

000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 OPAQUETOOLS BENCH: LEARNING NUANCES OF TOOL BEHAVIOR THROUGH INTERACTION

Anonymous authors

Paper under double-blind review

ABSTRACT

Tool-calling is essential for Large Language Model (LLM) agents to complete real-world tasks. While most existing benchmarks assume simple, perfectly documented tools, real-world tools (e.g., general “search” APIs) are often *opaque*, lacking clear best practices or failure modes. Can LLM agents improve their performance in environments with opaque tools by interacting and subsequently improving documentation? To study this, we create OPAQUETOOLS BENCH, a benchmark consisting of three distinct task-oriented environments: general function calling, interactive chess playing, and long-trajectory agentic search. Each environment provides underspecified tools that models must learn to use effectively to complete the task. Results on OPAQUETOOLS BENCH suggest existing methods for automatically documenting tools are expensive and unreliable when tools are opaque. To address this, we propose a simple framework, TOOLOBSERVER, that iteratively refines tool documentation by observing execution feedback from tool-calling trajectories. Our approach outperforms existing methods on OPAQUETOOLS BENCH across datasets, even in relatively hard settings. Furthermore, for test-time tool exploration settings, our method is also efficient, consuming 3.5-7.5 \times fewer total tokens than the best baseline.¹

1 INTRODUCTION

Tools expand the knowledge and capabilities of Large Language Model (LLM) agents beyond their learned parameters (Schick et al., 2023). With the right tools, LLMs can search the web, send emails, execute code, and interact with the world. Yet this extension fails when tools lack adequate documentation – a pervasive problem in deployed systems. Real-world APIs are complicated to explain, enterprise functions lack specifications, and domain-specific tools ship with minimal descriptions. Moreover, some tools are difficult (or impossible) to fully explain, even for humans, e.g., often the optimal usage of search engines or LLM-based QA APIs is not known, even to their creators. Such tools are **opaque**: their behavior unpredictable, their correct usage unknown. As shown in Figure 1, tool opacity harms the performance of LLM agents, a problem that compounds when models must coordinate multiple tools for complex tasks.

This raises a fundamental question: **Can LLM agents learn to use opaque tools by observing their behavior during interaction?** Such capability would transform tool-calling agents from brittle systems dependent on perfect specifications into adaptive ones that can improve through experience.

We introduce OPAQUETOOLS BENCH, a benchmark for learning in opaque tool settings where tool documentation is underspecified. Different than existing benchmarks that assume near-perfect tool specifications like ToolBench (Qin et al., 2023), APIBench (Patil et al., 2023), and Berkeley Function Calling Leaderboard (Patil et al., 2025), OPAQUETOOLS BENCH does *not* provide comprehensive function signatures, detailed descriptions, or explicit type specifications. OPAQUETOOLS BENCH spans three distinct environments: general function calling, interactive chess playing, and long-trajectory agentic search. Each provides underspecified tools that models must learn to use effectively, testing both single-instance discovery and cross-instance learning. Unlike existing benchmarks that focus on measuring model capacity to compose concrete, well documented tools, OPAQUETOOLS BENCH is designed to measure an LLM agent’s ability to adapt to the imperfect documentation of tools through interaction.

¹We release our code, data, and benchmark at anonymous.com

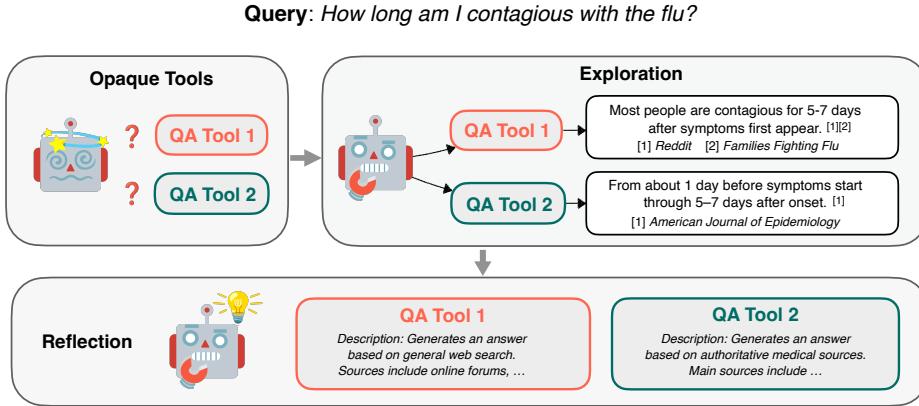


Figure 1: LLM agents may struggle when presented with *opaque tools* – tools lacking clear description of their usage best practices or their failure modes. To succeed in these settings, we posit that LLM agents must explore tool usage to learn their true behaviors.

We evaluate current approaches that optimize tool descriptions, including Play2Prompt (Fang et al., 2025) and EasyTool (Yuan et al., 2024). Both methods fall short on OPAQUETOOLSBENCH: systems either focus on compression of existing tool documentation, completely neglecting the interaction with tools (EasyTool) or require single-tool exploration phases separate from task execution, which becomes expensive in some settings (Play2Prompt). As a result, these methods are ineffective and sometimes expensive, often consuming thousands of tokens in preliminary exploration before attempting the composite task.

We propose an alternative framework, TOOLOBSERVER, that refines tool documentation by observing and learning from execution feedback acquired through composite task trajectories. On OPAQUETOOLSBENCH, our method exceeds baseline performance consistently by on average 18.6%, while consuming 3.5-7.5x fewer tokens in test-time settings. Our results demonstrate that learning from execution feedback provides an efficient path to handling opaque tools in real-world environments. It also demonstrates that LLM agents can adapt to underspecified tools through interaction, making tool-calling viable even in poorly-documented environments.

2 BACKGROUND

Language models estimate the conditional distribution $P(x_t|x_{<t})$ over a vocabulary \mathcal{V} , where $x_{<t}$ represents all preceding tokens (Radford et al., 2019; Brown et al., 2020). Despite their impressive results on language benchmarks, language models nonetheless face fundamental limitations: their knowledge is frozen, they can't take actions in the real world, and they are often unreliable on simple capabilities like adding numbers. These constraints have motivated the development of *tool-augmented* language models that invoke external functions to overcome these limitations (Schick et al., 2023; Mialon et al., 2023).

Tool-Calling in Language Models Language models interact with tools through a structured interface that enables them to extend their capabilities beyond parametric knowledge. Each tool t_i in the available tool library $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ is characterized by:

$$t_i = \{n_i, d_i, p_i, e_i\}$$

where n_i is the name of the function, d_i is the documentation of the function behavior, p_i is an optional dictionary of parameters consisting of their name, description, and whether or not they are required. e_i is the executable function itself. In practice, parameter information from p_i is often incorporated into the behavioral documentation d_i as a string description.

Given a user query q , the language model M generates tool calls through a systematic process of **retrieving** then **calling**, all through autoregressive decoding. The model conditions on both the query and the available tool descriptions in their context to produce a sequence that may include tool invocations:

$$s \sim P_M(s|q, \mathcal{T}) = P_M(s|q, (n_1, d_1, p_1), \dots, (n_n, d_n, p_n)) \quad (1)$$

108 where s is the sampled sequence. When the model determines a tool is needed, it produces a structured
 109 tool call $c = \langle n_i, \text{args}_i \rangle$ as part of this sequence, where args_i must conform to the parameter
 110 specification p_i . Upon generating a tool call, the system executes $r_i = e_i(\text{args}_i)$ and appends the
 111 result to the context. The model then continues its autoregressive generation, now conditioning on
 112 the expanded context:

$$s' \sim P_M(s' | q, \mathcal{T}, c, r_i) \quad (2)$$

115 This process may repeat, with the model invoking multiple tools or generating a final response that
 116 incorporates the tool outputs to address the user’s query. Such tool calling language models are often
 117 used to complete goal-oriented tasks and are referred to as LLM agents (Wang et al., 2023b).

119 3 THE OPAQUETOOLS BENCH BENCHMARK

121 Prior work creating and evaluating LLM agents assumes well-defined and well-documented tools
 122 (Guo et al., 2024a; Qin et al., 2023; Shen et al., 2024; Patil et al., 2023). However, many real-world
 123 tools are black boxes whose behavior can only be understood through interaction. As an example,
 124 consider an agent given access to a set of off-the-shelf Search APIs provided as tools. Such APIs may
 125 index documents at differing granularities and covering different domains/times/topics/styles, may
 126 (or may not) be keyword based, might automatically run multiple hops, and could even themselves
 127 use language models to orchestrate the search process. For LLM agents to perform optimally in
 128 such an environment, they need to learn these nuances by *interacting with these tools and observing*
 129 *the feedback*. Furthermore, the proliferation of Model Context Protocols (MCP)² have made it easy
 130 to connect LLM agents to tools that come with variable quality and accuracy of tool documentation.

131 We characterize such tools as **opaque**. Reflecting on the diverse challenges described above, from
 132 the inherent complexity of Search APIs to the variable documentation quality of MCP tools, we find
 133 that opacity stems from two distinct sources. We distinguish between two types of opacity – both
 134 practically important – that motivate our benchmark design:

- 135 • **Type 1: Documentation Opacity** Tools whose behavior is deterministic and describable, but
 136 whose documentation is inaccurate or missing. This is often inevitable in real-world settings:
 137 legacy systems often rely on unwritten “tribal” knowledge, while open marketplaces like the
 138 Model Context Protocol (MCP) contain inconsistent descriptions at scale.
- 139 • **Type 2: Intrinsic Opacity** Tools that are opaque due to inherent complexity (e.g., search
 140 engines, LLM-based tools, complex simulations). These tools often have a simple schema but
 141 complex, undocumented behavioral nuances. For example, a search API’s ranking logic is hard
 142 to fully capture a priori; even the tool creator cannot predict how a neural system handles every
 143 edge case. The agent must instead learn these nuances through interaction.

145 To effectively operate in environments characterized by these forms of opacity, LLM agents need to
 146 demonstrate the following abilities:

- 148 1. **Manipulate structured & natural language inputs:** Certain tools that expose traditional
 149 REST APIs (for example, currency conversion) have structured inputs. Whereas others like
 150 Search APIs accept a string where the nuances are encoded in natural language, which are
 151 more open-ended/opaque.
- 152 2. **Learn from process feedback:** Opaque tools can sometimes only be understood by observ-
 153 ing a trajectory/sequence of tool uses, rather than just the result from a single call, e.g., you
 154 might not be able to discern if a search API’s output is “good” until you try to compose the
 155 result with other information.
- 156 3. **Learn across trajectories:** In many real-world settings with a fixed set of tools, LLM agents
 157 need to accumulate experience from prior trajectories, e.g., using a search tool with one goal
 158 in mind should help one gain experience useful for using that tool for a different goal.
- 159 4. **Test-time generalization:** In settings where new tools are available at test time, we need to
 160 test the ability of LLM agents to learn their nuances while completing the task itself.

161 ²modelcontextprotocol.io

162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
Table 1: An overview of the datasets in OPAQUETOOLS BENCH

	BFCL-Opaque	Chess	BrowseComp Domains
Description	Complete user’s request by correctly calling a function with a limited description.	Play chess by choosing one of many specialized chess engine tools every turn.	Compose domain-specific search tools to locate hard-to-find information.
Skills required	Structured inputs, process feedback, test-time generalization	Process feedback, learn across trajectories, tool sequencing	Unstructured inputs, process feedback, learn across trajectories, tool sequencing
Settings	Documentation quality 1. Anon. function names 2. Anon. function names + description 3. Anon. function names + parameter names	Tool sets (chess engines) 1. Beginner, intermediate, advanced skill 2. Opening, midgame, endgame, late-endgame specialists	Tool sets (search tools) 1. Domain-specific (9) 2. Domain-specific (9) + Full Search
Evaluation	Evaluation Accuracy, Param Acc., AST Acc.	% of Optimal Tool Calls, ELO	Accuracy, # Tool Calls
# Train / # Test	- / 90	200 / 1800	83 / 747

5. **Learn tool sequencing:** Certain tool behaviors may only manifest in conjunction with other specific tool calls. For example, if tool B can only be called after a successful invocation of tool A, the agent needs to be able to create such trajectories to learn nuances of tool B.

We introduce OPAQUETOOLS BENCH to systematically evaluate these abilities across both types of opacity. It consists of three environments: an opacified version of BFCL (Patil et al., 2023) we call BFCL-Opaque (targeting **Type 1**), a novel game-playing environment based on Chess (targeting **Type 2**), and BrowseComp Plus with opaque search (Chen et al., 2025) (targeting **Type 2**). Our environments are designed to test the above aspects of learning the behavior of opaque tools while being reproducible and efficient to run. A summary of our datasets and key information is shown in Table 1 and in the following paragraphs. Examples from the tasks are shown in Figure 2.

BFCL-Opaque: The Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2025) consists of question-functions-answer tuples with functions from multiple programming languages and diverse application domains. Following Fang et al., 2025, we use the executable subset so feedback is available from executing the tools. The tasks are simple, for example, fetching the current weather or computing the area of a polygon. Each task has between 2-4 tools and on average 3. We modify this environment to evaluate **test-time generalization** and **Type 1 Opacity** – though the tools of each instance may slightly overlap a few others instances’, we instead treat the tools of each test instance as independent and unknown. Specifically, we modify this environment by obfuscating the function names as well as completely removing all docstrings. We do this independently for each problem – so the same function across different test instances will be named differently. For each test instance, the agent now has to learn what the function does and produce structured output in the form of its arguments and their types. We create progressively easier settings by providing either (1) only function description or (2) only parameter names. We use a binary task completion rate as our evaluation metric. Refer to Appendix C.1 for details on environment construction.

Chess In this environment the task is to win against a fixed-strength chessbot (Stockfish (Romstad et al., 2024) at search depth 2; higher=better). Instead of playing moves directly (as in Zhang et al. (2025)), models are given access to a set of move suggesting tools. At each turn, the LLM agent uses one of several undocumented tools that accept current board positions in FEN notation and play the move that’s suggested. While the tool interfaces are identical, we introduce undocumented behaviors in each tool. Specifically, in the first setting, each tool uses Stockfish but with different search depths (2, 4 & 8), testing if the agent can learn fine-grained discrimination between tools based on the final outcome. In the second setting, we provide tools optimized for opening, middlegame, endgame, and late endgame, enabling us to test the ability to explore and learn temporal patterns. Since the same set of tools are used across different instances, we test agent’s ability to learn tool behavior across trajectories (capturing **Type 2 Opacity** where behavior depends on game state). Refer to Appendix C.2 for details on environment construction.

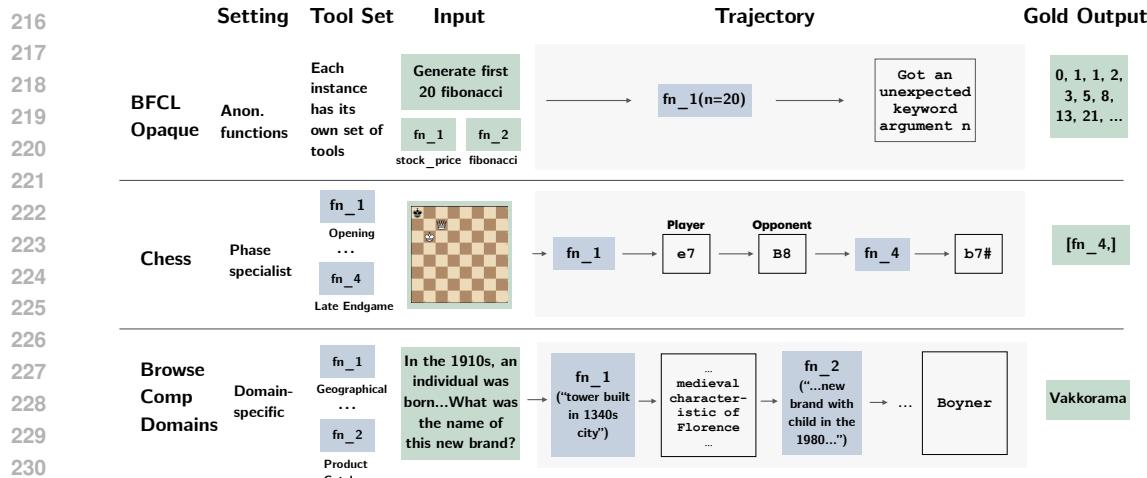


Figure 2: Examples from each of the environments in OPAQUETOOLSBENCH. For Chess and BrowseComp Domains, each setting has a fixed set of tools across all instances (chess and search engines respectively). In BFCL-Opaque, each instance has custom query-dependent tools. The dataset defines the input and output (green). For each instance, the agent makes tool calls iteratively (blue). For BFCL-Opaque and BrowseComp Domains we check for a match with the gold answer. For Chess, we match engine choices in green with optimal engine choices.

BrowseComp Domains The BrowseComp dataset consists of question and short answer pairs which measure the ability of agents to locate hard-to-find, entangled information on the internet, and might require browsing tens or even hundreds of websites in the process. BrowseComp Plus (Chen et al., 2025) further improves it by fixing the retrieval corpus and thus making this benchmark easy to reproduce. We leverage this corpus and create anonymous domain-specific search tools (4-6 tools) by partitioning the corpus (academic papers, product catalogs, geographical data, news articles). The LLM agent must not only discover the domain specialization of each tool, it must also learn to create optimal search queries for each tool and learn to search in the right sequence (addressing **Type 2 Opacity** in query formulation). Each instance in our dataset has a corresponding sequence of optimal tool calls. We compare the tool sequence with the optimal sequence and use accuracy as our evaluation metric. Refer to Appendix C.4 for details on environment construction.

These three environments test for the four key abilities of LLM agents: (1) With BFCL-Opaque we test for structured inputs and with BrowseComp Domains unstructured inputs, (2) All environments allow learning from intermediate outputs of tool calls, (3) BrowseComp Domains and Chess test if the LLM agent can learn across trajectories, while BFCL-Opaque checks for generalization to new tools at test time, (4) Chess and BrowseComp Domains test for LLM agent ability to sequence opaque tools in the right sequence.

4 TOOLOBSERVER: IMPROVING DOCUMENTATION VIA INTERACTION

Existing approaches for optimizing tool documentation have limitations that render them less effective in complex tool-calling environments. For example, EasyTool (Yuan et al., 2024) relies primarily on compressing existing long documentation of tools into more concise instructions, making it unsuitable to settings where the tool documentation is underspecified or completely lacking. Meanwhile, Play2Prompt (Fang et al., 2025) adopts an iterative refinement strategy, but it requires an isolated single-tool exploration phase separate from task execution, making it inefficient and (as we will show) not performant in complex environments that require a long trajectory of tool calls.

To address these limitations, we propose TOOLOBSERVER. TOOLOBSERVER requires no up-front documentation, and discovers and refines tool documentation through observation and reflection on execution trajectories. The key idea is to alternate between an *Exploration Phase* and a *Reflection Phase*. During exploration (line 3 of Algorithm 1), we collect execution trajectories using the current documentation $\mathcal{D}^{(k-1)}$, which can initially be null. There a separate reasoning model analyzes the

270 trajectories, identifies patterns, and updates tool descriptions accordingly. We repeat this process
 271 iteratively, improving tool descriptions and exploring better trajectories in the next iteration.
 272

273 **OPAQUETOOLS**BENCH contains two settings: (1) with **shared tools** between train and test set and
 274 (2) with **unseen tools** that can only be observed at test time (and there is no train set). We in-
 275 stantiate **TOOLOBSERVER** in two modes to accommodate the two settings: **offline mode**, which
 276 pre-optimizes documentation during training when all test instances share a common tool set and
 277 **online mode**, which optimizes documentation at test time for previously unseen tools.
 278

279 4.1 OFFLINE MODE

280 In offline mode, we pre-optimize docu-
 281 mentation using a set of training instances
 282 sharing the same tools. Algorithm 1 out-
 283 lines the procedure. The core intuition
 284 is to establish a feedback loop where the
 285 agent attempts tasks, observes execution
 286 outcomes against ground truth, and iter-
 287 atively refines its understanding of tool be-
 288 haviors from this signal. This approach
 289 leverages two key advantages of the offline
 290 setting: (1) multiple training instances and
 291 (2) the ability to evaluate the final answer.
 292 We discuss the two key phases below:

Algorithm 1 **TOOLOBSERVER**: Offline mode

Require: Initial descriptions \mathcal{D}_0 , iterations K , LLM
 Agent M_A , Editor LLM M_E
Ensure: Optimized documentation \mathcal{D}^*

```

1: Initialize  $\mathcal{D}^{(0)} \leftarrow \mathcal{D}_0$ 
2: for  $k = 1$  to  $K$  do
3:   // Exploration Phase
4:    $\mathcal{T}_k \leftarrow \text{CollectTrajectories}(\mathcal{D}^{(k-1)}, M_A)$ 
5:   // Reflection Phase
6:    $\mathcal{D}^{(k)} \leftarrow \text{ReflectAndUpdate}(\mathcal{T}_k, \mathcal{D}^{(k-1)}, M_E)$ 
7: end for
8: return  $\mathcal{D}^{(K)}$ 

```

293 **Exploration Phase: Collecting Trajectories** The goal of this phase is to generate diverse inter-
 294 action traces that reveal the latent behaviors and failure modes of the opaque tools. To do this, we
 295 execute the agent M_A on the training set \mathcal{X}_{train} using the current tool documentation $\mathcal{D}^{(k-1)}$. By
 296 running across the entire training set, we ensure the agent explores tool behaviors across a wide
 297 variety of contexts (e.g., diverse chess positions or search queries) and diverse usage patterns. We
 298 use temperature sampling to collect a breadth of trajectories $\mathcal{T}_k = \{\tau_1, \dots, \tau_{|\mathcal{X}_{train}|}\}$, where each
 299 trajectory τ_i captures the full sequential decision process: starting from the initial state, it records
 300 the alternating sequence of reasoning steps, tool calls, and environmental observations (e.g., tool
 301 execution outputs) leading to the final outcome.

302 **Reflection Phase: Reflecting and Updating Documentation** The goal of this phase is to distill
 303 raw interaction experience into explicit, generalizable tool documentation. However, processing the
 304 full set of trajectories \mathcal{T}_k simultaneously is infeasible due to context window constraints and the
 305 difficulty of extracting consistent patterns from massive, noisy data streams. To address this, we
 306 employ a meta-prompting strategy with an “**Editor**” LLM M_E (this can be the same as our LLM
 307 agent M_A). It analyzes trajectories via a hierarchical process:

- 308 **1. Batch Analysis (with ground truth):** We first split the trajectories into mini-batches to manage
 309 context limits. For each batch, we explicitly task the Editor with updating the tool descriptions
 310 based on the execution history. Acting as a local reasoner, it identifies causal links between tool
 311 usage patterns and success/failure outcomes. Crucially, it is provided with the **ground truth**
 312 (e.g., the gold answer) or a **task performance signal** derived from the training environment.
 313 Using this signal, it distinguishes effective usage from failures and generates a candidate descrip-
 314 tion specific to that batch. Consequently, this phase produces a diverse set of local descriptions,
 315 where each captures insights valid for its specific subset of trajectories.
- 316 **2. Consensus Merge:** Since observations from a single batch may be noisy or overfit to specific
 317 instances, a second Editor pass aggregates the candidate proposals from all batches. It acts as a
 318 **consensus filter**, retaining only those behavioral rules that appear consistently across multiple
 319 diverse batches while discarding instance-specific hallucinations.

320 This two-stage process is critical for the offline mode: it allows us to scale to large datasets while
 321 ensuring the learned documentation generalizes across different contexts. We repeat this explo-
 322 ration and reflection loop for K iterations. Finally, we run the LLM agent with the optimized
 323 documentation $\mathcal{D}^{(K)}$ on the test set. Further implementation details, including prompt templates
 and dataset-specific details, can be found in Appendix A and E.

324 4.2 ONLINE MODE
325

326 In the online mode, we are only given access to a single test instance with a set of unseen tools. For
327 a single test instance with unique tools \mathcal{T}_x that cannot be experimented with *a priori*, we first collect
328 a trajectory by executing the agent’s generated tool calls and recording the real-time environment
329 feedback (e.g., return values or error messages). Then, we use the **Editor LLM** to analyze that
330 trajectory and update the tool documentation based on the observed behavior(s). We repeat this
331 process for a maximum of K iterations. If the Editor does not update the documentation, we stop
332 the iteration early. Note that in online mode the gold output is not used as part of the process. We use
333 the final version of the tool documentation to run the LLM agent on the test instance and compute
334 an evaluation metric using the gold output.

335 4.3 COMPARISON TO PLAY2PROMPT
336

337 While Play2Prompt (Fang et al., 2025) also adopts an iterative refinement strategy, our method
338 differs from it in several crucial ways. First, Play2Prompt requires initial documentation D_0 to
339 bootstrap their reverse generation process; they first generate valid tool invocations based on this
340 documentation, then construct matching queries. In contrast, our method can operate under the
341 more challenging setting where tools are completely opaque, requiring discovery purely through
342 task-driven exploration. Second, Play2Prompt also requires optimization of all tools individually
343 beforehand. In contrast, our method explores all tools at once, balancing an exploration/exploitation
344 tradeoff and tool interactions vs. being forced to exhaustively explore all tools individually *a priori*.
345 In §5, TOOLOBSERVER’s design leads to notably better performance under opaque tool settings.

346 5 EXPERIMENTS
347

348 We benchmark three contemporary tool-calling LLM agents on OPAQUETOOLSBENCH. First, we
349 test on GPT-5 (OpenAI, 2025), the most capable model OpenAI model for reasoning and agentic
350 tool use at the time of writing. We also use GPT-5-mini, a cost-efficient yet still capable version. We
351 use the ReAct framework (Yao et al., 2022) to iteratively reason and call functions. Further details
352 can be found in Appendix A.

353 **TOOLOBSERVER and Baselines Experimental Details** For our main experiments with TOOLO-
354 SERVER, we also use GPT-5 as our model for reflecting on and updating tool descriptions (editor
355 model). Following TOOLOBSERVER, for all baselines we use GPT-5. We include (1) **Play2Prompt**
356 (Fang et al., 2025), which improves tool-documentation from self-play followed by self-reflection
357 and (2) **EasyTool** (Yuan et al., 2024) which automatically rewrite the tool documentation by
358 condensing tool descriptions and creating structured functional guidelines.

361 5.1 MAIN RESULTS
362

363 Our main results in Tables 2-4 demonstrate both the challenge of existing baselines on OPAQUE-
364 TOOLS BENCH, and the strong performance of TOOLOBSERVER, showing its potential to improve
365 LLM agent performance in opaque tool settings.

366 **TOOLOBSERVER outperforms baselines on BFCL-Opaque and recovers near-optimal perfor-
367 mance.** Table 2 demonstrates that TOOLOBSERVER outperforms all baselines across documenta-
368 tion levels and LLM agents. In the function name only setting, TOOLOBSERVER recovers 0.62
369 execution accuracy with GPT-5, whereas Play2Prompt only achieves 0.50. Furthermore, because
370 Play2Prompt exhaustively test hundreds of tools, most of which will not be relevant, the total input
371 + output tokens consumed for exploration is $\sim 1.7M$, which is $7.5 \times$ more than the exploration bud-
372 get of TOOLOBSERVER. Across documentation levels, this ranges from $3.5 \times - 7.5 \times$. This indicates
373 that for test-time optimization settings, our method is both effective and relatively inexpensive.

374 We identify an initially challenging but tractable scenario: when parameter information is missing.
375 OpenAI agents initially obtain 0 performance, repeatedly calling tools without arguments. How-
376 ever, TOOLOBSERVER effectively bridges this gap. By comparing against a Gold Oracle (perfect
377 documentation), we find our method recovers 93% of the performance gap in the underspecified
setting (0.86 vs. 0.92). To verify this is due to valid schema discovery rather than luck, we an-

378
 379
 380
 381
 382
 383 Table 2: Performance on BFCL across models and baselines. We use ReAct. Gold is the ground
 384 truth documentation and base is the opacified set. TO denotes TOOLOBSERVER, P2P denotes
 385 Play2Prompt, and ET denotes EasyTool. Columns denote **E** (Execution-based overall accuracy),
 386 **P** (Parameter accuracy), and **A** (AST accuracy). The highest **E** value in each row is **bolded**.
 387
 388
 389
 390
 391

Documentation	Model	ReAct														
		Gold			Base			+ TO			+ P2P			+ ET		
		E	P	A	E	P	A	E	P	A	E	P	A	E	P	A
Tool Setting: Individual Tools per Problem																
Anon. Fn.	GPT-5	0.92	0.91	0.93	0	0	0.59	0.62	0.61	0.83	0.50	0.54	0.80	0	0	0.59
Names Only	GPT-5-mini	0.94	0.90	0.95	0	0	0.59	0.62	0.54	0.82	0.52	0.54	0.82	0	0	0.59
Anon. Fn. +	GPT-5	0.92	0.91	0.93	0	0	0.59	0.86	0.82	0.89	0.50	0.47	0.80	0	0	0.60
Real Desc.	GPT-5-mini	0.94	0.90	0.95	0	0.01	0.60	0.80	0.77	0.90	0.46	0.45	0.79	0	0	0.60
Anon. Fn. +	GPT-5	0.92	0.91	0.93	0.78	0.82	0.93	0.82	0.85	0.94	0.78	0.80	0.94	0.70	0.71	0.90
Param Names	GPT-5-mini	0.94	0.90	0.95	0.82	0.85	0.94	0.88	0.88	0.96	0.84	0.84	0.95	0.74	0.76	0.93

392
 393
 394 Table 3: Average percentage of best tool calls (Acc) and ELO rating at a given state for Chess. TO
 395 denotes TOOLOBSERVER, P2P denotes Play2Prompt, and ET denotes EasyTool. Gold is the ground
 396 documentation. The highest value in each row is **bolded**.
 397

Model	ReAct									
	Gold		Base		+ TO		+ P2P		+ ET	
	Acc	ELO	Acc	ELO	Acc	ELO	Acc	ELO	Acc	ELO
Tool Setting: Opening, midgame, endgame, late-endgame specialists										
GPT-5	64.4	1411	23.5	728	40.1	1020	35.8	966	23.2	761
GPT-5-mini	52.8	1243	24.9	772	32.1	949	19.5	754	25.8	739
Tool Setting: Beginner, intermediate, advanced skill										
GPT-5	100	2341	24.9	1572	29.1	1756	0.25	1622	24.9	1584
GPT-5-mini	100	2346	25.7	1674	28.4	1778	22.7	1481	25.5	1645

408
 409
 410 alyze Parameter (P) and AST (A) accuracy. TOOLOBSERVER consistently outperforms baselines
 411 on these granular metrics (e.g., 0.61 Parameter Accuracy vs. 0.54 for Play2Prompt), confirming it
 412 successfully synthesizes correct input structures from scratch.

413 **BrowseComp Domains and Chess are challenging, but TOOLOBSERVER can still discover tool**
 414 **nuances** As shown in Table 3 and 4, BrowseComp Domains and Chess are empirically much harder
 415 tasks than BFCL Opaque – for Chess, the best performance for any model, measured by percentage
 416 of best tool calls, is only 40.1%. On the other hand, the best accuracy for BrowseComp Domains is
 417 only 24.1%. In both of these cases, despite the initial difficulty of the tasks, our method TOOLO-
 418 SERVER outperforms baselines across tool settings in both domains, achieving the best performance.

419 However, even the optimized chess performance is seemingly low - with only 29.1 on the skill tool
 420 setting with only three tools with GPT-5. We posit that this is because our evaluation metric is strict
 421 – during exploration, the model observes tool usage from three different tools, all of which will
 422 perform well against the opponent. When evaluating via *best tool %* , they are penalized even if
 423 they choose a relatively good tool. In contrast, the Streaming ELO scores – computed from win-
 424 rates against diverse opponents – verify that the learned documentation enables practically effective
 425 gameplay, improving over the baseline even if the strictly optimal tool is not always chosen.

426 For BrowseCompPlus, comparisons against the Gold Oracle (33.1% accuracy for GPT-5) reveal that
 427 this task is intrinsically difficult even with perfect documentation. TOOLOBSERVER recovers some
 428 of this gap while also reducing the number of average tool calls. This intuitively makes sense: as
 429 documentation is improved for search tools, the LLM agent is able to route queries more effectively,
 430 both improving downstream performance and reducing wasted search calls. Overall, the success of
 431 TOOLOBSERVER across both offline settings demonstrate the strength of our method: reflecting on
 and adapting based on real trajectories improves performance better than isolated tool testing.

432
 433 Table 4: Performance comparison table on BrowseComp Domains using Qwen-0.6B. Acc. refers
 434 to accuracy and #TC refers to number of tool calls. TO denotes TOOLOBSERVER, P2P denotes
 Play2Prompt, and ET denotes EasyTool. The highest Acc. value in each row is **bolded**.

Model	ReAct									
	Gold		Base		+ TO		+ P2P		+ ET	
	Acc.	#TC	Acc.	#TC	Acc.	#TC	Acc.	#TC	Acc.	#TC
Tool Setting: Domain-specific (9) Search										
GPT-5	30.8	22.6	18.8	25.3	20.3	23.2	19.4	23.8	18.7	23.8
GPT-5-mini	28.9	21.8	14.6	23.2	18.8	22.9	15.1	12.1	15.0	24.4
Tool Setting: Domain-specific (9) + Full Search										
GPT-5	33.1	20.5	21.4	24.8	24.1	21.9	23.8	14.4	20.6	25.5
GPT-5-mini	30.9	22.2	20.3	23.3	22.1	21.0	21.0	18.6	20.8	23.7

446 5.2 ANALYSIS

447
 448 We analyze a few components of
 449 TOOLOBSERVER and our own OPAQUE-
 450 TOOLS BENCH:

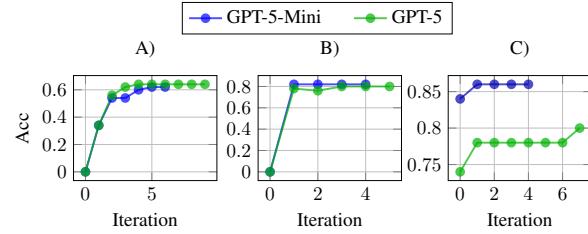
451 **TOOLOBSERVER performance over it- 452** Figure 3 shows that when minimal
 453 tool information is available (plot A),
 454 the model gradually improves across it-
 455 erations, measured by execution accuracy on
 456 BFCL-Opaque. When some useful infor-
 457 mation is already provided (plot B), per-
 458 formance spikes early on then plateaus,
 459 with little additional gain. Finally, in
 460 plot C, where most of the important tool
 461 is available, improvements are negligible
 462 and performance saturates almost immedi-
 463 ately. These patterns are consistent across
 464 all evaluated models. However, GPT-5
 465 takes more iterations to converge at the
 466 final tool documentation, while GPT-5-Mini
 467 consistently converges sooner.

468 We also report the average reflection it-
 469 erations required for convergence in Table 5.

470 We observe two key trends. First, convergence speed correlates with information availability: as
 471 starting documentation improves (from anonymized names to including parameters), average it-
 472 erations decrease by $\approx 35\%$ (from 3.2 to 2.2). Second, we observe that in the most opaque setting,
 473 GPT-5 uses more iterations (3.44) than GPT-5-mini (2.96). Combined with the performance gap in
 474 Figure 3, this suggests the stronger model engages in more thorough exploration to uncover com-
 475 plex behaviors. Even in the hardest setting, convergence occurs in under 3.5 iterations on average,
 476 demonstrating the experience-efficiency of TOOLOBSERVER.

477 **Fidelity of learned descriptions** We assess the quality of generated documentation both quan-
 478 titatively and qualitatively. First, we quantify the fidelity of learned documentation by measuring
 479 semantic (SBERT embedding Reimers & Gurevych (2019)) and lexical (ROUGE-1; Lin (2004))
 480 similarity against gold standards on BFCL-Opaque (Table 6). TOOLOBSERVER consistently outper-
 481 forms baselines on both metrics. In the hardest “Anon. Fn. Names” setting, we achieve a semantic
 482 similarity of **0.58** (vs. 0.51 for Play2Prompt) and a lexical overlap of **0.28** (vs. 0.18). The improve-
 483 ment in both metrics confirms that TOOLOBSERVER captures both the general semantic meaning
 and specific terminology and functional constraints from the gold documentation.

484 Second, we qualitatively analyze the descriptions generated for Chess tools in Appendix B. Interest-
 485 ingly, TOOLOBSERVER learns the nuances of *when* each tool must be used. For example, the middle
 game specialist final description explicitly mentions the tool is useful for stabilizing dynamic middle



451 Figure 3: Performance of TOOLOBSERVER on BFCL-
 452 Opaque with increased iterations on the tool settings:
 453 A): Anon. Function Names, B): Anon. Function
 454 Names + Descriptions, C): Anon. Function Names +
 455 Param. Names. Iterations stop after full convergence.
 456 These are expanded versions of the Table 2 results.

457 Table 5: Average number of reflection iters. required
 458 by TOOLOBSERVER to converge on BFCL-Opaque.

Configuration	GPT-5	GPT-5-mini
A) Anon. Fn. Names	3.44	2.96
B) Anon. Fn. Names + Desc.	2.66	2.60
C) Anon. Fn. Names + Param.	2.22	2.24

486
 487 Table 6: Average similarity metrics (semantic and lexical) of final documentation generated vs. gold
 488 documentation on BFCL-Opaque. The highest sem. is bolded, while the highest lex. is italicized.
 489 TO denotes TOOLOBSERVER, P2P denotes Play2Prompt, and ET denotes EasyTool.

Documentation	TO (GPT-5)		TO (GPT-5-Mini)		P2P		ET	
	Sem.	Lex.	Sem.	Lex.	Sem.	Lex.	Sem.	Lex.
Anon. Fn. Names	0.44	0.21	0.58	0.28	0.51	0.18	0	0
Anon. Fn. Names + Desc.	0.78	0.43	0.78	<i>0.44</i>	0.70	0.31	0.71	0.43
Anon. Fn. Names + Param. Names	0.71	<i>0.40</i>	0.71	<i>0.40</i>	0.69	0.28	0.69	0.39

496
 497 games. A similar pattern is observed in the depth specialist: the depth-2 specialist is described as
 498 effective at spotting immediate conversions, while the depth-16 specialist is ideal for volatile situations.
 499 On the other hand, Play2Prompt doesn’t perform well in the depth specialization setting –
 500 with a shocking performance of 0.25 – since the best tool’s description is under-specified. Instead,
 501 the agent always picks the *second* best tool which has a well-specified tool description.

502 **Strength of LLM Agent vs Editor LLM** We
 503 ablate the editor, using weaker models, i.e.
 504 GPT-5-Mini and O3, and measure the perfor-
 505 mance on BFCL in Table 7. The strength of the
 506 editor model directly affects the performance of
 507 both agents. Furthermore, the stronger agent
 508 model is generally more robust to a weak editor
 509 model. GPT-5-Mini sees a steep drop of 8 points when using GPT-mini as the editor. However, with
 510 GPT-5 as the LLM agent and GPT-5-Mini as the editor, the observed drop is a meager 2 points.

511 6 RELATED WORK

512 **LLM agents** LLM Agents that can call tools enable interaction with external systems. Schick
 513 et al. (2023) showed language models can learn to use tools through self-supervised learning, while
 514 ReAct (Yao et al., 2022) introduced the reasoning-action paradigm where agents alternate between
 515 thinking and acting. Shinn et al. (2023) further enhanced tool-using agents with the ability to learn
 516 from failures through verbal self-reflection. These advances have enabled complex agent behaviors,
 517 from multi-agent coordination (Park et al., 2023) to continuous skill acquisition (Wang et al., 2023a).

518 **Tool Documentation** Hsieh et al. (2023) demonstrate the critical role of comprehensive documentation
 519 in tool learning. Recent work has focused on automatically refining these descriptions: Qu et al.
 520 (2024) introduces the DRAFT framework where gathered experience is used to rewrite tool docu-
 521 mentation, while Fang et al. (2025) introduce Play2Prompt, which employs self-play followed by
 522 self-reflection to iteratively improve tool documentation. Yuan et al. (2024) propose EASYTOOL,
 523 showing that condensing verbose descriptions and adding usage examples significantly improves
 524 downstream performance by reducing hallucination rates. Wang et al. (2024) extend this line of
 525 work by incorporating short-term and long-term memory mechanisms after self-reflection phases.

526 **Tool Evaluation Benchmarks** Current tool-use benchmarks include BFCL (Berkeley Function
 527 Calling Leaderboard), which evaluates simple, single-turn function calls, ToolBench (Xu et al.,
 528 2023) and StableToolBench (Guo et al., 2024b), which offer thousands of real-world REST APIs.
 529 However, ToolBench and StableToolBench suffer from API key accessibility issues and poor repro-
 530 ducibility due to their reliance on external services.

531 7 CONCLUSION

532 We introduce OPAQUETOOLS BENCH, a benchmark consisting of three goal-oriented environments
 533 with opaque tools, which better reflects the reality of working with underspecified, poorly doc-
 534 umented tools in real-world scenarios. Through our proposed TOOLOBSERVER framework, we
 535 demonstrate that LLM agents can effectively learn to improve tool documentation through iterative
 536 observation of execution feedback, achieving superior performance while being significantly more
 537 token-efficient than existing approaches.

540 **LLM USAGE STATEMENT**
541542 We declare that we have used LLMs for polishing the content of this paper.
543544 **REFERENCES**
545

546 Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhari-
547 wal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agar-
548 wal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh,
549 Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler,
550 Ma teusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam Mc-
551 Candlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot
552 learners. *ArXiv*, abs/2005.14165, 2020. URL <https://api.semanticscholar.org/CorpusID:218971783>.

553 Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green,
554 Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifmoghaddam, Yanxi Li, Haoran Hong,
555 Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhui Chen, and Jimmy Lin.
556 Browsecmp-plus: A more fair and transparent evaluation benchmark of deep-research agent.
557 2025. URL <https://api.semanticscholar.org/CorpusID:280565737>.

558 Wei-Wen Fang, Yang Zhang, Kaizhi Qian, James Glass, and Yada Zhu. Play2prompt: Zero-shot
559 tool instruction optimization for llm agents via tool play. *ArXiv*, abs/2503.14432, 2025. URL
560 <https://api.semanticscholar.org/CorpusID:277104481>.

561 Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong
562 Sun, and Yang Liu. StableToolBench: Towards stable large-scale benchmarking on tool learning
563 of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of*
564 *the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156, Bangkok, Thailand,
565 August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.
566 664. URL <https://aclanthology.org/2024.findings-acl.664>.

567 Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong
568 Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning
569 of large language models, 2024b.

570 Cheng-Yu Hsieh, Sibei Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander J. Ratner, Chen-Yu Lee,
571 Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large
572 language models. *ArXiv*, abs/2308.00675, 2023. URL <https://api.semanticscholar.org/CorpusID:260351459>.

573 Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Annual Meeting of the*
574 *Association for Computational Linguistics*, 2004. URL <https://api.semanticscholar.org/CorpusID:964287>.

575 Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru,
576 Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard
577 Grave, Yann LeCun, and Thomas Scialom. Augmented language models: a survey. *Trans. Mach.
578 Learn. Res.*, 2023, 2023. URL <https://api.semanticscholar.org/CorpusID:256868474>.

579 OpenAI. Introducing gpt-5. OpenAI, August 2025. URL <https://openai.com/index/introducing-gpt-5/>.

580 Joon Sung Park, Joseph C. O’Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and
581 Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL
582 <https://arxiv.org/abs/2304.03442>.

583 Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language
584 model connected with massive apis. *ArXiv*, abs/2305.15334, 2023. URL <https://api.semanticscholar.org/CorpusID:258865184>.

594 Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and
 595 Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agen-
 596 tic evaluation of large language models. In *Forty-second International Conference on Machine*
 597 *Learning*, 2025.

598 Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru
 599 Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li,
 600 Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+
 601 real-world apis. *ArXiv*, abs/2307.16789, 2023. URL <https://api.semanticscholar.org/CorpusID:260334759>.

602 Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and
 603 Jirong Wen. From exploration to mastery: Enabling llms to master tools via self-driven in-
 604 teractions. *ArXiv*, abs/2410.08197, 2024. URL <https://api.semanticscholar.org/CorpusID:273233320>.

605 Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever.
 606 Language models are unsupervised multitask learners. 2019. URL <https://api.semanticscholar.org/CorpusID:160025533>.

607 Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-
 608 networks. *ArXiv*, abs/1908.10084, 2019. URL <https://api.semanticscholar.org/CorpusID:201646309>.

609 Tord Romstad, Marco Costalba, Joonas Kiiski, et al. Stockfish: Open source chess engine, 2024.
 610 URL <https://github.com/official-stockfish/Stockfish>.

611 Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer,
 612 Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to
 613 use tools. *ArXiv*, abs/2302.04761, 2023. URL <https://api.semanticscholar.org/CorpusID:256697342>.

614 Haiyang Shen, Yue Li, Desong Meng, Dongqi Cai, Sheng Qi, Li Zhang, Mengwei Xu, and
 615 Yun Ma. Shortcutsbench: A large-scale real-world benchmark for api-based agents. *ArXiv*,
 616 abs/2407.00132, 2024. URL <https://api.semanticscholar.org/CorpusID:270870543>.

617 Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion:
 618 Language agents with verbal reinforcement learning. *Advances in Neural Information Processing*
 619 *Systems*, 36:8634–8652, 2023.

620 Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. Llms in the imaginarium:
 621 Tool learning through simulated trial and error. In *Annual Meeting of the Association for Com-
 622 putational Linguistics*, 2024. URL <https://api.semanticscholar.org/CorpusID:268264353>.

623 Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan,
 624 and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models,
 625 2023a. URL <https://arxiv.org/abs/2305.16291>.

626 Lei Wang, Chengbang Ma, Xueyang Feng, Zeyu Zhang, Hao ran Yang, Jingsen Zhang, Zhi-Yang
 627 Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji rong Wen. A
 628 survey on large language model based autonomous agents. *ArXiv*, abs/2308.11432, 2023b. URL
 629 <https://api.semanticscholar.org/CorpusID:261064713>.

630 Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool
 631 manipulation capability of open-source large language models, 2023. URL <https://arxiv.org/abs/2305.16504>.

632 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
 633 React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629, 2022. URL
 634 <https://api.semanticscholar.org/CorpusID:252762395>.

648 Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng
649 Li, and Deqing Yang. Easytool: Enhancing llm-based agents with concise tool instruc-
650 tion. *ArXiv*, abs/2401.06201, 2024. URL <https://api.semanticscholar.org/CorpusID:266977201>.
651
652 Yinqi Zhang, Xintian Han, Haolong Li, Kedi Chen, and Shaohui Lin. Complete chess games enable
653 llm become a chess master. In *North American Chapter of the Association for Computational Lin-*
654 *guistics*, 2025. URL <https://api.semanticscholar.org/CorpusID:275954108>.
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701

702 **A EXPERIMENTAL DETAILS**
703704 We list further experimental details and results in this section.
705706 **A.1 TOOLOBSERVER HYPERPARAMETERS**
707708 For all datasets, we use the ReAct (Yao et al., 2022) framework, which prompts the agent to generate
709 interleaved reasoning traces and executable tool calls to solve tasks dynamically.
710711 **Max iterations:** For BFCL Opaque, we use a max iterations of 10 to iteratively test and optimize
712 tool descriptions. For BrowseComp and Chess, we use a max iterations of 3 due to computational
713 constraints (these trajectories are much larger and we have many more of them). Recall that for
714 BFCL, the LLM agent can stop early if it reaches a final answer; we report the average number of
715 iterations taken in Table 2.
716717 **Offline Batch Size** For the two tasks which we use the offline version of TOOLOBSERVER for -
718 Chess and BrowseComp Domains – we use a mini-batch size of 10 trajectories.
719720 **Chess Specifics:** To ensure computational efficiency during the *exploration phase*, we simulate
721 partial trajectories of 10 moves starting from the sampled position. We evaluate the state at the end
722 of this partial trajectory using the board value estimate from Stockfish (Romstad et al., 2024). This
723 scalar value acts as the dense process reward described in §4.1, allowing the editor language model
724 to critique the strategic quality of tool choices without needing to reach a checkmate.
725726 **Generation Details** Our experiments are primarily done with GPT-5 and GPT-5-mini ³; we sam-
727 ple from these models (OpenAI does not allow setting temperature or top-p parameters for these
728 models). To reduce latency during exploration on Chess which has long trajectories, we set rea-
729 soning effort to be minimal. Everywhere else, we use "medium" reasoning effort, including on the
730 documentation updates.
731732 Finally, in Figures 6 - 12 in Appendix E, we detail the prompts we use across our three datasets
733 for 1) tool usage/exploration 2) documentation updates and 3) the prompts for merging the learned
734 descriptions (if applicable).
735736 **A.2 OPAQUETOOLS BENCH DATASET DETAILS**
737738 **BrowseComp Domains** We run BrowseComp Domains using the Qwen-0.6B embedder model,
739 following Chen et al., 2025. Also following this work, we set the max number of tokens in retrieved
740 snippets to be 512, and the number of retrieved documents per search tool to be $k = 5$.
741742 **Chess** For the *final evaluation* on the test set (Table 3), we simulate full games. We enforce a maxi-
743 mum limit of **120 moves**; if the game exceeds this limit without a decisive result (checkmate/stale-
744 mate), it is recorded as a draw.
745746 **B COMPARISON OF FUNCTION DESCRIPTIONS FOR CHESS TOOLS**
747748 **B.1 GROUND TRUTH**
749750 The following are the ground truth descriptions generated for different chess tools.
751752 **Opening Specialist**
753754 A chess player that specializes in the opening phase. Plays strong moves (depth 16 anal-
755 ysis) during the opening, but resorts to random moves during middlegame and endgame
756 phases. Uses game phase detection to determine current phase.
757758 ³<https://platform.openai.com/docs/models/gpt-5> and <https://platform.openai.com/docs/models/gpt-5-mini>

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

Middle Game Specialist

A chess player that specializes in the middlegame phase. Plays strong moves (depth 16 analysis) during the middlegame, but resorts to random moves during opening and endgame phases. Uses game phase detection to determine current phase.

End Game Specialist

A chess player that specializes in the endgame phase. Plays strong moves (depth 16 analysis) during the endgame, but resorts to random moves during opening, middlegame, and late endgame phases. Uses game phase detection to determine current phase.

Late Game Specialist

A chess player that specializes in the late endgame phase. Plays strong moves (depth 16 analysis) during the late endgame, but resorts to random moves during opening, middlegame, and regular endgame phases. Uses game phase detection to determine current phase.

Depth-2 Specialist

Analyzes the current chess position using Stockfish engine at depth 2 (shallow analysis). Takes a FEN string as input and returns the best move in algebraic notation. This provides quick but less accurate move suggestions.

Depth-8 Specialist

Analyzes the current chess position using Stockfish engine at depth 8 (moderate analysis). Takes a FEN string as input and returns the best move in algebraic notation. This provides a good balance between speed and accuracy.

Depth-16 Specialist

Analyzes the current chess position using Stockfish engine at depth 16 (deep analysis). Takes a FEN string as input and returns the best move in algebraic notation. This provides highly accurate move suggestions but takes more time.

B.2 TOOLOBSERVER

The following are the descriptions generated by our method for different chess tools.

Opening Specialist

Best as a consolidation tool when already worse or in simplified positions, especially endgames. It favors quiet, low-variance moves that limit further damage. Less suitable for seizing the initiative or equalizing from balanced positions, and it can misjudge in sharp middlegames or make superficial material grabs in quiet endings. Use it to steady the ship, not to press for precise gains.

810

Middle Game Specialist

811

812

813

814

815

Best for stabilizing dynamic middlegames and defusing pressure through sound structural choices, particularly when defending. In calm or technical endgames and balanced positions, it tends toward passivity or inaccurate king/pawn decisions and is not a precise converter. Choose it to neutralize threats, not to fine-tune technical outcomes.

816

817

818

End Game Specialist

819

820

821

822

823

824

Best in simplified, technical positions that reward precision – especially rook endgames – and when an accurate, active move is needed to maintain control from an equal or slightly worse stance. Avoid it in sharp, tactical middlegames or when low-risk, resilient defense is required; it can overreach and swing the evaluation heavily. It's also not ideal for clinging on in lost endgames as White.

825

826

827

Late Game Specialist

828

829

830

831

Best for holding calm, technical endgames and maximizing resistance with safe, non-committal moves. Serviceable as a middle option when the position isn't tactically charged. Avoid it in sharp or tactical situations and whenever king safety is delicate; it tends to overlook immediate threats and can trigger large single-step collapses.

832

833

834

835

Depth-2 Specialist

836

837

838

839

Best when you want a decisive, forcing solution. Excels at spotting immediate conversions – tactical captures, central pawn breaks, direct mating or queening threats – and at jump-starting counterplay through rapid coordination (e.g., quick castling, active piece placement). Ideal for positions where concrete calculation can resolve tension right away. Tends to be high-variance: strong at seizing chances, but can overpress or misjudge tactical safety in messy defenses. Less suited to slow consolidation or risk-minimizing play.

840

841

842

843

844

845

Depth-8 Specialist

846

847

848

Best for building and sustaining initiative with forcing play. Excels at creating and maintaining pressure through checks, rook lifts, and energetic pawn breaks, and will alter the structure when it strengthens activity. Strong at converting an edge by keeping the opponent on the back foot. Less reliable when the position demands quiet consolidation or a concrete defensive neutralization; can overlook the need to stabilize before pressing.

849

850

851

852

853

Depth-16 Specialist

854

855

856

857

Best for stabilization and neutralization. Excels at consolidating king safety, coordinating pieces, recapturing accurately, and simplifying when under pressure. Finds direct defensive resources – exchanges and file contests – that reduce the opponent's practical chances. Ideal in worse or volatile positions and whenever risk control is paramount. Tends to forgo sharp attacking continuations in favor of solid, resilient play.

858

859

860

861

862

863

B.3 PLAY2PROMPT

The following are the descriptions generated by Play2Prompt for different chess tools.

864
865**Opening Specialist**866
867

function_1 analyzes a chess position from a FEN string and returns a single best-move suggestion in SAN (Standard Algebraic Notation).

868

Parameters

869

- `board_state` (string, required): Complete 6-field FEN: <pieces> <active> <castling> <en-passant> <halfmove> <fullmove>. Provide a legal position and include all fields, separated by single spaces. Example: `rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1`

873

Output

874

- A SAN move string, e.g., “e4”, “Nf3”, “exd5”, “O-O”, “O-O-O”, “Bb5+”, “Qh7#”.

876

Usage example

877

- Call name: `function_1` (do not use the original name).

879

Input JSON:

880

```
{ "board_state": "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR
w KQkq - 0 1" }
```

883

- Example result: “e4”

884

Notes

885

886

- Use the exact key `board_state`.
- Output is SAN only (not UCI/LAN).

888

889

Middle Game Specialist

890

891

Analyzes a standard chess position and returns a single best move suggestion.

892

Input:

893

A complete 6-field FEN string.

894

Parameter:

895

- `board_state` (string, required) — Valid FEN with:

896

1. piece placement (8 ranks separated by “/”, digits for empty squares, pieces PN-BRQK/pnbrqk),
2. active color “w” or “b”,
3. castling rights as a subset of “KQkq” or “-”,
4. en-passant target square “-” or a3–h6,
5. halfmove clock (non-negative integer),
6. fullmove number (≥ 1).

897

Output:

898

One move in Standard Algebraic Notation (SAN), e.g., “e4”, “Nf3”, “exd5”, “Qh5+”, “e8=Q#”, including disambiguation as needed. No extra text.

899

Scope:

900

Standard chess only (not Chess960).

901

Validation:

902

Malformed/illegal FEN may be rejected — ensure correct field count and values.

903

Examples:

904

- `board_state: "rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR
b KQkq - 0 1" → "d5"`

905

- `board_state: "r1bqkbnr/pppppppp/2n5/8/8/2N5/PPPPPPP/R1BQKBNR
w KQkq - 0 3" → "Nf3"`

916

917

918
919**End Game Specialist**920
921
922
923
924

Analyzes a chess position and returns a single move suggestion in Standard Algebraic Notation (SAN). Alias: `endgame_specialist`. Input must be a complete, valid FEN string (six space-separated fields): piece placement, side to move (w/b), castling rights, en passant target square, halfmove clock, fullmove number. The side to move is taken from the FEN.

925
926
927
928**Parameters:**

- `board_state` (string, required): Full FEN for the current position. Include correct castling rights and en passant target if applicable.

929

Output:930
931
932
933

- One SAN move string (not UCI/coordinate), e.g., “Nf3”, “exd5”, “O-O”, “O-O-O”, “a8=Q”, “Qh7+”, “Rxf8#”, with standard disambiguation as needed. Captures use “x”; promotions use “=Q/R/B/N”; checks “+”; checkmates “#”.

934

Example:935
936
937

- `board_state: "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1" → "e4"`

938
939
940
941
942
943
944
945
946947
948**Late Game Specialist**949
950
951

Analyzes a chess position and returns a single move suggestion in Standard Algebraic Notation (SAN) as a plain string.

952
953
954
955
956**Parameters**

- `board_state` (string, required): Full 6-field FEN of the current position. Format: “[piece placement] [side to move] [castling rights] [en passant target] [halfmove clock] [fullmove number]”. Example: “rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1”. Must represent a legal position.

957

Output958
959
960
961

- SAN move string only (no JSON object). Examples: “e4”, “Nf3”, “exd5”, “O-O”, “O-O-O”, “e8=Q”, “Qh7#”, “Rd1+”.

962

Notes963
964
965
966
967
968
969
970
971

- Input must be FEN (not PGN or UCI).

- SAN uses uppercase piece letters (pawn omitted), “x” for captures, “+/#” for check/mate, and “=Q/R/B/N” for promotions.

Example call

- `function_4 ({ "board_state": "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1" })`

Example response

- “e4”

972
973**Depth-2 Specialist**974
975
976

Suggests a single chess move for the side to move in the given position using a shallow fixed-depth analysis (~2 plies). Always call this tool when a move is requested from a FEN; do not infer moves without it.

977

Parameters:978
979
980
981
982

- `board_state` (string, required): A valid full FEN for standard chess with all 6 fields: piece placement, active color (w/b), castling rights (KQkq or -), en passant target (square or -), halfmove clock, fullmove number. The FEN must be legal and consistent; castling and en passant fields affect legality.

983

Output:984
985
986

A single move in Standard Algebraic Notation (SAN), not UCI/LAN. Examples: “e4”, “Nf3”, “O-O”, “R1e2”, “exd5”, “e8=Q#”. Includes “+” or “#” if applicable. No extra text.

987

Notes:
If multiple moves are near-equal, one is returned.

988

Example:989
990
991

```
board_state="rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w
KQkq - 0 1" → "e4".
```

992

993

994

995

996

997

998

999

1000

1001

1002

Depth-8 Specialist

1003

1004

Analyzes a standard chess position and returns the single best move at fixed search depth 8 plies. Use exactly one required parameter.

1005

1006

Required parameter:1007
1008

- `board_state` (string): A single-line valid FEN (Forsyth–Edwards Notation). Must include exactly 6 space-separated fields:

1009

1010

1011

1012

1013

1014

1015

1016

1017

1. piece placement,

2. side to move (w/b),

3. castling rights (KQkq or -),

4. en passant target square (e.g., e3 or -),

5. halfmove clock (integer),

6. fullmove number (integer).

No extra whitespace or newlines; standard chess only; position should be legal.

Returns:1018
1019
1020

One move in SAN (Standard Algebraic Notation), e.g., “Nf3”, “exd5”, “O-O”, “O-O-O”, “e8=Q”, “Rxd8+”, “Qh7#” (with disambiguation if needed). Not UCI; no scores or move lists.

1021

Example call:1022
1023

```
{"board_state": "rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR
b KQkq - 0 1"}
```

1024

Example output format:

1025

“Nf6”

1026
1027**Depth-16 Specialist**1028
1029

Analyzes a standard chess position and returns the engine’s best move as a single SAN string.

Parameter:1030
1031
1032
1033
1034

- `board_state` (string, required) — a Forsyth–Edwards Notation (FEN) of a legal position; include side-to-move and other fields (full 6-field FEN preferred). The move is determined from the FEN’s side-to-move.

1035
1036
1037**Output:**

Exactly one move in Standard Algebraic Notation (SAN), e.g., “e4”, “Nf3”, “O-O”, “exd5”, “Qh8#”, “b8=Q+”, “R1e2”. Do not expect UCI/LAN or any extra text.

1038

Notes:

Fixed internal search depth (cannot be configured); standard chess only (no variants). Use the exact key name `board_state` and pass only the FEN string.

1039

Example call:1040
1041
1042
1043

```
{"board_state": "rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR  
b KQkq - 0 1"}
```

1044
1045**Example response:**

“d5”

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

B.4 EASYTOOL

1057

1058

The following is the description generated by EasyTool for chess tools; note that they are almost all identical since the initial description is the same (differences are only up to sample differences).

1059

1060

1061

1062

1063

1064

1065

1066

All tools

1067

1068

Tool Description: ‘function_1’ analyzes a chess position from a FEN ‘board_state’ and returns a suggested move in algebraic notation.

1069

Tool Guidelines:

1070

Scenario:

1071

1072

if you want to analyze a chess position (e.g., the Ruy Lopez after 4...Nf6) and get a suggested move for White.

1073

Parameters:

1074

1075

```
{"board_state": "r1bqkb1r/1ppp1ppp/p1n2n2/4p3/B3P3/5N2/PPPP1PPP/RNBQK2R  
w KQkq - 2 5"}
```

1076

1077

1078

1079

1080 **C TASKS**
 1081

1082 **C.1 BFCL-OPAQUE: DISCOVERING TOOL FUNCTIONALITY FROM OPAQUE DESCRIPTIONS**
 1083

1084 The Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2025) We systematically degrade
 1085 BFCL’s tool descriptions to create an opaque setting where models must infer functionality through
 1086 interaction. This tests the fundamental ability to map between ambiguous tool interfaces and their
 1087 underlying behaviors.

1088 **Task Setup:** Models receive user queries requiring specific tool calls (e.g., “What’s the weather
 1089 in San Francisco?”, “Schedule a meeting for tomorrow at 3pm”) but must discover which tools
 1090 accomplish each task. We provide tools with systematically degraded documentation: function
 1091 names are replaced with generic identifiers (e.g., `tool_1`, `tool_2`), descriptions are removed or
 1092 made ambiguous, and parameter specifications lack type information or semantic hints. Models
 1093 must experiment with different tools and parameter combinations to discover correct usage patterns.
 1094

1095 **Tool Degradation Strategy:** Initially, the complete tool specifications include function descriptions,
 1096 parameter names, and parameter descriptions. We replace semantic function names with
 1097 generic identifiers (`function_1`, `function_2`, etc.), removing the primary semantic cue for tool
 1098 selection. We then create three different documentation levels:
 1099

1. **Anon. function name only**, where we remove everything (function description, parameter
 1100 names, and parameter descriptions), testing pure behavioral discovery through trial and
 1101 error with **no** documentation
2. **Anon. function name + Description**, where we remove all parameter names/descriptions
 1103 while keeping only the function description, testing whether models can infer argument
 1104 structure from behavioral descriptions alone.
3. **Anon. function name + Parameter names**, where we remove the function description
 1106 and parameter descriptions while keeping only parameter names, testing discovery of func-
 1107 tionality from argument structure without semantic guidance.

1110 **Data Collection:** We evaluate on BFCL’s executable subset, which provides deterministic, pro-
 1111 grammatically verifiable tasks across four categories: executable simple, executable multiple func-
 1112 tion, executable parallel function, and executable parallel multiple function. This subset ensures
 1113 reproducible evaluation—each task has ground-truth tool calls and expected outputs.

1114 **Primary Evaluation Metrics:** We measure binary task completion accuracy—whether the model
 1115 successfully calls the correct tool with proper arguments to satisfy the user query.

1118 **Enhanced Evaluation Metrics** Standard BFCL evaluation relies on a binary success metric
 1119 (Pass/Fail). To better diagnose *how* agents learn opaque tool behaviors, we introduce two granu-
 1120 lar metrics that distinguish between semantic understanding (selecting the right tool) and syntactic
 1121 mastery (calling it correctly).

1122 **1. Parameter Accuracy.** This metric measures the exact correctness of the arguments provided,
 1123 conditional on the agent selecting the correct tool. If the model chooses the wrong function, the
 1124 score is 0. When the correct function is chosen, we calculate the percentage of expected parameters
 1125 that are perfectly recovered. Specifically, it is the ratio of arguments where both the parameter
 1126 name and the assigned value exactly match the ground truth, divided by the total number of required
 1127 parameters. This metric strictly penalizes missing arguments or incorrect values, distinguishing
 1128 agents that “know” the tool from those that merely guess the function name.

1129 **2. AST (Abstract Syntax Tree) Accuracy.** AST Accuracy evaluates the structural validity and
 1130 “grammar” of the tool call, independent of whether the values are correct. It is calculated as the
 1131 average of five components:

- 1132 • **Format Validity:** Whether the output is parsable as valid JSON or a Python Abstract
 1133 Syntax Tree (using `ast.parse`).

- **Structure Validity:** Whether the parsed object contains the standard `function` and `args` keys.
- **Type Correctness:** The percentage of parameters where the data type (e.g., string, integer, list) matches the ground truth schema.
- **Schema Compliance:** A strict check ensuring the structure is valid, all types are correct, and no hallucinated parameters exist.
- **Hallucination Check:** Whether the agent generated parameters that do not exist in the tool definition.

C.2 CHESS: LEARNING STRATEGIC TOOL SELECTION THROUGH EXPERIENCE

We challenge LLMs to play Chess, but instead of predicting moves directly, models are given access to several undocumented tools that accept current board positions in FEN notation and return move recommendations. Each next move suggestion function has an identical interface, but undocumented behavioral differences. Thus, the agent must discover through gameplay that each implements different strategies. Performance directly reflects the model’s ability to document each tool’s strengths.

Task Setup: Models play chess games against a fixed-strength opponent (Stockfish at depth 2) by selecting from available tool sets. Each trajectory consists of a max number of moves where the model must select a tool and play against the fixed opponent.

Tool Sets: We construct two tool sets of increasing complexity:

1. **Phase specialization** (4 tools) - These tools are engines that work well for specific phases: opening, middlegame, endgame and late endgame. These phases are defined by number of pieces on the board: opening phase has at least 28 pieces, middlegame at least 16, endgame at least 10 and late endgame has less than 10 pieces. Each tool plays moves according to a strong engine (depth 16 analysis) in its own phase but plays randomly otherwise. An optimal agent would learn to document these temporal patterns
2. **Depth gradients** (3 tools) Tools 1, 2, and 3 are Stockfish with search depth 2, 4, and 8 (higher=better); this tests fine-grained discrimination between similar high-quality tools.

Data Collection: We sample 2000 chess positions from the Lichess database⁴, which provides hundreds of millions of positions with chess engine evaluations. We split these 2000 positions into training (10%) and test (90%) sets, maintaining the same stratified distribution across both game phases and position evaluations to ensure comparable evaluation conditions:

- **Game phase** (determined by piece count): opening (25%), middlegame (40%), endgame (25%), late endgame (10%)
- **Position evaluation** (from Lichess engine analysis): equal positions (40%), slight advantages for white/black (10% each), winning positions for white/black (8% each), and crushing/mate positions for each side (6% each)

This stratified sampling ensures models encounter diverse board states that test tool selection across different game scenarios.

Main Evaluation Metrics: Since we know the optimal tool call at every turn (the correct phase specialized tool for phase specialization or the highest search depth tool for depth gradients), we simply calculate the accuracy of LLM agent tool calls as our evaluation metric.

C.3 ADDITIONAL CHESS EVALUATION METRIC: STREAMING ELO

To provide a fine-grained measurement of strategic decision quality beyond binary tool-choice accuracy, we implement a **Streaming Elo** rating system. The Elo rating system is a method for calculating the relative skill levels of players in zero-sum games.

⁴<https://database.lichess.org>

1188
 1189 **Opponent Pool.** We evaluate the agents against a diverse set of deterministic opponents using the
 1190 Stockfish engine at varying difficulty levels to represent different tiers of play:

1191 • **Beginner:** Stockfish Level 1 (Approx. Elo 800)
 1192 • **Intermediate:** Stockfish Level 5 (Approx. Elo 1600)
 1193 • **Master:** Stockfish Level 10 (Approx. Elo 2400)

1195
 1196 **Update Rule.** The agent starts with a standard baseline rating of $R_0 = 1200$. After each game i ,
 1197 the rating is updated based on the result against an opponent with rating R_{opp} . We use a K-factor of
 1198 $K = 32$. The expected score E_i and the updated rating R_{i+1} are calculated as follows:

1199
 1200
$$E_i = \frac{1}{1 + 10^{(R_{opp} - R_i)/400}} \quad (3)$$

1201
 1202
$$R_{i+1} = R_i + K \cdot (S_{actual} - E_i) \quad (4)$$

1203 where S_{actual} is the game outcome (1.0 for a win, 0.5 for a draw, 0.0 for a loss).

1204
 1205 **Experimental constraints and Bootstrapping.** Due to the high computational cost of running
 1206 full tool-augmented chess trajectories, we evaluate on a subset of 300 games played against the
 1207 opponent pool. To prevent infinite loops in drawn or lost positions, any game exceeding **120 moves**
 1208 is automatically adjudicated as a draw.

1209
 1210 Streaming Elo ratings can be sensitive to the specific chronological order of matches (e.g., facing a
 1211 string of Master-level opponents early can depress the rating, making recovery difficult). To elim-
 1212 inate this variance and ensure a robust final metric, we employ **bootstrapping**. We shuffle the se-
 1213 quence of the 300 completed games into **1,000 random permutations**, calculate the final streaming
 1214 Elo for each permutation, and report the mean rating across all permutations.

1215
 1216 **C.4 BROWSECOMP DOMAINS: LEARNING MULTI-TOOL COORDINATION FOR COMPLEX**
 1217 **INFORMATION SEEKING**

1218
 1219 Complex question-answering requires discovering not just individual tool capabilities, but how to
 1220 coordinate multiple tools strategically. BrowseComp Plus (Chen et al., 2025) provides an ideal
 1221 testbed for this challenge—human-curated questions that demand synthesizing information from
 1222 dozens of search queries. Unlike simple retrieval tasks, these questions require models to discover
 1223 through interaction which tools access which information sources, how to formulate effective queries
 1224 for each, and how to combine results to build comprehensive answers.

1225
 1226 **Task Setup:** Models must answer complex, multi-hop questions using search tools with opaque
 1227 documentation. Each question requires aggregating information from multiple sources—for exam-
 1228 ple, comparing statistics across countries, tracing historical developments, or synthesizing technical
 1229 specifications. While BrowseComp Plus provides a fixed corpus containing all necessary docu-
 1230 ments, models receive no documentation about which tools search which subsets or how query
 1231 syntax varies between tools. They must discover these constraints through experimentation during
 1232 actual question-answering trajectories.

1233
 1234 **Tool Sets and Degradation Strategy:** We construct two search environments that test different
 1235 aspects of tool discovery: (1) **Domain-specific search** (9 tools) where specialized tools each query
 1236 distinct document subsets (academic papers, product catalogs, geographical data, news articles),
 1237 testing discovery of tool coverage boundaries and domain-specific query patterns ; and (2) **Mixed**
 1238 **search** (10 tools) which combines specialized domain tools with a general tool that searches the
 1239 entire corpus, testing strategic selection between targeted and broad search approaches. We introduce
 1240 realistic opacity patterns that mirror production search systems—tools are provided with generic
 1241 names (`search_1`, `search_2`) and minimal documentation. Models must discover through inter-
 1242 action: coverage boundaries (which document types each tool can access), query constraints (max-
 1243 imum query length, required syntax, boolean operator support), and ranking behaviors (how each

1242 tool prioritizes results by recency, relevance, or popularity). These undocumented behaviors only
 1243 emerge through varied usage patterns across multiple queries.
 1244

1245 **Data Collection:** We use BrowseComp Plus’s curated question set, which includes 830 complex
 1246 questions designed to require extensive information gathering. Questions span diverse domains
 1247 including science, history, geography, and current events. Each question has human-validated an-
 1248 swers and requires on average 15-30 search queries when using well-documented tools, making this
 1249 an ideal benchmark for measuring if models can learn tool capabilities while solving real tasks.
 1250

1251 **Evaluation Metrics:** We measure both answer accuracy (F1 score against gold answers) and
 1252 search efficiency (number of queries required). Unlike single-shot benchmarks, we track improve-
 1253 ment across questions—does the model become more efficient at using discovered tool capabilities?
 1254 We also measure cross-question transfer: when models discover a tool searches academic papers
 1255 while answering a science question, can they apply this knowledge to a history question requiring
 1256 scholarly sources?
 1257

1258 D BASELINES

1259 Following TOOLOBSERVER, for all baselines we use GPT-5.
 1260

1262 D.1 PLAY2PROMPT

1263 **Play2Prompt** (Fang et al., 2025) improves tool-documentation from self-play followed by self-
 1264 reflection. It iteratively generates a set of tool usage examples by “playing” with the tool, using the
 1265 responses until it generates valid example tool usages. Using these examples, the documentation is
 1266 iteratively improved based on the tool use errors observed while using the current documentation.
 1267

1268 D.2 EASYTOOL

1269 (Yuan et al., 2024) which automatically rewrite the tool documentation in two stages. First, it con-
 1270 denses the tool descriptions to eliminate redundant information and focuses only on core functional-
 1271 ility. Then, it creates structured functional guidelines with usage scenarios and parameter examples to
 1272 help LLMs understand when and how to use each tool. EasyTool is limited by its lack of execution
 1273 of the tools themselves. Furthermore, the descriptions and functional guidelines are beforehand,
 1274 hence cannot benefit from any knowledge gained as the trajectory rolls out.
 1275

1277 E TOOLOBSERVER PROMPTS

1278 You are an expert in composing and exploring functions. You are given a
 1279 user question and a set of available tools.
 1280
 1281 You must call at least one tool in response to every user question.
 1282 There are no exceptions. Refusing to call a tool is not allowed.
 1283
 1284 If you are confident in a tool’s purpose, use it appropriately to
 1285 address the user’s request. If you are unsure what a tool does, make a
 1286 best guess and try it with plausible parameters to learn how it behaves.
 1287 It is better to experiment than to fail to respond.
 1288
 1289 Always format tool calls correctly with all required parameters. You
 1290 should only return function calls in the tool call sections.
 1291

1292 Figure 4: BFCL Exploration Prompt
 1293
 1294
 1295

```

1296
1297 "pre": """You are improving function documentation by analyzing real
1298 usage examples. Your goal is to write descriptions that help future
1299 users understand exactly what each function does and how to use it
1300 correctly.
1301 Current Function Definitions:
1302 {available_functions}
1303
1304 Observed usage examples:
1305 """
1306 -----
1307 "middle": """
1308 Example {example_num}:
1309 User Question: {question}
1310 Function Called: {function_call}
1311 Function Output: {function_output}
1312 """
1313 -----
1314 "post": """
1315 ## Important
1316
1317 **If the existing description is already accurate, do not include it in
1318 your response.** Only provide updates for functions that actually need
1319 improvement. You can tell a description is accurate if in the usage
1320 examples the function was called successfully and produced the expected
1321 output.
1322
1323 ## Instructions
1324
1325 **Only analyze and update functions that were actually called in the
1326 examples above.** Do not provide descriptions for functions that weren't
1327 used.
1328
1329 For each function that was called:
1330 1. Examine what inputs were provided and what the function returned
1331 2. Compare this with the existing description
1332 3. **Only provide updated descriptions for functions that need
1333 improvement**
1334
1335 ## Requirements for Updated Descriptions
1336
1337 Each improved description must clearly explain:
1338 - **What the function does** (its purpose and behavior)
1339 - **What inputs it expects** (parameter types, format, requirements)
1340 - **What it returns** (output format and content)
1341 - **How to call the function** (usage pattern or syntax)
1342
1343 Write 1-2 clear, comprehensive sentences that give users everything they
1344 need to use the function correctly.
1345
1346 ## Format
1347
1348 FUNCTION: [function_name]
1349 DESCRIPTION: [Your improved description here]
1350
1351 ## Updated Descriptions (only for functions that need improvement):"""

```

1346
1347 Figure 5: The BFCL reflection prompt. It is split into three parts. A “pre-prompt”, a “middle-
1348 prompt”, and a “post-prompt”. We concatenate them together along with the real usage behaviors
1349 (ie, the function calls and the outputs).

```

1350
1351
1352
1353
1354
1355 You are a chess-playing AI assistant with access to chess analysis tools
1356 for evaluating positions and suggesting moves.
1357
1358 <objective>
1359 Play the strongest move in the current position by making exactly ONE
1360 tool call to analyze the position.
1361 </objective>
1362
1363 <tool_selection_strategy>
1364 - Make exactly ONE tool call per move decision
1365 - Choose the strongest tool available for the current position
1366 - If tools appear similar in capability or are similarly unknown,
1367 consider trying a different tool than recently used ones to diversify
1368 your information sources
1369 - When one tool is clearly superior for the position type, use it
1370 regardless of history
1371 </tool_selection_strategy>
1372
1373 <exploration_principle>
1374 - Primary goal: Select the strongest tool for each position
1375 - Secondary consideration: If multiple tools seem equally strong or
1376 equally unknown, vary your selection based on recent usage history
1377 - This diversification helps avoid potential blind spots from relying on
1378 a single tool's perspective
1379 - Never sacrifice move quality for exploration - only explore when tools
1380 are genuinely comparable
1381 </exploration_principle>
1382
1383 <decision_framework>
1384 With your single tool call, consider:
1385 - What type of position is this? (tactical, positional, endgame, opening
1386 )
1387 - Which tool is strongest for this specific position?
1388 - If multiple tools seem equally strong, which have I used recently?
1389 - Is there a clear best tool, or are several tools comparably suitable?
1390 </decision_framework>
1391
1392 <tool_preamble>
1393 Before making your tool call:
1394 - Explain which tool you're selecting and why it's the strongest choice
1395 for this position
1396 - If multiple tools seemed equally viable, briefly note why you selected
1397 this one over the others
1398 </tool_preamble>
1399
1400 <quality_checks>
1401 - Select the strongest available tool (or make a reasonable choice among
1402 equals)
1403 - Make exactly one tool call
1404 </quality_checks>
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
35
```

```

1404
1405
1406 Analyze chess tool performance across N game trajectories to generate
1407 improved tool descriptions that clearly differentiate when to use each
1408 tool.
1409
1410 <input>
1411 - Game trajectories with tool calls, moves, and positions
1412 - Board evaluations (positive=White advantage, negative=Black advantage)
1413 - Current tool descriptions
1414 - Side played by agent in each game
1415 </input>
1416
1417 <analysis_requirements>
1418 For each tool:
1419 - Identify consistent patterns in its behavior and performance
1420 - Determine what distinguishes it from other tools
1421 - Provide concrete proof: cite specific trajectories and moves showing
1422 these patterns
1423 - Focus on situations where this tool performs differently than others
1424
1425 Evaluation notes:
1426 - Higher eval is better for White, lower eval is better for Black
1427 - IMPORTANT: Always compare tools relatively, not absolutely
1428 - Example for White: Tool A suggesting move to +2 is better than Tool B
1429 suggesting +1
1430 - Example for Black: Tool A suggesting move to -3 is better than Tool B
1431 suggesting -1
1432 - Critical: Even in losing positions, compare which tool finds the best
1433 continuation
1434     * For White: -5 is much better than -10 (both losing, but one is more
1435 resilient)
1436     * For Black: +10 is much better than +15 (both losing, but one offers
1437 more resistance)
1438 - Don't dismiss a tool just because it suggested moves in bad positions
1439 - focus on whether it found the BEST move among the alternatives
1440 </analysis_requirements>
1441
1442 <output_per_tool>
1443 **Tool: [name]**
1444
1445 Observed patterns: [Key behaviors identified with specific trajectory
1446 evidence]
1447
1448 Distinguishing characteristics: [What makes this tool different from
1449 others, with examples]
1450
1451 Updated description:
1452 [Concise description stating when to use this tool relative to others]
1453
1454 Reasoning: [Justification based on trajectory evidence]
1455 </output_per_tool>
1456
1457 <final_output>
1458 After analyzing all tools, provide a decision framework for selecting
1459 between tools based on the patterns discovered.
1460 </final_output>
1461
1462 Key: Every claim must reference trajectories. Descriptions must be
1463 comparative (tool X better than Y for Z) not absolute.
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
```

```

1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468 You will receive N LLM responses, each analyzing different batches of
1469 chess game trajectories. Synthesize these into definitive tool
1470 descriptions.

1471 <synthesis_task>
1472 For each tool:
1473 1. Identify patterns that appear across multiple responses
1474 2. Note contradictions between responses
1475 3. Distinguish true patterns from batch-specific noise
1476 4. Look for emergent patterns that no single analysis identified but
1477 become visible when viewing all analyses together
1478 5. Create ONE final description based on the most reliable patterns

1479 Critical:
1480 - A behavior mentioned in only 1-2 responses is likely batch-specific
1481 noise
1482 - Focus on patterns that multiple independent analyses discovered
1483 - Also identify meta-patterns: behaviors that emerge from the collective
1484 evidence but weren't explicitly stated in any single response
1485 - When responses conflict, examine their evidence strength
1486 - Final descriptions should capture the tool's strengths/weaknesses but
1487 NOT explicitly name other tools
1488 </synthesis_task>

1489 <output_format>
1490 **Tool: [name]**
1491
1492 Synthesis reasoning:
1493 [Explain which patterns were most consistent across analyses, what
1494 emergent patterns were discovered, how conflicts were resolved, and why
1495 certain behaviors were included/excluded in the final description.]
1496
1497 Final description:
1498 [Single definitive description of when to use this tool. Describe its
1499 characteristics and optimal use cases WITHOUT referencing other tools by
1500 name. Example: "Best for tactical positions requiring deep calculation.
1501 Excels at finding forcing sequences and material sacrifices. Tends to
1502 be overly aggressive in quiet positions."]
1503 </output_format>

```

1501 Figure 8: The Chess consensus merge reflection prompt. The chess descriptions generated from the
1502 previous step are appended to this prompt.
1503

```

1504
1505
1506
1507
1508
1509
1510
1511

```

1512 You are a question-answering AI assistant with access to search tools
1513 that return different types of results.
1514
1515 <objective>
1516 Find the correct answer to the question by making strategic tool calls
1517 over multiple turns. Each turn, you make exactly ONE tool call and
1518 receive its results before deciding your next action.
1519 </objective>
1520
1521 <how_this_works>
1522 - You will be called multiple times for the same question
1523 - Each time, you'll see the full history of your previous tool calls and
1524 their results
1525 - Each turn, make exactly ONE tool call to gather more information
1526 - Use what you've learned from previous tool calls to inform your next
1527 choice
1528 - Once you have enough information, provide your final answer
1529 </how_this_works>
1530
1531 <tool_selection_strategy>
1532 - Make exactly ONE tool call per turn
1533 - Choose the strongest tool available for the current information needs
1534 - Review what you've already learned from previous tool calls
1535 - If tools appear similar in capability or you're uncertain about what
they return, consider trying a different tool than recently used ones to
1536 diversify your information sources
1537 - When one tool is clearly superior for the remaining information needs,
use it regardless of history
1538 </tool_selection_strategy>
1539
1540 <exploration_principle>
1541 - Primary goal: Select the strongest tool for your current information
gap
1542 - Secondary consideration: If multiple tools seem equally strong or you'
re uncertain about their outputs, vary your selection based on what you'
ve already tried
1543 - This diversification helps avoid potential blind spots from relying on
1544 a single tool's perspective
1545 - Never sacrifice answer quality for exploration - only explore when
tools are genuinely comparable or unknown
1546 - Learn from previous tool results: if a tool gave poor results before,
1547 consider alternatives
1548 </exploration_principle>
1549
1550 <decision_framework>
1551 Each turn, consider:
1552 - What information do I still need to answer this question?
1553 - What have I learned from previous tool calls?
1554 - What type of question is this? (factual, current events, historical,
1555 technical, domain-specific)
1556 - Which tool is strongest for filling my current information gap?
1557 - If multiple tools seem equally strong or I'm uncertain about them,
1558 which have I used already?
1559 - Do I have enough information to answer, or should I make another tool
call?
1560 </decision_framework>

Figure 9: BrowseComp Domains exploration prompt, part 1. We append the context, including the recent tool-calls and results, to this prompt before receiving the next tool-call.

```

1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584 <tool_preamble>
1585 Before making your tool call:
1586 - Review what you've learned from previous tool calls (if any)
1587 - Explain which tool you're selecting and why it's the strongest choice
1588 for your current information needs
1589 - If you're uncertain about what a tool returns, acknowledge this
1590 uncertainty
1591 - If multiple tools seemed equally viable or unknown, briefly note why
1592 you selected this one over the others
1593 </tool_preamble>
1594 <quality_checks>
1595 - Review previous tool calls and their results
1596 - Select the strongest available tool for your current needs (or make a
1597 reasonable choice among equals/unknowns)
1598 - Make exactly one tool call per turn
1599 - Use accumulated tool results across turns to formulate your answer
</quality_checks>

```

Figure 10: BrowseComp Domains exploration prompt, part 2. We append the context, including the recent tool-calls and results, to this prompt before receiving the next tool-call.

```

1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619

```

```

1620
1621
1622
1623
1624 Analyze search tool performance across N question-answering trajectories
1625 to generate improved tool descriptions that clearly differentiate when
1626 to use each tool.
1627
1628 <input>
1629 - Question-answering trajectories with tool calls and results
1630 - Search results returned by each tool (content may vary by tool)
1631 - Whether the final answer was correct or incorrect
1632 </input>
1633
1634 <analysis_requirements>
1635 For each tool:
1636 - Identify consistent patterns in the type and quality of results it
1637 returns
1638 - Determine what distinguishes it from other tools
1639 - Provide concrete proof: cite specific trajectories and queries showing
1640 these patterns
1641 - Focus on situations where this tool performs differently than others
1642
1643 Evaluation notes:
1644 - IMPORTANT: Compare tools relatively, not absolutely
1645 - A tool is effective if it helps the agent reach the correct answer
1646 - Consider both successful and unsuccessful trajectories
1647 - Focus on: result relevance, information completeness, and query type
1648 suitability
1649 - Don't dismiss a tool just because it was used in failed trajectories -
1650 focus on whether it provided useful information compared to
1651 alternatives
1652 </analysis_requirements>
1653
1654 <output_per_tool>
1655 **Tool: [name]**
1656
1657 Observed patterns: [Key behaviors identified with specific trajectory
1658 evidence]
1659
1660 Distinguishing characteristics: [What makes this tool different from
1661 others, with examples]
1662
1663 Updated description:
1664 [Concise description stating when to use this tool relative to others]
1665
1666 Reasoning: [Justification based on trajectory evidence]
1667 </output_per_tool>
1668
1669 <final_output>
1670 After analyzing all tools, provide a decision framework for selecting
1671 between tools based on the patterns discovered.
1672 </final_output>
1673
1674 Key: Every claim must reference trajectories. Descriptions must be
1675 comparative (tool X better than Y for Z) not absolute.

```

Figure 11: The BrowseComp domains batch analysis reflection prompt. The trajectories are appended to this prompt.

```

1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684 You will receive N LLM responses, each analyzing different batches of
1685 question-answering trajectories. Synthesize these into definitive tool
1686 descriptions.
1687
1688 <synthesis_task>
1689 For each tool:
1690 1. Identify patterns that appear across multiple responses
1691 2. Note contradictions between responses
1692 3. Distinguish true patterns from batch-specific noise
1693 4. Look for emergent patterns that no single analysis identified but
1694 become visible when viewing all analyses together
1695 5. Create ONE final description based on the most reliable patterns
1696
1697 Critical:
1698 - A behavior mentioned in only 1-2 responses is likely batch-specific
1699 noise
1700 - Focus on patterns that multiple independent analyses discovered
1701 - Also identify meta-patterns: behaviors that emerge from the collective
1702 evidence but weren't explicitly stated in any single response
1703 - When responses conflict, examine their evidence strength
1704 - Final descriptions should capture the tool's strengths/weaknesses but
1705 NOT explicitly name other tools
1706 </synthesis_task>
1707
1708 <output_format>
1709 **Tool: [name]**
1710
1711 Synthesis reasoning:
1712 [Explain which patterns were most consistent across analyses, what
1713 emergent patterns were discovered, how conflicts were resolved, and why
1714 certain behaviors were included/excluded in the final description.]
1715
1716 Final description:
1717 [Single definitive description of when to use this tool. Describe its
1718 characteristics and optimal use cases WITHOUT referencing other tools by
1719 name. Example: "Best for queries requiring recent information or real-
1720 time data. Returns comprehensive results with detailed snippets. May be
1721 less effective for historical or archival content."]
1722 </output_format>

```

Figure 12: The BrowseComp domains consensus merge reflection prompt. The descriptions generated from the previous step are appended to this prompt.

```

1723
1724
1725
1726
1727

```