BEKCHIAI: MEASURING, OBSERVING, AND CONTROLLING LLM AGENTS IN ONE CLICK

Mesut Toruk
Istanbul, Turkiye
m.mesut.toruk@gmail.com

ABSTRACT

Large-language-model *agents* reason, call tools, and act autonomously over many steps, but their *agentic* skills—correctly sequencing tools, planning under dependencies, judging untrusted inputs, and grounding generated arguments—are hard to measure with accuracy-only leaderboards. We present BekchiAI, which addresses both sides: a benchmark for measuring agentic skill and a platform for observing and controlling live agents. The **BekchiAI-Benchmark**, a suite of 13 tool-using ReAct agents across 7 task categories (arithmetic, structured/SQL, security detection, URL grounding, planning, orchestration, and tool-policy), totalling 2,057 deterministic, committed test tasks. Every task is verifier-checkable—gold answers are computed by running canonical SQL against a real database, computing the exact schedule of a directed acyclic graph (DAG), or evaluating closed-form lambdas including adversarial security samples paired with deliberately *imperfect* signature scanners so a score reflects the model's own judgment, not the copying of an oracle. We define a small set of behavioral metrics beyond accuracy—tool-call adherence, URL hallucination and source-match, and per-model token cost—and report a four-model comparison (`Qwen3.7-Max`, `gemma-4-31B-it`, `gemma4:26b`, `gpt-oss-120b`) whose story is in the per-family spread, not the aggregate. The benchmark runs are executed using the provided evaluation scripts. **BekchiAI-Platform** is a complementary web-based observability and control layer for deployed agents, providing full token and latency telemetry as well as remote run termination. The benchmark, evaluation tools, and platform are publicly released.

Keywords LLM agents · benchmark · tool use · planning · agent security · evaluation · telemetry

1 Introduction

Large language models (LLMs) are increasingly deployed not as single-shot text generators but as *agents*: systems that interleave model calls with tool use, memory, and multi-step planning to pursue goals with limited human supervision [Yao et al., 2023, Schick et al., 2023, Wang et al., 2024, Xi et al., 2023]. This paradigm has moved quickly from research demonstrations into consequential applications. In software engineering, agents equipped with shell, editor, and search tools resolve real GitHub issues end-to-end [Jimenez et al., 2024, Yang et al., 2024]; on the web, they navigate live sites to complete multi-step tasks such as booking, form-filling, and shopping [Deng et al., 2023, Zhou et al., 2024]; and in the sciences, LLM-driven systems autonomously design, plan, and execute laboratory experiments [Boiko et al., 2023]. Industry adoption is following: Gartner projects the share of enterprise applications embedding task-specific AI agents to rise from under 5% in 2025 to roughly 40% by 2026, with a growing fraction of routine decisions delegated to agents [Gartner, 2025]. As agents take real actions—writing code, moving money, calling external services—the practical question shifts from "how fluent is the model?" to "how good are its *agentic skills*?"

These skills—does the agent select and sequence the right tools, respect dependencies when planning, refuse a poisoned input, and ground the arguments (URLs, queries) it fabricates?—are precisely what a single accuracy number over a mixed test set fails to isolate. Worse, many popular agent tasks are easily gamed by pattern-matching or leak their gold through an oracle tool, so a high score need not reflect real competence or safe judgment.

We address this with the **BekchiAI-Benchmark**: 13 tool-using ReAct agents, each a self-contained task family with a verifier-checkable gold answer, spanning 7 categories and 2,057 committed test tasks (Section 3). Three design choices make the suite hard to game: (i) gold is computed by a *verifier*—canonical SQL over a real database, the exact schedule of a directed acyclic graph (DAG), or a closed-form lambda—so it cannot drift; (ii) the hard cases adversarial security inputs, homoglyph URL traps, deep dependency graphs) are *hand-authored one by one*, not templated; and (iii) the security detectors the agent may call are *deliberately imperfect* signature scanners, so accuracy measures the model's judgment rather than its ability to echo an oracle. The agents run on a small, purpose-built **tool library** (Section 4). We report each model's accuracy and per-model token cost; the harness additionally computes behavioral **metrics**—tool-call adherence, and URL hallucination and source-match (Section 5)—that we release with the benchmark.

The benchmark runs on a self-contained evaluation harness that logs per-call token and latency next to each outcome, so capability and cost come from a single run. Our four-model comparison (Section 6) shows that the ranking: a model that leads overall can collapse on structured or multi-step tool tasks while another inverts the order on grounding. Complementing the benchmark, the **BekchiAI-Platform** (Section 7) is a web-based observability and control layer for deployed agents—streaming the same per-call telemetry live and letting an operator *stop*, *pause*, *resume*, or *revoke* a running agent with one click.

We run every rollout under the **BekchiAI-Platform**, an observability and control layer that instruments the benchmark drop-in—streaming per-call token/latency telemetry and enabling a remote kill-switch—so capability and cost are measured from one stream (Section 7). Our four-model comparison (Section 6) shows that the ranking lives in the *per-family spread*: a model that leads overall can collapse on structured or multi-step tool tasks while another inverts the order on grounding.

Contributions.

- **The BekchiAI-Benchmark**: 13 verifier-checked, tool-using ReAct task families (2,057 committed tasks) with hand-authored adversarial samples and imperfect-by-design security tools (Section 3).
- **A released tool library** of arithmetic, SQL, security-scanner, web/URL, and business tools the agents call (Section 4), and a set of behavioral **metrics** beyond accuracy (Section 5).
- **A four-model agentic-skill comparison** read from a single instrumented stream, capturing both capability (per family) and cost (Section 6).
- **The BekchiAI-Platform**: a drop-in observability + control layer (agent-native event model, grounding and plan-vs-actual analytics, policy enforcement, remote control) used to run the benchmark (Section 7).

2 Related Work

Agent and tool-use benchmarks. A growing body of benchmarks evaluates LLM agents. General suites such as AgentBench [Liu et al., 2024] and GAIA [Mialon et al., 2023] probe multi-step reasoning and tool use across heterogeneous environments, while domain benchmarks target software engineering (SWE-bench [Jimenez et al., 2024]), web navigation (Mind2Web [Deng et al., 2023], WebArena [Zhou et al., 2024]), and function/API calling (ToolLLM [Qin et al., 2024], τ-bench [Yao et al., 2024]). Most report a single end-to-end success rate per task, which conflates distinct skills and is sensitive to prompt formatting and to gold that can leak through an oracle tool. The BekchiAI-Benchmark is complementary: it *isolates* seven skill categories into verifier-checked families, computes gold from a real database, scheduling DAG, or closed-form lambda so it cannot drift, and reports behavioral metrics—tool-call adherence, plan-vs-actual consistency, and grounding—alongside accuracy.

Agent security. InjecAgent [Zhan et al., 2024] and AgentDojo [Debenedetti et al., 2024] measure whether an agent is *hijacked* by malicious instructions injected into tool outputs or retrieved content. Our three security families ask a related but distinct question: given an untrusted artifact—a form value, a retrieved passage, a generated output, or a dependency—and a *deliberately imperfect* detector, does the model correctly decide to block or proceed? Because the scanner agrees with the truth only 40–55% of the time, the score reflects the model's own judgment rather than oracle-echoing, and it penalises over-blocking benign inputs as well as missing real threats.

Agent observability. Orthogonally, generic tracing and metrics stacks capture service calls but not agentic structure—sessions, tool trees, per-call token usage, and argument provenance—and offer no remote control of an autonomous process. The BekchiAI-Platform targets that gap and lets us read capability and cost from one instrumented stream.

Table 1: The 13 agents of the BekchiAI-Benchmark, their category, committed test-set size n, the tools they call, and the skill each isolates.

Agent	Category	Tools	n	Skill probed
ar_math	arithmetic	calc	154	multi-step computation
ar_multistep	arithmetic	lookup, calc	152	constant lookup + compute
sql_analytics	structured	list_tables, schema, run_sql	157	SQL analytics (joins, group-by)
sec_sqli	security	scan_sql	154	SQL-injection judgment
sec_context_poison	security	scan_context	154	prompt-injection judgment
sec_guard	security	security_scan	151	LLM security-risk gating
ug_fetch	url-grounding	websearch, fetch	155	argument generation (no URL given)
ug_imposter	url-grounding	websearch, fetch	150	homoglyph/typosquat resistance
dependency_planning	planning	customer-record lookups	180	dependency resolution
multi_tool_planning	planning	customer, fx_rate, calc, discount	152	multi-tool orchestration
orchestrator	orchestration	—	196	as-soon-as-possible scheduling
tool_error_recovery	tool-policy	cache, mirror, authdb	150	failover / retry policy
tool_arguments	tool-policy	5 record-creation tools	152	correct argument construction
Total			**2,057**	

3 The BekchiAI-Benchmark

Structure. The suite is 13 agents grouped into 7 categories (Table 1). Each agent is a strict-JSON ReAct loop: the model emits one JSON action per turn—a tool call with arguments, or a final answer—against a commodity LLM API, and the harness executes tools and feeds results back until the model answers or a step budget is reached. Every task family ships a *deterministic* test set committed as JSONL; the 13 files total 2,057 held-out tasks, so a run is exactly reproducible and inspectable.

Verifier-checked gold. For `sql_analytics`, each question's canonical SQL is run against the same in-memory database the agent queries, so the gold reflects the schema conventions (amounts in integer cents; cancelled orders excluded) exactly. For `orchestrator`, the gold is the unique as-soon-as-possible level-partition of a dependency DAG, computed from the scenario graph. Arithmetic families evaluate closed-form lambdas over the sampled constants. Because the verifier and the agent see the same world, gold cannot silently drift, and a numeric tolerance (a cent for money, tight relative tolerance for arithmetic) absorbs only formatting noise.

Adversarial samples. Planning and orchestration scenarios use asymmetric-join DAGs, long idle-span dependencies, and mid-graph dead-end distractors that must be pruned by transitive reachability. The URL-grounding families pair each question with a *different real host* and, for `ug_imposter`, a pasted homoglyph link (e.g. `examp1e.com`) the agent must ignore in favour of the authoritative host.

Imperfect-by-design security tools. The three security families (`sec_sqli`, `sec_context_poison`, and `sec_guard`, whose risk classes follow the OWASP Top-10 for LLM Applications) present a *realistic operation* (process this form value, use this retrieved context, ship this generated output) carrying an untrusted artifact, and expose a scanner tool the agent may call. Crucially the scanner is a *signature heuristic that disagrees with the truth*: it misses obfuscated attacks (false negatives) and fires on benign look-alikes (false positives), agreeing with the curated gold labels only ~ 40–55% of the time. A model that blindly trusts or inverts the tool therefore scores near chance; the correct behavior is to read the artifact and decide. This converts a would-be oracle into a genuine judgment test.

4 Benchmark Tools

The agents run on a small, shared tool library which exist in github repo of benchmark; every agent imports its tools from this library rather than defining its own, so tool behavior is consistent and independently testable (Table 2). Two properties matter for the benchmark. First, the data-backed tools operate over *real* deterministic state: `run_sql` executes against an in-memory SQLite database of customers/products/orders, and `fetch` serves an offline index of real canonical web pages (404-ing everything else), so the verifier and the agent share one ground truth. Second, the security scanners are deliberately heuristic (Section 3), returning a hint (SUSPICIOUS / NO SIGNAL), never a verdict.

Table 2: The shared tool library the agents call.

Family	Tools	Backing
arithmetic	`calc`, `lookup`	evaluator + constant table
SQL	`list_tables`, `schema`, `run_sql`	in-memory SQLite (3 tables)
security	`scan_sql`, `scan_context`, `security_scan`	signature scanners (imperfect)
web / URL	`websearch`, `fetch`	offline index of real pages
business	`find_customer`, `list_invoices`, `get_payment`, `fx_rate`, `account_discount`	customer / finance store
failover	`cache_get`, `mirror_get`, `authdb_get`	tiered read with faults
records	`create_invoice`, `schedule_shipment`, `book_appointment`, `configure_alert`, `register_device`	typed record creation

5 Metrics

Beyond a single accuracy number, each rollout emits a small set of canonical behavioral metrics that aggregate across families:

- **Accuracy** (success rate): fraction of tasks whose extracted answer matches the verifier gold, reported per family and per category with Wilson 95% score intervals [Wilson, 1927, Brown et al., 2001], which are well-behaved at the small per-family sample sizes here.
- **URL grounding**: for the grounding families, the **hallucination rate** [Ji et al., 2023] (a fabricated argument that is neither a real indexed URL nor was returned by any search) and the **source-match rate** (the cited answer host equals the gold host).
- **Safety**: for the security families, a **violation rate** (proceeding on a truly unsafe artifact) and, where applicable, an **over-refusal rate** (blocking a benign one).
- **Cost**: LLM calls and input/output token counts per model, captured by the platform on the same runs (Section 6).

All metrics come from the github repo, so every reported number is reproducible and the released harness.

6 Results: Four Models on the Benchmark

We evaluate all 13 families across four models —`Qwen3.7-Max`, `gemma-4-31B-it`, `gemma4:26b`, and `gpt-oss-120b`—each on the same $2{,}057$ held-out tasks. Task conventions (e.g. the SQL cents/cancelled rules) are *stated* in the prompt, so a score measures the underlying skill.

Findings. Table 3 gives each model a distinct **capability** profile (Figure 1), and the story is in the per-family spread rather than the aggregate. Overall the four models rank `Qwen3.7-Max` (0.88) $>$ `gemma-4-31B` (0.86) $>$ `gemma4:26b` (0.82) $>$ `gpt-oss-120b` (0.71), but the top two are within two points and no model dominates everywhere. `gpt-oss-120b` is a telling outlier that *inverts* the usual ordering: it is the best of the four on both grounding families (`ug_fetch` 0.78, where every other model sits at 0.43–0.53; `ug_imposter` 0.89) yet by far the worst on the structured and multi-step families (`sql_analytics` 0.24, `tool_error_recovery` 0.25, `multi_tool_planning` 0.44)—a profile an overall score would hide entirely. The families that discriminate most widely are exactly these tool/structured tasks: `sql_analytics` spans 0.24–0.87, `tool_error_recovery` 0.25–0.76, and `multi_tool_planning` 0.44–0.96, so a single family separates the field by more than sixty points. `gemma-4-31B` leads on `orchestrator` (0.95) and ties for the top on the security and dependency-planning families, while the small local `gemma4:26b` is competitive throughout (0.82 overall) and even leads `ar_math` (0.91) and `multi_tool_planning` (0.96). A few families are saturated across all four (`sec_sqli` 0.94–0.99) and serve as sanity checks rather than ranking signal. The platform captures the **operational** side of the same runs (Table 5): every model issues 7k–8k LLM calls over the suite, but token usage varies almost $2\times$—`gemma-4-31B` spends the most (10.1M) and `gemma4:26b` the least (5.4M), and `gpt-oss-120b` burns 8.8M for the lowest accuracy. Capability and cost are thus read from one instrumented stream rather than separate experiments.

Table 3: Test-set accuracy (success rate on $[0, 1]$) per family across four models, computed from the committed test sets in `data/eval` (2,057 tasks per model). Best per family in bold. The wide per-family spread—not an aggregate—is the benchmark's ranking signal.

Family	Category	Qwen3.7-Max	gemma-4-31B	gemma4:26b	gpt-oss-120b
ar_math	arithmetic	0.90	0.86	**0.91**	0.69
ar_multistep	arithmetic	**0.97**	**0.97**	0.85	0.65
sql_analytics	structured	**0.87**	0.81	0.77	0.24
sec_sqli	security	**0.99**	**0.99**	0.95	0.94
sec_guard	security	**0.99**	0.96	0.93	0.85
sec_context_poison	security	**0.92**	0.91	0.84	0.82
ug_fetch	url-grounding	0.45	0.53	0.43	**0.78**
ug_imposter	url-grounding	0.82	0.78	0.56	**0.89**
dependency_planning	planning	**0.99**	**0.99**	0.96	0.68
multi_tool_planning	planning	0.94	0.80	**0.96**	0.44
orchestrator	orchestration	0.86	**0.95**	0.91	0.94
tool_error_recovery	tool-policy	**0.76**	0.74	0.65	0.25
tool_arguments	tool-policy	**0.99**	0.88	0.97	**0.99**
overall	(task-weighted)	**0.88**	0.86	0.82	0.71

Table 4: The same accuracies collapsed to the seven **task categories** (each cell is the task-weighted mean over that category's families; n is its total test tasks). This is the level the radar in Figure 1 plots; best per category in bold.

Category	n	Qwen3.7-Max	gemma-4-31B	gemma4:26b	gpt-oss-120b
arithmetic	306	**0.93**	0.91	0.88	0.67
structured	157	**0.87**	0.81	0.77	0.24
security	459	**0.97**	0.95	0.91	0.87
url-grounding	305	0.63	0.65	0.49	**0.83**
planning	332	**0.97**	0.90	0.96	0.57
orchestration	196	0.86	**0.95**	0.91	0.94
tool-policy	302	**0.88**	0.81	0.81	0.62
overall	2,057	**0.88**	0.86	0.82	0.71

7 The BekchiAI-Platform

BekchiAI-Platform is a drop-in observability and control layer. Enabling telemetry points a lightweight SDK at the platform; the agent's task logic is unchanged. This is what lets us read capability and cost (Section 6) from the same run, and it adds security observability and a **remote kill-switch** on top.

Architecture and event model. The platform has four parts: a **client SDK** that emits telemetry and polls for control commands; an **ingestion + API** and REST endpoints for agents, sessions, traces, and findings; a store with a wide `events` table (JSONB payloads plus relational metadata); and a **React dashboard**. Ingestion is idempotent on a event id, authenticated with per-project API keys, and multi-tenant. Agent activity is modelled as *sessions* of linked events forming a trace tree. Core types are `session_start`; `llm_call` (model, input/output tokens, latency); `tool_call` / `tool_result` (name, arguments, success); `url_access` (host, status, allowlisted); `plan` (intended tools and URLs); `denied` (policy events); and `outcome`. Table 6 shows how each benchmark behavior maps onto these events, so no separate evaluation harness is needed.

Behavioral-security analytics. Over the event store the platform computes two signals. *Argument grounding*: from each session's `plan` and allowlisted accesses it builds a trusted-host set and labels each accessed host grounded, ungrounded, or `imposter_url` (an ungrounded host within Levenshtein distance ≤ 2 of a trusted one—a likely typosquat), reporting the grounding rate.

Plan-vs-actual consistency: it compares the intended tool set to the executed `tool_call` sequence, yielding intersection / unplanned / missing sets and a per-session verdict. These are the platform-side counterparts of the benchmark's grounding and tool-adherence metrics.

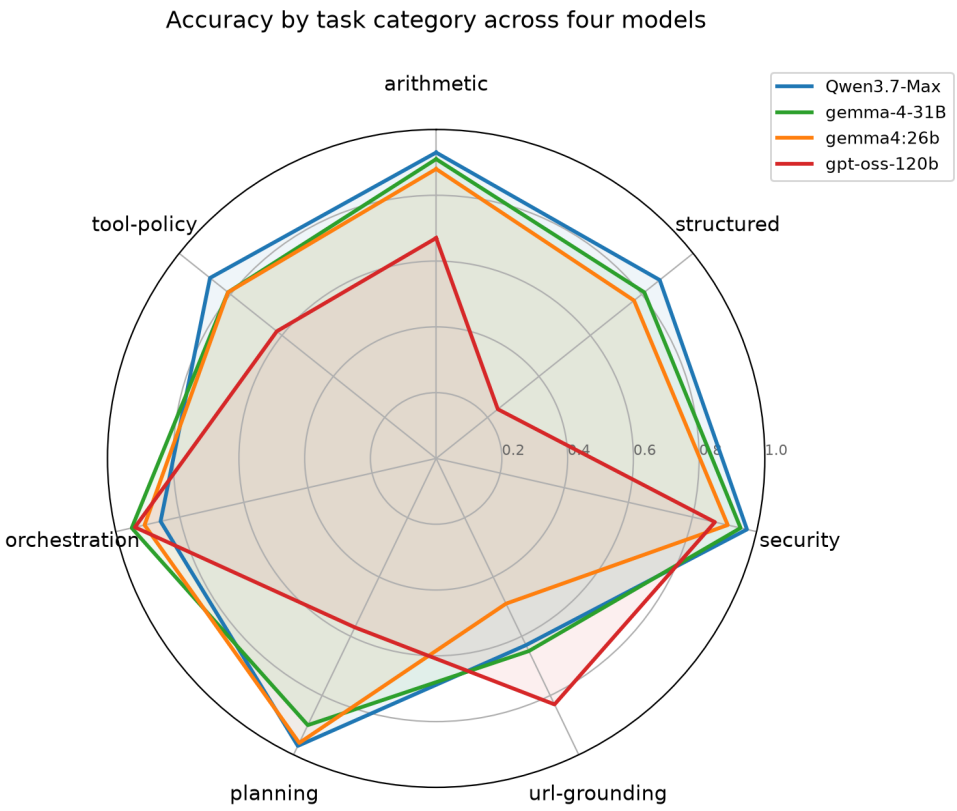

Figure 1: Zero-shot accuracy by task category for the four models—one axis per category (the task-weighted mean over its families of Table 4; radius = success rate on $[0, 1]$). The polygons expose each model's *profile*: `gpt-oss-120b` (red) caves in on the structured (0.24), planning (0.57), and tool-policy (0.62) categories yet reaches farthest on url-grounding (0.83), while the other three trace a broadly similar high envelope—a shape an overall score cannot convey.

Table 5: Operational cost of the same runs, measured by the platform: LLM calls and token usage per model over the full 2,057-task suite (input + output tokens in millions). Read alongside Table 3, this shows capability and cost from one instrumented stream: `gemma-4-31B` and `gpt-oss-120b` spend the most tokens without leading on accuracy, while `gemma4:26b` is the leanest.

Model	LLM calls	input (M)	output (M)	total (M)
Qwen3.7-Max	**6,965**	**3.86**	1.96	5.83
gemma-4-31B	7,715	8.32	1.80	10.12
gemma4:26b	7,258	4.61	**0.82**	**5.43**
gpt-oss-120b	7,158	6.56	2.25	8.81

Policy enforcement and remote control. Operators define `allow/block/alert` policies over host globs and tool names; at ingest, `block` matches emit a synthetic `denied` event, giving a durable audit of prevented actions. The platform is also bidirectional: an operator issues `stop/pause/resume/revoke` from the dashboard, the agent polls the control endpoint, applies pending commands, and acknowledges; commands move `pending`→`delivered`→`applied` with timestamps. In the benchmark run this let us abort a running agent remotely within a single poll cycle. Enforcement is cooperative: an agent that cannot reach the channel is not controllable.

8 Conclusion

We introduced the BekchiAI-Benchmark, a suite of 13 tool-using ReAct agents across seven categories (2,057 committed tasks) that measures the *agentic skills* of LLMs—tool selection and sequencing, planning under dependencies, security

Table 6: How each benchmark behavior maps onto BekchiAI-Platform signals. One instrumented run yields all four.

Model behaviour	Platform signal (event)	Reveals
task correctness	`outcome` (success/mismatch)	accuracy, per family
model call	`llm_call` (input/output tokens, latency)	token usage and speed
tool use	`tool_call`, plan-vs-actual	tool competence, adherence
URL arguments	`url_access` (grounded/hallucinated)	URL-hallucination, source-match

judgment, and argument grounding—rather than a single blended accuracy. Three design choices keep the suite honest: gold is computed by a verifier (canonical SQL, the exact schedule of a dependency DAG, or a closed-form lambda) and so cannot drift; the discriminating cases are hand-authored one by one rather than templated; and the security families replace the usual oracle detector with a deliberately imperfect signature scanner, so a score reflects the model's own judgment rather than tool-copying. We pair accuracy with behavioral metrics—tool-call adherence, plan-vs-actual consistency, URL grounding, and per-model token cost—all read from one instrumented stream via the drop-in BekchiAI-Platform.

Across four models, the results argue for reporting the *per-family spread* rather than an aggregate. The top two models finish within two points overall, yet no model dominates every skill: `gpt-oss-120b` leads both grounding families while collapsing on the structured and multi-step tool tasks (e.g. `sql_analytics` 0.24 vs. 0.87)—an inversion a single leaderboard number erases. The tool and structured families discriminate most sharply, separating the field by more than sixty points. Because the platform records token cost on the same runs, capability and efficiency can be weighed together—and the most accurate model is not the cheapest. As agents take on real, consequential actions, we argue that skill- and cost-resolved, hard-to-game evaluation of this kind is a prerequisite for trustworthy deployment.

The benchmark is deliberately compact and evaluates one task per rollout; natural extensions include broader model coverage, additional adversarial rounds and skill categories, longer-horizon and multi-turn interactions, and applying the platform's grounding and plan-vs-actual analytics to live agent deployments. We release the benchmark, tool library, metric scripts, and we released to platform on web and mobile markets to support this work.

Reproducibility. The benchmark—provided test sets, the tool library, harness, and metrics—is released at `https://github.com/bekchiai/bekchiai-benchmark`.

References

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

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 *International Conference on Learning Representations (ICLR)*, 2024.

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

Gartner. Gartner predicts 40% of enterprise apps will feature task-specific AI agents by 2026, up from less than 5% in 2025. Gartner Newsroom, press release, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics (ACL)*, 2024.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

Lawrence D Brown, T Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.