf fits.

{TAXONOMY}

Rules:
1.  MULTI-LABEL. Assign every label the evidence supports, typically
1--4.  Do not pad.
2.  Every label needs evidence:  a short verbatim quote or a turn
reference from the trajectory.  No evidence -> do not assign the
label.
3.  Pick exactly one primary label:  the error that most directly
caused the failure.  It must also appear in labels.
4.  Judge the JUDGE too.  The rubric verdicts are given; if you
believe the judge was wrong (the agent actually satisfied the
requirement), say so in judge_disputed and explain.
5.  multilingual.* labels:  assign ONLY for genuine language-induced
problems.
  - The agent replying in the wrong language -> wrong_language_output.
  - The agent passing a Chinese (source-language) value into a tool
argument in a non-Chinese task -> tool_use_language_failure.  Also set
agent_generated_source_lang to true if that Chinese string never
appeared in any earlier tool result or user turn (i.e.  the agent
invented it), false if it was copied from something visible.
  - Untranslated Chinese appearing in TOOL RESULTS is a benchmark
localisation artifact, not an agent error:  record it in
benchmark_artifact, do NOT label the agent for it.
6.  novel_pattern:  if the dominant failure mode is real but the
taxonomy has no good slot, describe it in one sentence.  Otherwise
null.
7.  Output STRICT JSON only.  No markdown fence, no commentary.

Schema:
{"labels":[{"id":"<taxonomy id>","evidence":"<quote/turn
ref>","confidence":0.0-1.0}],
  "primary":"<taxonomy id>",
  "agent_generated_source_lang":  true|false|null,
  "benchmark_artifact":  "<one sentence or null>",
  "judge_disputed":  "<one sentence or null>",
  "novel_pattern":  "<one sentence or null>",
  "summary":"<one sentence, in English, on why it failed>"}
```

Figure 5: The system prompt used for failure annotation with Gemini-3.7-Flash. The {TAXONOMY} placeholder is replaced at runtime by the complete taxonomy in Table 9. The annotator receives one failed trajectory with the model's hidden thinking removed and returns multi-label error annotations together with one primary failure label.

D More Analysis on Failure Types and Resource Costs

D.1 Failure Type Breakdowns

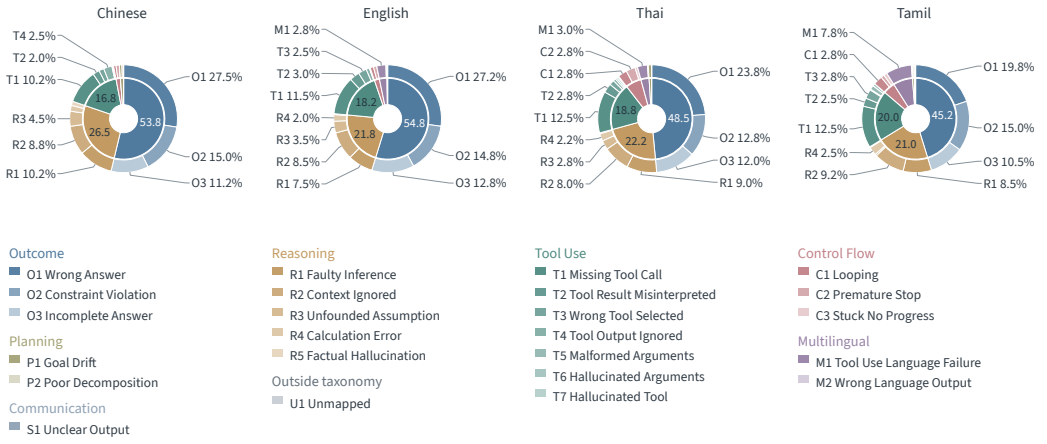


Figure 6: Detailed error subtypes by language. Inner rings show broad categories and their percentages among sampled failures; outer rings show subtypes identified by the legend. Each language has 400 trajectories, and each trajectory contributes one primary label.

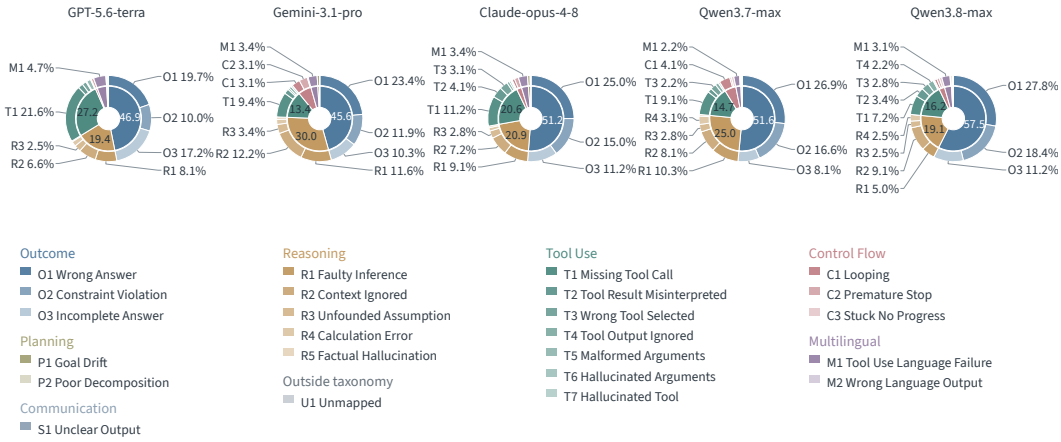


Figure 7: Detailed error subtypes by model. Inner rings show broad categories and their percentages among sampled failures; outer rings show subtypes identified by the legend. Each model has 320 trajectories, and each trajectory contributes one primary label.

Figure 6 and Figure 7 provide the detailed error distributions by language and model, respectively, for the same 1,600 failed VitaBench trajectories summarized in Figure 2, including subtypes pooled in the main figure. Consistent with the main-text analysis, outcome errors account for the largest share across languages and models, while Thai and Tamil have larger shares of tool use and control flow errors than English and Chinese. Tamil also has the largest share of multilingual errors, primarily tool use language failure. The model breakdown highlights distinct failure profiles: missing tool call is the most frequent subtype for GPT-5.6-Terra, while Gemini-3.1-Pro has the largest reasoning share, mainly involving context ignored and faulty inference. Claude-Opus-4.8 has comparable shares of reasoning and tool use errors. Compared with Qwen-3.7-Max, Qwen-3.8-Max has smaller shares of reasoning and control flow errors but a larger outcome share.

D.2 RESOURCE ACCOUNTING AND PAIRED COMPARISONS

The resource analysis in Section 5 examines whether solving the same tasks in different languages also changes execution costs. On VITABENCH, low-resource languages consume substantially more tokens than English despite similar numbers of turns (Table 6), whereas DP-SHOP also shows more turns and tool calls (Table 7). These findings motivate tracking token consumption alongside interaction counts and comparing matched tasks across languages. This subsection details the resource definitions, trajectory selection, and aggregation underlying the main-text comparisons, followed by an additional analysis that holds task success and interaction counts fixed.

Resource measures and trajectory selection Turns count generated agent responses, including responses that invoke tools, and calls count individual tool invocations. Input and output tokens sum provider-reported usage over agent-model calls, excluding user-simulator and evaluator calls. The *constant input* reported in the main text is operationalized as the input token count of the first agent-model call. It therefore captures the context present before the first agent response, including system instructions, tool descriptions and parameter schemas, the opening user message, and any supplied conversation history. For VITABENCH, we retain trajectories with complete token records and tasks with all three repetitions available in both English and the target language. Within each model, trajectories are matched by domain, task identity, and repetition, and repeated trajectories are averaged within each task. As in the main-text resource analysis, the comparison includes both successful and failed trajectories.

Token overhead persists on successful trajectories with equal turns and tool calls. We further compare paired τ^3-BENCH trajectories that succeed in both English and other languages under matching configurations and use exactly the same numbers of turns and tool calls. Table 10 shows that input and output token use remain higher than English for every model in the represented low-resource languages.

Table 10: Token use relative to paired English trajectories for successful τ^3-Bench trajectories with exactly equal turns and tool calls. Tasks counts distinct domain–task identities; trajectory pairs counts matched English/target repetitions.

Model	Tasks	Trajectory pairs	Input (×)	Output (×)
GPT-5.6-Terra	80	238	2.02	1.50
Gemini-3.1-Pro	103	442	1.61	1.38
Claude-Opus-4.8	18	45	3.20	2.59
Qwen-3.7-Max	108	413	1.98	2.14
Qwen-3.8-Max	84	235	2.09	1.83

D.3 REPEATED RETRIEVAL INCREASES INTERACTION COSTS IN LOW-RESOURCE LANGUAGES

We examine whether repeated retrieval contributes to the higher interaction costs observed in the shopping domain of DEEPPLANNING (DP-SHOP; Section 5). Whereas VITABENCH combines dialogue with a simulated user and service actions such as ordering and booking, DP-SHOP provides the full multi-item objective at the outset. The agent must retrieve products, inspect their attributes and delivery constraints, and assemble a valid cart. Search ranks localized catalog records by lexical overlap with the query, making retrieval sensitive to query wording and script. Unsuccessful queries can therefore lead to repeated searches and reformulation.

Repeated searches account for much of the additional tool use. Table 11 compares retrieval activity on the same tasks in English and low-resource languages. All five models make more product searches in low-resource languages. Across models, empty results account for 53.7%–68.7% of these searches, compared with 6.1%–17.1% in English. Additional product searches account for 49.8%–79.9% of the increase in tool calls relative to English.

The benchmark also prompts agents to verify and, if needed, correct their carts. This stage can extend a trajectory, but 69.3%–94.0% of the additional low-resource calls occur in the initial stage,

before the verification prompt. Most of the interaction overhead thus arises during the initial attempt to assemble the cart.

	Searches / run		Empty results (%)		Search share
Model	English	Low	English	Low	of extra calls (%)
GPT-5.6-Terra	5.65	10.41	9.0	53.7	52.2
Gemini-3.1-Pro	6.30	30.32	17.1	68.7	76.6
Claude-Opus-4.8	4.62	19.08	6.2	56.1	79.9
Qwen-3.7-Max	4.47	14.22	6.1	66.8	49.8
Qwen-3.8-Max	5.43	21.86	6.7	67.2	68.8

Table 11: Retrieval activity across all paired DP-SHOP trajectories, with equal weight per low-resource language (Low). Empty-result rates are the mean number of empty product-search responses divided by the mean number of product searches. Search share is the increase in mean product searches divided by the increase in mean tool calls, both relative to paired English trajectories.

Paired successes illustrate how retrieval overhead arises. We manually examine paired English and low-resource trajectories that succeed on the same task. For Qwen-3.8-Max on task 37, the Telugu trajectory initially uses English search phrases that return empty results. Repeated searches continue into the cart-verification stage, and Telugu queries at calls 159–161 eventually retrieve the required candidates. Both trajectories obtain the same four products, but Telugu uses 64 turns and 201 calls, compared with nine turns and 25 calls in English. Product searches increase from five in English to 158 in Telugu, of which 126 return empty results. For Claude-Opus-4.8 on task 4, the English query "Himalayan" immediately retrieves the required parka. The Belarusian trajectory repeatedly reformulates unsuccessful queries before a localized phrase retrieves it at call 102. Both trajectories obtain the same required products and coupon choices, but Belarusian uses 43 turns, 121 calls, and 90 searches, including 59 empty results, compared with 13 turns, 25 calls, and four searches in English. All 121 Belarusian calls occur before the cart-verification prompt, illustrating that substantial retrieval overhead can arise within the initial stage alone. Together, the aggregate measurements and paired examples support repeated retrieval as a source of language-related interaction overhead, including on tasks that ultimately succeed.