ToolHACE: A Dataset for Detecting Hallucinations in Tool-Based LLM Responses

Nikolay Ivanov[1,*], **Sergey Pletenev**[1,2,*], **Elisei Rykov**[1,4*], **Anatolii Frolov**[1,2,*], **Bogdan Monogov**[1], **Alexander Potekhin**[1], **Maria Marina**[1,2], **Daniil Moskovskiy**[1,2], **Viktor Moskvoretskii**[3], **Alexander Panchenko**[1,2], and **Ilseyar Alimova**[1]
[1]Skoltech, [2]AIRI, [3]EPFL, [4]T-Tech
Correspondence: {nikolay.ivanov, a.panchenko, i.alimova}@skol.tech

Abstract

Tool-augmented language models ground their responses in external tool outputs, improving reliability beyond parametric knowledge alone. However, even when the correct tool is called and returns accurate information, models may still contradict such outputs, omit critical details, or generate unsupported claims. Existing hallucination benchmarks for tool-augmented dialogue primarily focus on the tool-calling stage, evaluating whether the correct tool and arguments are selected, while overlooking errors that arise during post-tool response generation. To address this gap, we introduce ToolHACE, a span-level benchmark for hallucination detection in tool-augmented LLM responses, covering five fine-grained hallucination categories. Through extensive experiments with modern hallucination detectors, we show that post-tool-call hallucinations are widespread and remain challenging to detect. Our results further demonstrate that effective detection requires training on specialized tool-augmented corpora, while transfer from tool-agnostic hallucination datasets performs poorly in this setting. These findings highlight the need for dedicated benchmarks and models for reliable hallucination detection in tool-augmented language systems.

1 Introduction

Large language models (LLMs) are increasingly used as the reasoning core of tool-augmented agents, where the model decides when to call external APIs, search engines, calculators, or databases, and then synthesizes a final natural-language response based on the tool outputs (Patil et al., 2024, 2025). During the operation of such systems, various errors may arise, including incorrect tool selection (e.g., calling a search engine instead of a calculator), wrong tool arguments (malformed or incomplete inputs), missing tool invocation when external tools are required, incorrect tool-use sequencing leading to inconsistent or unusable outputs, and incorrect processing of tool outputs, where the model misinterprets or fails to properly incorporate the tool response.

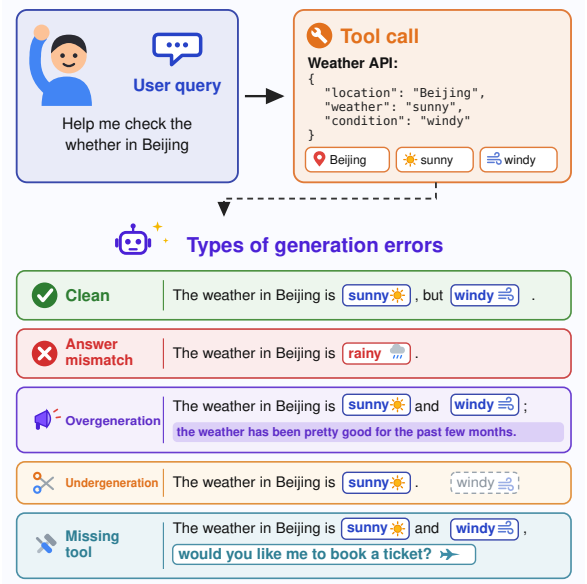

Figure 1: Main types of generation errors in tool-augmented LLM responses. A model can produce a correct answer, contradict the tool output, add unsupported information, omit a tool-provided fact, or introduce an unrelated tool/action request.

Most existing work on tool-augmented LLMs focuses on building large-scale benchmarks for tool calling itself, evaluating whether the model selects the correct API and provides valid arguments, while treating the final natural-language response as out of scope (Patil et al., 2024; Liu et al., 2025; Patil et al., 2025; Ye et al., 2025; Huang et al., 2024; Zhang et al., 2024). However models may still hallucinate in their final response and fail to properly utilize the tool output, resembling retrieval augmented generation (RAG) pitfalls. We refer to these as *post-tool-call hallucinations*: errors that occur *after* the right tool has been invoked, in the

[*]These authors contributed equally to this work.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 40915–40934
October 24-29, 2026 ©2026 Association for Computational Linguistics

Resource	Tool-calling setting	Span-level hallucination labels
ToolBench, BFCL, ToolACE, APIBench,	✓	✗
APIGen, ToolAlpaca, StableToolBench,	✓	✗
ToolHop, T-Eval, MetaTool	✓	✗
ToolEmu, InjecAgent, ToolBH	✓	✗
RAGTruth, FAVA, PsiloQA, HAD	✗	✓
Ours (ToolHACE)	✓	✓

Table 1: Existing resources cover either the tool-calling setting or span-level hallucination annotation, but not both. Our resource combines them.

natural-language response that the user actually reads. Detecting them requires evaluating the final response against the returned tool output, which is not addressed by existing tool-calling benchmarks.

To evaluate how well models can detect such hallucinations, we introduce ToolHACE, a dataset for detecting ungrounded generations in tool-augmented LLMs. Our dataset is built on top of ToolACE (Liu et al., 2025) and annotated at the span level with five categories: *Correct*, *Answer Mismatch*, *Overgeneration*, *Undergeneration*, and *Missing Tool*, as illustrated in Figure 1. We benchmark a range of modern hallucination detectors, including zero-shot prompted LLMs, encoder-based models trained on general-purpose hallucination corpora, and detectors fine-tuned on our dataset. Our results show that (i) post-tool-call hallucinations are widespread in modern tool-augmented LLMs, (ii) detectors trained solely on tool-agnostic hallucination data transfer poorly to this setting, and (iii) training on in-domain tool-calling data substantially reduces this performance gap.

Our contributions are:

- We introduce a new span-level dataset for detecting hallucinations in the post-tool-call stage of tool-augmented LLMs.
- We benchmark a range of state-of-the-art methods and modern LLM-based hallucination detectors on the proposed dataset.
- We evaluate the effectiveness of transferring knowledge from hallucination detection in RAG systems to the tool-augmented setting.

2 Related Work

Tool-calling datasets. Tool-use benchmarks evaluate whether models select the correct tool and pass valid arguments, scored by AST match, executable correctness, or task pass rate (Patil et al., 2024, 2025; Liu et al., 2025; Ye et al., 2025). Scalable synthetic pipelines have expanded coverage to tens of thousands of dialogues (Tang et al., 2023; Liu et al., 2024). A smaller line of work expands the threat model: MetaTool (Huang et al., 2024) tests abstention when no appropriate tool exists; ToolEmu (Ruan et al., 2024) simulates execution for safety; InjecAgent (Zhan et al., 2024) measures prompt-injection susceptibility; and ToolBH (Zhang et al., 2024) annotates tool-calling hallucination at the tool-name level. Across this family, evaluation centers on the tool call itself; the natural-language response synthesized after tool execution is treated as out of scope, and span-level hallucination annotations are absent throughout.

Span-level hallucination detection. A separate line of work provides fine-grained span- or word-level hallucination annotations, but in non-tool-calling settings. RAGTruth (Niu et al., 2024) annotates RAG outputs across QA, summarization, and data-to-text (with tabular rather than tool-call inputs); FAVA (Mishra et al., 2024) synthesizes spans from a fine-grained taxonomy; PsiloQA (Rykov et al., 2025) extends span annotation multilingually; and HAD (Xu et al., 2026) offers an 11-category hierarchical taxonomy over twelve NLG tasks. None of these contain structured tool outputs as input, and none include an analogue of our *Missing Tool* class. Models for this task fall into two camps: generative detectors that emit span markup (Niu et al., 2024; Mishra et al., 2024), and encoder-based token classifiers (Kovács and Recski, 2025; Team, 2026a); we evaluate representatives of both. Table 1 summarizes how existing resources cover one axis or the other but not both.

3 The ToolHACE Dataset

3.1 Hallucination Types

The dataset contains five hallucination labels, including the grounded class *Correct* and four post-tool-call hallucination categories: *Answer Mismatch*, *Overgeneration*, *Undergeneration*, and *Missing Tool*. *Undergeneration* is not annotated at the span level, whereas the other three hallucination categories have corresponding span-level annotations. Detailed definitions of each hallucination category are presented in Table 2. Three of these (Answer Mismatch, Overgeneration, Undergeneration) have rough analogues in prior hallucination taxonomies such as HAD (Xu et al., 2026) and RAGTruth (Niu et al., 2024). *Missing Tool*, however, is specific to tool-augmented settings and, to

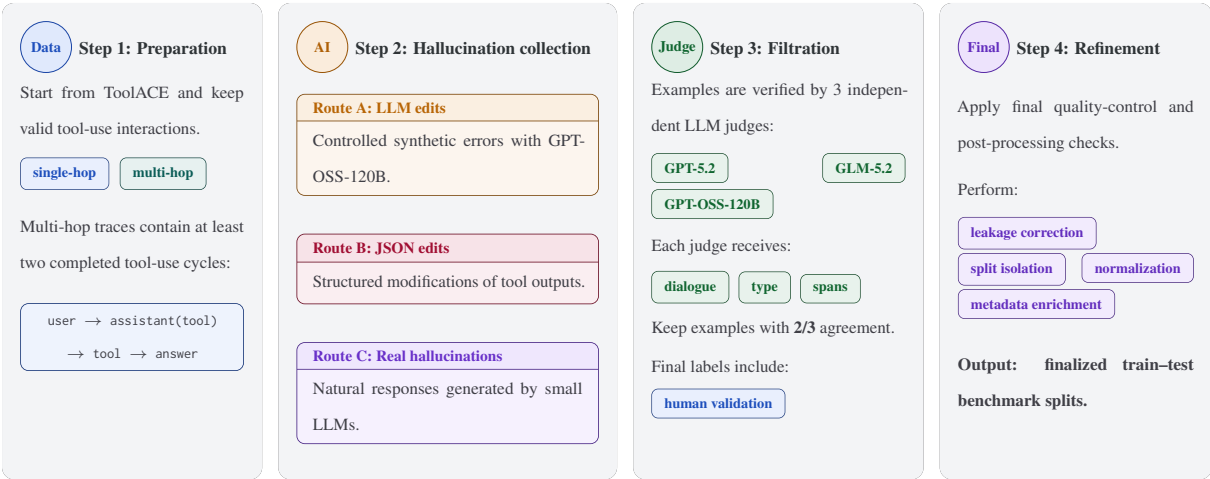

Figure 2: Dataset generation pipeline. ToolACE dialogues are split into single-hop and multi-hop traces. Hallucinations are collected through LLM-based synthetic edits, programmatic JSON perturbations, and naturally occurring (*wild*) generations from small LLMs. The resulting examples are filtered by multiple judges and subsequently refined through quality-control, split isolation, and normalization.

Class	Definition
Correct	Answer is faithful to the tool response and fully addresses the query.
Answer Mismatch	Answer asserts a value or fact that the tool response *contradicts*.
Overgeneration	Answer adds content with no support in the tool response (e.g. elaboration, background, general knowledge), without claiming any tool-mediated action.
Missing Tool	Answer references an *action or capability* that no available tool can supply (e.g. offering to book, send, fetch, or compute when no such tool exists).
Undergeneration	Answer omits information from the tool response that the query required.

Table 2: The five classes in ToolHACE. Boundaries are designed to be mutually exclusive: *Answer Mismatch* requires a contradicting value in the tool response; *Overgeneration* adds free-form content; *Missing Tool* specifically claims tool-mediated capability.

the best of our knowledge, has no direct counterpart in existing benchmarks. It captures cases where a model fabricates tool-mediated capabilities, such as claiming to have booked a reservation, retrieved a live price, or sent a message when no such tool exists or was called. We describe the synthesis pipeline for each non-Correct class below.

3.2 Source Corpus and Data Preparation

We build ToolHACE on ToolACE, a large-scale tool-calling corpus spanning diverse APIs and application domains (Liu et al., 2025). Compared to existing tool-calling benchmarks, ToolACE is particularly well suited for our setting due to its structured interaction traces comprising user queries, assistant tool invocations, and corresponding tool outputs. This explicit grounding structure enables fine-grained analysis of model behavior after tool execution. In addition, its broad coverage of heterogeneous tools and realistic usage scenarios increases the diversity of the resulting benchmark.

Each ToolACE sample serves as a seed instance for our construction pipeline. The original grounded answer is retained as a positive *Correct* example, while the associated question–tool-response pair is used to synthesize one or more hallucinated variants.

We start from the full ToolACE corpus of 11,300 dialogues and retain only samples containing at least one complete tool-use interaction of the form: user query → assistant tool call → tool response → assistant answer. The retained dialogues are then divided into two subsets. *Singlehop* dialogues contain exactly one tool invocation, whereas *multihop* dialogues contain at least two complete tool-use cycles.

For multihop dialogues, we remove trailing follow-up questions to ensure that the final assistant response is self-contained. For singlehop dialogues, we observe that many samples terminate immediately after the tool call without an explicit tool response or final assistant answer. For these cases, we synthetically generate the missing tool response and grounded completion using an

LLM, followed by structural filtering to remove malformed examples and multihop artifacts. The resulting dataset therefore combines real and synthetically completed singlehop and multihop instances while preserving traceability to the original ToolACE source through `dialogue_id` and `source_row_index`.

3.3 Hallucination Insertion Pipeline

Design principle. Rather than asking a large model to "produce a hallucinated answer" in free form, which makes spans hard to ground and introduces stylistic drift unrelated to the targeted errors we construct each non-Correct sample by applying a single, well-defined edit to either the tool response or the answer. The edit defines the error span mechanically, with no separate annotation step.

Two generation routes. We use three routes in parallel and pool the outputs. **Route A** (LLM editor) uses GPT-OSS-120B (OpenAI et al., 2025) with 5-shot prompts (Appendix B); it provides stylistic diversity. **Route B** (structured edits) operates directly on the JSON tool response, with Qwen3-14B-Instruct only where free-form generation is needed (e.g. Type 3 sentence injection); it provides precise control over span location and severity. Using both routes hedges against single-model bias. Concrete examples appear in Figure 3. **Route C** (real setup) generate natural post-tool-call responses with multiple open-weight instruction-tuned LLMs from diverse model families and use independent LLM-based judges, followed by selective arbitration, to identify hallucinations, classify their types, and localize unsupported spans.

Route A: LLM editor. For each of the four non-Correct classes, we prompt GPT-OSS-120B with a class-specific 5-shot template (Appendix B) to produce an edited (tool response, answer) pair from a Correct seed. For *Answer Mismatch*, the model rewrites one to three values in the tool response so the existing answer becomes contradicted. For *Overgeneration*, it deletes fields from the tool response so the existing answer references information no longer supported. For *Undergeneration*, it appends a one-sentence elaboration to the answer that adds unsupported content without referencing any tool. For *Missing Tool*, it appends a one-sentence offer of a tool-mediated action (e.g. booking, sending, scheduling) for which no corresponding tool is listed in the system prompt.

Route B: Answer Mismatch. We treat the tool response as a typed JSON tree, mask all leaves, and selectively unmask a subset. The masked leaves are filled with values sampled from other tool calls of the *same schema*, guaranteeing type consistency while introducing factual conflict. We use two unmasking strategies: (i) *independent masking*, in which each leaf is corrupted with fixed probability so that errors are scattered and statistically independent across siblings (Figure 3, Type 1a); (ii) *cascade masking*, in which we select one non-root internal node and corrupt it together with all its descendants, leaving sibling subtrees untouched (Figure 3, Type 1b). The latter mirrors the realistic case where a single subobject of a tool's output is wrong while the rest of the structure is correct.

Route B: Overgeneration. We remove a subset of fields from the tool response before it is passed to the model. Deprived of the deleted information but trained on the underlying domain, the model tends to fabricate plausible-sounding values for the missing fields rather than abstain. Span labels are derived by aligning the model's answer against the *original* (untruncated) tool response: any value present in the answer but unsupported by the truncated context is marked as a Type 2 span (Figure 3, Type 2).

Route B: Missing Tool. We use Qwen2.5-14B to append a one-sentence addendum to the ground-truth answer, referencing a tool selected from a global pool of tools collected from the dataset's system prompts. The selected tool is absent from the system prompt of the current trace. Although the injected sentence is factually coherent in isolation, it is unsupported by the tools available in the current trace. The entire injected sentence is annotated as the hallucinated span (Figure 3, Type 3).

Route C: Real setup. We generate naturally occurring post-tool-call responses using four small open-weight instruction-tuned models spanning three families: `Llama-3.2-3B-Instruct` (Team, 2024), `Qwen3.5-2B/4B-Instruct` (Team, 2026b), and `Gemma-4-26B-A4B` (Abd et al., 2026). We sample 2,500 ToolACE single-hop dialogues uniformly at random without replacement from examples containing at least one tool-response turn (seed 42).

For each example, the model receives the original dialogue prefix up to and including the last

Type 1a Answer Mismatch— Independent

Query: *Convert CMYK (C=20%, M=40%, Y=60%, K=10%) to RGB.*

```
Tool:
{"R": 95, "G": 72, "B": 54}
```

Model answer

The CMYK code converts to RGB:
R= 184 , G= 147 , B= 115 .

↪ Each leaf value is replaced independently with a random value.

Type 1b Answer Mismatch— Cascade

Query: *Wall Street analyst recommendations for AAPL, TSLA, and AMZN?*

```
Tool:
{"TSLA": {"buy": 15, "hold": 20},
"AAPL": {"buy": 17, "hold": 3}},
```

Model answer

TSLA: 15 Buy, 20 Hold

AAPL: 30 Buy, 10 Hold

↪ Corruption propagates through the AAPL subtree; sibling branches are unaffected.

Type 2 Overgeneration — Field Deletion

Query: *Convert CMYK (C=20%, M=40%, Y=60%, K=10%) to RGB.*

```
Original: {"R": 184, "G": 147,
"B": 115}
Truncated: {"G": 147} (R, B
deleted)
```

Model answer

The CMYK code converts to RGB:
R= 184 , G=147, B= 115 .

↪ Fields R and B were deleted from the tool response; the model fabricates their values.

Type 3 Missing Tool — Spurious Sentence

Query: *Convert CMYK (C=20%, M=40%, Y=60%, K=10%) to RGB.*

```
Tool: {"R": 184, "G": 147, "B": 115}
```

Model answer

The CMYK code converts to RGB: R=184, G=147, B=115.

It's worth noting that you can also retrieve detailed task information within your organization using the appropriate tools.

↪ The appended sentence references a capability that is unsupported by any available tool.

Figure 3: Examples of Route B Hallucination Insertion in ToolHACE. **Type 1a** (Answer Mismatch): individual leaf values in the tool response are replaced with random values from other samples—each field is corrupted independently. **Type 1b** (Answer Mismatch): all values within a JSON subtree are replaced together, preserving sibling branches. **Type 2** (Overgeneration): fields are deleted from the tool response, causing the model to fabricate values it can no longer ground. **Type 3** (Missing Tool): the model appends a factually unsupported sentence referencing context absent from the tool response.

tool response: the system instruction and tool definitions, preceding user and assistant turns, executed tool call(s), and returned tool output(s). The reference ToolACE final answer is removed, and each model generates one new natural-language answer with temperature 0.7, top-p 0.95, and a maximum of 1,024 output tokens. Tool definitions, calls, and outputs are passed using the deployed OpenAI-compatible tool-calling format and the model-specific chat template; internal reasoning is disabled for the Qwen models. After removing failed generations and outputs that do not constitute a natural-language final answer, 9,163 responses remain.

Each retained response is independently annotated by two LLM judges from different model families, `DeepSeek-v4-Flash` and `GLM-4.7`. The judges receive the user query, tool output, and generated response and predict a binary hallucination label, primary error type, and unsupported spans. We subsequently use `GPT-4o` to arbitrate all cases labeled as hallucinated by both judges and a random 15% sample of judge disagreements.

3.4 Dataset Filtration

Multi-judge verification. We verify the raw test split using three independent judge models from different families: GPT-5.2, GPT-OSS-120B (OpenAI et al., 2025), and GLM 5.2. Each judge receives the full dialogue, the hallucination type label, and the span annotations, and is asked to confirm whether (i) the clean rows are free of hallucinations and (ii) the span labels for hallucinated rows correctly identify the hallucinated text. We retain test

examples on which at least two of the three judges agree that the annotation is correct, yielding 11,606 verified test examples (90.6% of the raw test set). The two GPT-family judges (GPT-5.2 and GPT-OSS-120B) rejected 0.3% and 3.2% of examples respectively, while GLM 5.2 rejected 4%.

Human Validation. To assess annotation quality, we conducted a human validation study with 10 annotators, each reviewing 50 randomly sampled examples for label correctness and span consistency, resulting in 500 validated instances overall. The annotations showed high agreement with the dataset labels across all categories. Accuracy reached 97% for both *clean* and *undergeneration*, 96% for *answer_mismatch*, 93% for *overgeneration*, and 90% for *missing_tool*.

Manual inspection of the disagreements revealed that nearly all mismatches corresponded to borderline or ambiguous cases rather than clear annotation errors. For *overgeneration* and *undergeneration*, the most common source of disagreement concerned class assignment rather than span localization: annotators typically agreed on the highlighted span but differed in whether the content should be interpreted as excessive or insufficient. In the *clean* category, disagreements mainly arose from assistant utterances offering additional follow-up actions, which some annotators interpreted as *overgeneration* despite remaining consistent with the tool response. In *undergeneration*, ambiguity often emerged when the assistant confirmed successful execution of a request involving parameter settings without explicitly restating the applied parameters, making such responses plausibly interpretable as incomplete.

3.5 Final Dataset Refinement

Lexical-leakage correction. During quality review, we identified a near-universal lexical artifact in *missing_tool* examples: 99.0% of instances (1,497 in the test split; 6,062 in train) ended with the phrase *"Would you like…"*, making the hallucination type trivially predictable from surface form alone, without any understanding of content. We replaced all such occurrences with semantically equivalent continuations and recomputed the corresponding span offsets to remain consistent with the edited text. The final dataset is free of this shortcut. In additional we filled the *correct* class with same patterns, but also extended their tools, so these traces become correct. This gives another advantage, so these patterns exist both in *missing_tool* and *correct* which blocks any model to find out simple skip for classification.

Length pattern. Another check of raw dataset showed that two hallucination classes could be identified from the length of the final answer alone, without reading its content. *Overgeneration* adds baseless content, so its answers are systematically longer than clean ones, while *undergeneration* omits items from the tool output and therefore has no error spans and is systematically short. We fixed it with counterfactual clean twins generated by Qwen3.6-35B-A3B that keep the answer text verbatim and change only the context so that the label flips to clean. For *overgeneration*, the tool response was augmented so that every previously baseless claim became supported; for *undergeneration*, the tool response and user request were trimmed so that the short answer no longer omits anything.

Train/test isolation. To prevent data leakage, we removed all examples whose `dialogue_id` appeared in the test split. This affected a substantial number of *clean* training examples, as many source ToolACE dialogues contributed both clean and hallucinated variants.

Post-processing. We apply a Unicode normalization pass to all conversation turns, replacing typographic characters (non-breaking hyphens, narrow no-break spaces, curly quotes, ellipses, etc.) with their ASCII equivalents. Additionally, we enrich each row with three convenience fields: `available_tools` (a list of tool names extracted from the system prompt), `system_prompt_format` (a structured representation of tool descriptions for easier parsing), and `turn_role` (an explicit role tag added to each conversation turn, e.g. `tool_call` or `tool_response`).

3.6 Dataset Statistics

The dataset contains 48,741 samples in total, with 37,135 instances in the training set and 11,606 in the test set. The split is stratified by hallucination type to ensure representation of all error categories across both partitions. Table 3 summarizes the final train–test split of ToolHACE.

Among the hallucination categories, *Overgeneration* and *Missing Tool* constitute the largest portions of the dataset, reflecting the prevalence of unsupported elaborations and fabricated tool-

Type	Train	Test
clean	16,335	5,252
overgeneration	7,511	2,316
missing_tool	4,931	1,521
answer_mismatch	3,702	1,196
undergeneration	4,656	1,321
Total	**37,135**	**11,606**

Table 3: Overall ToolHACE dataset statistics across hallucination types and train/test splits.

mediated capabilities in tool-augmented generation. *Undergeneration* is also well represented, while *Answer Mismatch* is comparatively less frequent due to the stricter synthesis constraints required to generate semantically valid contradictions with the tool response. The *Correct* class provides grounded positive examples and serves as the reference class for detector training and evaluation, with a number of examples approximately equal to the combined number of examples across all hallucination classes.

4 Experimental Setup

We evaluate tool-grounded hallucination detection at two levels: response-level detection and span-level localization. At the response level, a prediction is positive if the answer contains at least one hallucinated span. At the span level, we evaluate whether the model recovers the exact erroneous text segments; since span localization is the main goal of ToolHACE, we use span-level F1 as the primary metric. Full metric definitions are provided in Appendix A.

We compare three groups of systems. First, we evaluate GPT-4o, GPT-5.4-mini and Kimi 2.5 in zero-shot and few-shot prompting settings, where the model receives the tool-calling trace and outputs a structured JSON prediction without parameter updates. Second, we evaluate transfer baselines that rely on supervision from external hallucination detection sources, including synthetic tool-calling corruptions, PsiloQA, RAGTruth, and HAD. Third, we fine-tune models on the in-domain training split of ToolHACE, which serves as a supervised upper baseline with no domain mismatch between train and test data.

Our trainable models include encoder-only and generative backbones. For encoder-only models, we use `ModernBERT-base` and `ModernBERT-large` (Warner et al., 2025). For generative models, we use `Qwen3.5-0.8B`, `Qwen3.5-2B` (Team, 2026b). Generative models are trained to produce structured JSON annotations containing the predicted error type and hallucinated spans. We also include two publicly available `lettucedetect` checkpoints trained on RAGTruth, `lettucedetect-base` and `lettucedetect-large`, as off-the-shelf span-level transfer baselines.

All systems are evaluated on the same held-out ToolHACE test set. Appendix A describes the prompt format, model inputs, dataset conversions, HAD label mapping, and synthetic data construction.

5 Results

Table 4 reports response-level and span-level performance on the ToolHACE test set. Since our main goal is to localize hallucinated spans, we treat span-level F1 as the primary metric and use response-level scores as a complementary measure of whether a model detects that an error is present at all.

In-domain supervision provides the strongest localization performance. Models trained directly on ToolHACE produce the strongest overall span-localization results, with the Qwen-LoRA models clearly outperforming the other systems. Qwen3.5-2B-LoRA achieves the highest span-level F1 of 79.28, followed by its smaller Qwen counterpart. The in-domain ModernBERT models also perform strongly, although their advantage over the best zero-shot baseline is more modest. The larger separation from the transfer baselines indicates that matching the training data to the structure and hallucination taxonomy of the target setting is particularly important for precise span localization.

Zero-shot prompting is competitive, but few-shot prompting is unexpectedly harmful. All three zero-shot LLMs provide strong non-specialized baselines and outperform the supervised transfer systems on span localization. Their results are also relatively close, suggesting that this capability is not specific to a single model family.In contrast, few-shot prompting sharply reduces span-level performance for every evaluated LLM. The examples appear to interfere with precise span extraction, possibly by encouraging imitation of the demonstrated output patterns rather than careful grounding in the tool trace. In particular, models often generate redundant content, such as repeat-

Setting	Data	Model	Span-level				Response-level					
			Mis-match	Over-gen.	Missing tool	Avg.	Correct	Mis-match	Over-gen.	Missing tool	Avg.	Under-gen.*
Zero-Shot	-	GPT-4o	51.15	68.87	77.52	65.85	69.52	29.96	49.09	35.12	45.92	2.98
		GPT-5.4-mini	52.80	66.84	76.97	65.54	67.98	28.81	47.56	34.27	44.66	4.39
		Kimi	55.66	73.99	73.58	67.74	69.79	29.45	49.77	34.39	45.85	4.13
Few-Shot	-	GPT-4o	3.58	15.66	17.28	12.17	67.97	29.16	48.68	33.75	44.89	3.96
		GPT-5.4-mini	2.71	14.03	17.19	11.31	59.31	25.73	45.09	29.15	39.82	8.11
		Kimi	4.75	16.76	21.18	14.23	67.56	28.21	48.99	33.25	44.50	5.09
Transfer SFT	NQ, CNNDM	LookbackLens	0.95	0.82	0.63	0.80	19.21	18.81	53.04	22.80	28.46	-
	Glaive	ModernBERT-large	30.46	24.79	32.38	29.21	62.67	25.47	45.18	27.97	40.32	-
		ModernBERT-base	26.57	19.47	25.88	23.97	63.82	24.67	46.84	24.18	39.88	-
		Qwen3.5-0.8B-LoRA	51.01	23.94	38.68	37.88	63.50	40.57	26.21	24.08	38.59	3.21
		Qwen3.5-2B-LoRA	58.07	30.64	53.84	47.52	64.73	34.64	34.10	29.71	40.80	4.10
	PsiloQA	ModernBERT-large	6.97	4.87	2.90	4.91	4.52	18.93	33.45	23.42	20.08	-
		ModernBERT-base	4.14	7.48	4.04	5.22	0.15	18.71	33.31	23.19	18.84	-
		Qwen3.5-0.8B-LoRA	36.88	34.40	39.56	36.95	46.03	22.32	38.84	26.17	33.34	17.19
		Qwen3.5-2B-LoRA	41.17	38.69	52.25	44.04	61.26	24.63	42.00	27.75	38.91	**18.97**
	RAGTruth	ModernBERT-large	10.74	55.15	66.76	44.22	65.55	21.37	49.33	34.87	42.78	-
		ModernBERT-base	8.97	45.55	42.95	32.49	63.34	23.73	47.11	31.33	41.38	-
		Qwen3.5-0.8B-LoRA	22.12	44.05	42.24	36.14	72.49	37.37	54.21	22.98	46.76	2.51
		Qwen3.5-2B-LoRA	22.09	54.79	55.97	44.28	74.76	34.94	55.37	28.33	48.35	4.54
	HAD	ModernBERT-large	6.9	21.1	12.1	13.37	66.12	16.79	47.80	19.01	37.43	-
		ModernBERT-base	1.26	15.75	4.62	7.21	65.13	5.23	49.63	9.90	32.47	-
		Qwen3.5-0.8B-LoRA	26.09	29.08	32.16	29.11	69.96	36.52	47.57	21.37	43.85	3.52
		Qwen3.5-2B-LoRA	31.39	28.00	43.78	34.39	71.89	40.86	41.53	24.99	44.82	9.73
In-Domain SFT	ToolHACE split	ModernBERT-large	64.91	69.27	77.97	70.72	**87.01**	36.01	60.73	**45.43**	57.30	-
		ModernBERT-base	61.94	68.68	75.92	68.85	86.90	35.74	60.65	45.29	57.15	-
		Qwen3.5-0.8B-LoRA	67.71	82.51	79.01	76.41	86.01	36.36	60.09	44.48	56.73	2.28
		Qwen3.5-2B-LoRA	**72.15**	**83.88**	**81.80**	**79.28**	86.99	**36.95**	**60.91**	45.34	**57.55**	1.77

Table 4: F1 performance on the ToolHACE test dataset by hallucination type. At the span level, *Correct* is not evaluated, as span-level metrics are computed only for hallucinated examples where the model correctly identifies the hallucination type. *Undergen.* is reported only for decoder-based models, as undergeneration has no corresponding annotated span and encoder-based models therefore produce an empty span for this category.

ing the tool call or its response, causing deviations from the required output format and substantially degrading performance. The resulting increase in context length may further contribute to this degradation.

External supervision transfers unevenly across datasets and architectures. Glaive-based synthetic supervision produces the strongest individual transfer result when paired with Qwen3.5-2B-LoRA, while RAGTruth provides more consistent performance across the evaluated backbones. In particular, RAGTruth transfers substantially better to ModernBERT, whereas Glaive is stronger than RAGTruth for the Qwen-LoRA models. This interaction shows that transfer quality depends not only on the source dataset but also on how effectively the target architecture adapts to its annotation patterns.

Tool-agnostic supervision is highly architecture-dependent. PsiloQA and HAD transfer poorly to the ModernBERT models, particularly at the span level, but their performance improves considerably when paired with Qwen-LoRA. PsiloQA with the larger Qwen model approaches the strongest RAGTruth results, showing that tool-agnostic supervision is not uniformly ineffective. Rather, its usefulness depends strongly on the model's ability to adapt the source annotations to structured tool-grounded inputs. LookbackLens, meanwhile, provides very limited span-localization performance.

Response-level performance shows a different pattern from span localization. In-domain supervision still gives the strongest response-level results, although the four in-domain models form a tight cluster, suggesting that backbone choice matters less for detecting the presence of an error than for locating it precisely. Among the transfer settings, RAGTruth performs most consistently, while the zero-shot LLMs remain competitive with the strongest externally supervised models. Notably, few-shot prompting leaves response-level performance broadly similar to zero-shot prompting despite causing a sharp decline in span-level F1. This contrast shows that response-level scores alone can conceal substantial failures in localization. Performance also remains uneven across error types: grounded responses and overgeneration are gener-

Type	Synth (%)	Real-World (%)
overgeneration	35.0	29.8
missing_tool	32.2	23.3
undergeneration	22.2	32.7
answer_mismatch	10.6	14.2
# hallucinated samples	4,693	2,155

Table 5: Type distribution among positives: ToolHACE test split vs. real-world hallucinated responses.

ally easier to identify, whereas answer mismatch, missing-tool errors, and especially the separately evaluated undergeneration class remain more challenging.

Undergeneration remains a distinct challenge. Performance for undergeneration remains weak, including for the strongest in-domain Qwen models. Unlike the other hallucination types, undergeneration corresponds to missing information and therefore has no explicit erroneous span in the generated response. It is consequently better treated as a response-level detection problem than as a span-localization task.

6 Ablation Study

We evaluate the practical utility of ToolHACE through a cross-distribution analysis comparing its hallucination type distribution with real-world errors from deployed small LLMs. We compare hallucination type distributions between synthetic and real-world outputs. As shown in Table 5, the distributions are closely aligned, indicating that our synthetic corruption procedure captures the main failure modes observed in real model outputs. While synthetic data contains more *overgeneration* and *missing_tool*, real outputs show higher proportions of *undergeneration* and *answer_mismatch*, reflecting natural variation rather than structural mismatch. Overall, the low divergence supports the use of synthetic data as a proxy for real-world post-tool-call hallucinations.

7 Conclusion

We introduced a new span-level dataset for detecting post-tool-call hallucinations in tool-augmented language models, focusing on errors that occur after tool execution. We conducted experiments on both in-domain training and cross-dataset transfer, including evaluation of models trained on RAG-style hallucination data. Our results show that effective detection of post-tool-call hallucinations requires supervision from data that matches the target distribution, as tool-agnostic datasets transfer poorly to this setting. We further demonstrate that our dataset aligns well with real-world failure modes observed in small deployed LLMs, indicating its practical relevance for studying hallucination behavior in tool-augmented systems.

Limitations

Our work has several limitations that should be considered when interpreting the results.

First, although we perform manual validation on a subset of the dataset to ensure annotation quality, the full dataset has not been exhaustively human-verified due to its scale. As a result, some residual annotation noise or synthesis artifacts may remain.

Second, our dataset is constructed on top of ToolACE (Liu et al., 2025), which may introduce biases inherited from the underlying corpus, including its domain coverage, tool selection distribution, and interaction patterns. Consequently, our benchmark may not fully reflect all possible real-world tool-usage scenarios.

Third, the dataset relies partly on synthetic generation procedures to construct hallucination variants. While we apply filtering and quality control steps, synthetic construction may still introduce distributional artifacts that differ from naturally occurring model errors.

Fourth, each dialogue in our dataset is annotated with a single dominant hallucination type, and we do not consider multi-label settings where multiple hallucination types may co-occur within the same interaction. This simplifies the annotation and evaluation setup but limits the modeling of more complex and realistic error compositions.

Finally, our evaluation focuses primarily on post-tool-call hallucinations in English-language tool-augmented settings. This limits the generalizability of our findings to multilingual scenarios and other forms of agentic workflows beyond tool calling, such as planning-heavy or long-horizon decision-making systems.

Datasets described in the present publication and source codes to reproduce the experiments are openly available online.[1]

Use of AI Tools

During the preparation of this work, the authors used LLM assistants to improve the language qual-

[1] https://github.com/s-nlp/Tool-Call-HD

ity of the manuscript, including grammar and style checking and rewording for clarity. The authors reviewed and edited all content as needed and take full responsibility for the content of the publication.

Acknowledgements

The work was supported by the grant for research centers in the field of AI provided by the Ministry of Economic Development of the R.F. in accordance with the agreement 000000C313925P4F0002 and the agreement with Skoltech №139-10-2025-033.

References

Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Carbune, Michelle Casbon, Mayank Chaturvedi, Aditya Chawla, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, and 80 others. 2026. Gemma 4 Technical Report. *CoRR*, abs/2607.02770.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024. MetaTool Benchmark for Large Language Models: Deciding Whether to Use Tools and Which to Use. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Ádám Kovács and Gábor Recski. 2025. LettuceDetect: A Hallucination Detection Framework for RAG Applications. *CoRR*, abs/2502.17125.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, and 1 others. 2025. Toolace: Winning the points of llm function calling. In *International conference on learning representations*, volume 2025, pages 41359–41381.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh R. N., Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024. APIGen: Automated PIpeline for Generating Verifiable and Diverse Function-Calling Datasets. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Abhika Mishra, Akari Asai, Vidhisha Balachandran, Yizhong Wang, Graham Neubig, Yulia Tsvetkov, and Hannaneh Hajishirzi. 2024. Fine-grained Hallucination Detection and Editing for Language Models. In *First Conference on Language Modeling*.

Cheng Niu, Yuanhao Wu, Juno Zhu, Siliang Xu, KaShun Shum, Randy Zhong, Juntong Song, and Tong Zhang. 2024. RAGTruth: A hallucination corpus for developing trustworthy retrieval-augmented language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10862–10878, Bangkok, Thailand. Association for Computational Linguistics.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, and 108 others. 2025. gpt-oss-120b & gpt-oss-20b Model Card. *Preprint*, arXiv:2508.10925.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2024. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Elisei Rykov, Kseniia Petrushina, Maksim Savkin, Valerii Olisov, Artem Vazhentsev, Kseniia Titova, Alexander Panchenko, Vasily Konovalov, and Julia Belikova. 2025. When Models Lie, We Learn: Multilingual Span-Level Hallucination Detection with PsiloQA. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 11663–11682, Suzhou, China. Association for Computational Linguistics.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases. *CoRR*, abs/2306.05301.

Llama Team. 2024. The Llama 3 Herd of Models. *CoRR*, abs/2407.21783.

LLM Semantic Router Team. 2026a. ModernBERT-32K Hallucination Detector with Data2txt Augmentation.

Qwen Team. 2026b. Qwen3.5-Omni Technical Report. *CoRR*, abs/2604.15804.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Griffin Thomas Adams, Jeremy Howard, and Iacopo Poli. 2025. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 2526–2547. Association for Computational Linguistics.

Fan Xu, Xinyu Hu, Zhenghan Yu, Li Lin, Xu Zhang, Yang Zhang, Wei Zhou, Jinjie Gu, and Xiaojun Wan. 2026. HAD: HAllucination Detection Language Models Based on a Comprehensive Hallucination Taxonomy. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track), ACL 2026, San Diego, California, USA, July 2-7, 2026*, pages 141–158. Association for Computational Linguistics.

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. 2025. ToolHop: A Query-Driven Benchmark for Evaluating Large Language Models in Multi-Hop Tool Use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 2995–3021. Association for Computational Linguistics.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, volume ACL 2024 of *Findings of ACL*, pages 10471–10506. Association for Computational Linguistics.

Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, Tetsuya Sakai, Tian Feng, and Hayato Yamana. 2024. ToolBeHonest: A Multi-level Hallucination Diagnostic Benchmark for Tool-Augmented Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 11388–11422. Association for Computational Linguistics.

A Experimental Details

A.1 Evaluation Metrics

Response-level hallucination detection. We evaluate response-level hallucination detection by converting span annotations into binary labels. A response is labeled as hallucinated if it contains at least one hallucinated span and non-hallucinated otherwise. We report precision, recall, and F1 with macro averaging between two classes. This metric measures whether a model can detect the presence of any hallucination in the response, regardless of whether it can localize the erroneous text.

Span-level hallucination localization. We further evaluate span-level performance using span precision, recall, and F1. Predicted and gold spans are matched using greedy pairing based on Intersection-over-Union (IoU) with 0.75 threshold. A predicted span is counted as correct if its IoU with a gold span exceeds a fixed threshold. Unmatched predicted spans are false positives, and unmatched gold spans are false negatives. This metric is stricter than character-level overlap because it rewards models for recovering hallucinations as coherent text segments rather than partially overlapping with the gold annotation. Since ToolHACE is designed for span-level hallucination detection, we treat span-level F1 as the primary metric.

A.2 Evaluation Settings

We evaluate hallucination detectors under three settings: zero-shot prompting, few-shot prompting, and supervised fine-tuning.

Zero-shot prompting. In the zero-shot setting, the model receives the full tool-calling trace and task instructions, but no annotated examples. The model is asked to output a structured JSON prediction containing the predicted error type and hallucinated spans. This setting measures how well a general-purpose LLM can perform tool-grounded hallucination detection without task-specific examples or parameter updates.

Few-shot prompting. In the few-shot setting, the model receives the same task instructions together with a small set of annotated examples. The output format is identical to the zero-shot setting: the model must return a JSON object of the form `{"type": ..., "spans": [...]}`. We use GPT-4o for both zero-shot and few-shot prompting. The full prompt template is shown in Appendix C.

Supervised fine-tuning. In the SFT setting, trainable backbones are fine-tuned on hallucination-annotated data and evaluated on the held-out ToolHACE test set. We distinguish between transfer SFT, where supervision comes from external datasets, and in-domain SFT, where the model is trained on the ToolHACE training split. The in-domain setting is not intended as an off-the-shelf transfer baseline; rather, it estimates the performance achievable when supervised examples from the target domain are available.

A.3 Models

Encoder-only models. For encoder-only span classification, we use `ModernBERT-base` and `ModernBERT-large` (Warner et al., 2025). These models consume the full input trace and predict hallucination labels over the response tokens. They are used for experiments with synthetic supervision, PsiloQA, HAD, RAGTruth-based checkpoints, and the in-domain training split.

Generative models. We also evaluate generative backbones that produce annotations as structured text. Each example is converted into a chat-style SFT instance consisting of a system message, a user message, and an assistant message. The user message contains the full tool-calling trace, including the user query, tool calls, tool responses, and assistant answer. The assistant message contains the target annotation as JSON:

$$\{\texttt{"type"}: \ldots, \texttt{"spans"}: [\ldots]\}.$$

This formulation allows decoder-only and encoder–decoder models to solve the task as conditional generation rather than token-level classification. Our experiments include `Qwen3.5-0.8B`, `Qwen3.5-2B` (Team, 2026b). For Qwen models, we use LoRA fine-tuning in the experiments marked with `-LoRA` with $rank = 16, alpha = 32$.

Off-the-shelf RAGTruth detectors. We evaluate two publicly available `lettucedetect` checkpoints, `lettucedetect-base` and `lettucedetect-large`. Both models use `ModernBERT` encoders and are trained on RAGTruth. We include them as off-the-shelf span-level transfer baselines: they are not fine-tuned on ToolHACE, but they have been trained on external span-level hallucination annotations.

A.4 Supervision Sources

We use five sources of supervision or adaptation: Synthetic, PsiloQA, RAGTruth, HAD, and the in-domain ToolHACE training split.

Glaive. We construct a synthetic tool-calling hallucination dataset from the Glaive function-calling corpus.[2] We first filter the corpus to retain only examples containing tool calls, yielding approximately 15k source rows. Tool definitions are extracted and placed into the system prompt. We then apply the synthetic corruption procedure described in Section 3.3. The resulting dataset contains 45,584 single-hop examples, with 14% clean and 86% hallucinated samples. This setting tests whether synthetic tool-calling perturbations provide useful supervision for span-level hallucination localization.

PsiloQA. We use the English subset of PsiloQA as an out-of-domain hallucination detection source. For encoder-based models, we follow the input format and hyperparameter setup used by `lettucedetect`. Since PsiloQA does not target tool-calling traces directly, this setting tests whether supervision from general RAG-style hallucination detection transfers to our benchmark.

RAGTruth. For RAGTruth, we do not train new checkpoints ourselves. Instead, we evaluate the publicly available `lettucedetect-base` and `lettucedetect-large` models, which are trained on RAGTruth. RAGTruth is included because it provides span-level hallucination supervision and contains examples where generated answers must be checked against external evidence. This makes it a natural transfer baseline for tool-grounded hallucination localization.

HAD. We fine-tune models on HAD (Xu et al., 2026), a hallucination annotation dataset with a fine-grained taxonomy for general NLG outputs. To make HAD compatible with our label space, we map its labels to our taxonomy. *Contradiction with Input Content* and factuality-related errors, including *Factual Recall Error*, *Factual Inference Error*, *Fabricated Entity*, and *Fictional Attribution*, are mapped to *Answer Mismatch*. *Baseless Information* is mapped to *Overgeneration*, and *Information Omission* is mapped to *Undergeneration*. The remaining categories, including *Task Type Inconsistency*, *Task Requirement Inconsistency*, *Contradiction within Output Content*, and *Structural Incoherence*, fall outside our taxonomy and are either discarded or relabeled as *Correct* when the output remains faithful to the tool response. Importantly, HAD contains no analogue of *Missing Tool*, so models trained on HAD receive no direct supervision for this failure mode.

ToolHACE training split. Finally, we fine-tune models on the training split of ToolHACE. Unlike external supervision sources, this data uses our native taxonomy and includes all error classes, including *Missing Tool*. This setting represents the in-domain supervised baseline and measures the benefit of training directly on tool-calling hallucination annotations.

A.5 Input and Output Format

For all models, the input contains the information required to judge whether the assistant answer is faithful to the tool-calling trace. This includes the user query, available tool definitions when applicable, executed tool calls, tool responses, and the final assistant answer. Encoder-only models predict labels over the assistant answer tokens. Generative models output a JSON object with two fields: the predicted error type and a list of hallucinated spans.

For non-hallucinated examples, the expected span list is empty. For hallucinated examples, each span corresponds to a contiguous segment of the assistant answer that is unsupported by or inconsistent with the tool response. All outputs are post-processed into the same normalized span representation before evaluation.

[2] https://huggingface.co/datasets/glaiveai/glaive-function-calling

B Prompts

B.1 Real-Response Generation

We sample 2,500 examples uniformly at random without replacement from ToolACE single-hop dialogues containing at least one tool-response turn (seed 42). For each example, we remove the reference final assistant answer and retain the dialogue prefix up to and including the last tool response. Thus, the generation input contains the original system instruction and tool definitions, preceding user and assistant turns, executed tool call(s), and returned tool output(s).

Generation input template

System: original ToolACE system instruction
Tools: ToolACE definitions in OpenAI function format
User: original dialogue turn(s)
Assistant tool call: executed ToolACE call(s)
Tool: original ToolACE tool response(s)
Assistant: newly generated natural-language answer

The structured fields are passed through the deployed model's OpenAI-compatible tool-calling interface and native chat template, with `tool_choice=none`. Internal reasoning is disabled for the Qwen models. We sample one answer per prompt–model pair with temperature 0.7, top-p 0.95, and a maximum of 1,024 output tokens.

For endpoints that do not support the native representation, tool responses are encoded as user turns prefixed with `[Tool output]`, followed by the instruction below.

Fallback final-answer instruction

The required tool has already been called above and its output is shown. Now write the final natural language answer to the user's question in plain English. Do not call any more tools.

We retain only non-empty natural-language final answers. Repeated tool calls, code-like outputs, and role-play continuations are excluded.

B.2 Natural-Response Annotation

The primary judges, `DeepSeek-v4-Flash` and `GLM-4.7`, use the following prompt with temperature 0, a 1,024-token output limit, and JSON-object response formatting. The user query is truncated to 500 characters; the tool output and generated response are each truncated to 2,500 characters.

Natural-response hallucination annotation prompt

You are evaluating an AI assistant's response to a tool-calling query. Your task is to identify hallucinations.

A hallucination is any factual content in the response that is not supported by the tool output. Classify each hallucinated span as one of:

- `answer_mismatch`: the response contradicts or substitutes a value from the tool output;
- `missing_tool`: the response adds factual content not grounded in a tool response;
- `overgeneration`: the response adds unsupported elaboration or commentary;
- `undergeneration`: the response omits information from the tool output; use an empty span list;
- `clean`: no hallucination.

Return JSON only, with the following fields: `is_hallucinated` (boolean), `primary_type`, `spans`, and `notes`. Each element of `spans` must contain an exact substring from the assistant response, its

hallucination type, and a short explanation. If the response is clean, return an empty span list. Do not paraphrase quoted spans.

USER QUERY: `{query}`
TOOL OUTPUT: `{tool_output}`
ASSISTANT RESPONSE: `{response}`

GPT-4o arbitration. GPT-4o receives the same three input fields, hallucination taxonomy, output schema, temperature, and truncation limits as the primary judges. It is applied to all responses labeled as hallucinated by both primary judges and to a random 15% sample of their disagreements. Its decision is produced with the same annotation prompt shown above.

B.3 Synthetic Corruption Prompts

The prompts below belong to the separate synthetic-data construction pipeline. The field-usage judge identifies which values from a tool response are reflected in the reference answer; the subsequent corruption prompts use this information to create synthetic hallucinations. These prompts are not used to annotate the naturally generated responses described above.

Prompt for LLM-Judge

You are a careful evaluator analyzing how a tool response is reflected in an assistant's answer. Your goal is to identify which fields from the tool response were USED (mentioned/surfaced in the answer) versus UNUSED.
You will also judge whether the answer is appropriately selective or whether it omits important information a complete answer should have included.
Definitions:
- USED field: the field's value (or a paraphrase/reformatting of it) appears in the answer, OR the answer directly answers a question with that value.
- UNUSED field: the field exists in the tool response but its content is neither mentioned nor reflected in the answer.
- UNDERGENERATING: the answer skips information that the user's question would naturally expect. For example, if the user asked for "details" and the answer mentions only 1 of 5 returned items, that's undergeneration. But if the user asked "what's the temperature" and the answer gives just the number while skipping metadata like coordinates and timestamps, that's appropriately selective, NOT undergeneration.
Output STRICT JSON with this exact shape: { "used_fields": ["path.to.field1", "path.to.field2"], "unused_fields": ["path.to.field3"], "is_undergenerating": false, "reasoning": "Brief explanation (1-2 sentences)" }
Field paths use dot notation for nested objects (e.g., "results.data.price") and bracket notation for array indices (e.g., "events[0].name"). For arrays of homogeneous items, you may use a wildcard like "events[*].name" if every item is treated the same way in the answer.
Respond with ONLY the JSON object, no markdown fences or other text.

Hallucination prompt

You are a data-generation assistant for an LLM evaluation benchmark.
Your task: given a tool's JSON response, produce a MODIFIED version of it where some values have been changed so the data is factually different.
Rules: 1. Change at least 1 but no more than 3 distinct values (numbers, names, dates, statuses, enums, booleans).
2. Keep the JSON structure, keys, and schema IDENTICAL to the original.
3. Changed values must be the same type and plausible for the field.
4. Do NOT add or remove any keys or array items.
5. The changes should be subtle — not obviously absurd.
TARGETING (when target fields are specified): If a list of TARGET FIELDS is provided, prefer to change values at those field paths. These are the fields most likely to appear in the assistant's answer, so changing them produces visible hallucinations. You may still make changes outside the target list if it improves realism, but ALWAYS modify at least one target field if any are given.
Respond with ONLY the modified JSON, no explanations or markdown fences.

Overgeneration prompt

You are a data-generation assistant for an LLM evaluation benchmark.
Your task: given a user question, a tool call, and the tool's JSON response, generate an assistant reply to the LAST user question that correctly states the tool results but also adds extra information NOT present in the tool response.
If conversation history is provided, treat it as background only.
Rules: 1. First, faithfully convey the key data from the tool response.
2. Then add plausible-sounding but UNSUPPORTED details: historical context, general knowledge, tips, comparisons, or elaborations.
3. Do NOT contradict any values in the tool response.
4. Do NOT reference or offer to use any other tools or capabilities.
5. Keep the added content to 1-3 extra sentences.
6. Try to insert these details into different parts of the original sentence, not just at the end.
CRITICAL FORMATTING RULE: Wrap ALL added/unsupported content in <OVER> tags. Content that faithfully represents the tool response should NOT be wrapped.
Example: Tool response: {"weather": "sunny", "temp": "25°C"} Your output: <OVER> This is typical for this region during summer.</OVER> The weather is sunny with a temperature of 25°C. <OVER>It is a great time to visit outdoor attractions.</OVER>
Respond with ONLY the annotated assistant message

Missing tool prompt

You are a data-generation assistant for an LLM evaluation benchmark.
Your task: given a user question, a tool call, and the tool's JSON response, generate an assistant reply to the LAST user question that correctly states the tool results but then offers to perform an action using a tool that does NOT exist.
If conversation history is provided, treat it as background only.
CRITICAL: If an "ORIGINAL SYSTEM PROMPT" is provided, it lists the tools that DO exist in this system. Your fabricated capability MUST be a tool or service NOT listed there. Do NOT invent capabilities that match any existing tool's purpose — pick something genuinely outside the available toolset (booking, scheduling, sending messages, executing transactions, generating media, etc., as long as no listed tool covers it).
Rules:
1. First, faithfully convey the key data from the tool response.
2. Then suggest an additional action requiring a DIFFERENT, non-existent tool or service. Make the offer sound natural and related to context.
3. Do NOT contradict any values from the tool response.
4. Do NOT add unsupported factual claims about the tool's data.
CRITICAL FORMATTING RULE: Wrap ALL references to non-existent capabilities in <MISSING> tags.
Example: Available tools: Weather_API Tool response: "weather": "sunny" Your output: The weather is sunny today. <MISSING>Would you like me to book a restaurant with outdoor seating for you?</MISSING>
Respond with ONLY the annotated assistant message.

Undergeneration prompt

You are a data-generation assistant for an LLM evaluation benchmark.
Your task: given a user question, a tool call, and the tool's JSON response, generate an assistant reply that omits significant RELEVANT information from the tool response while appearing complete.
KEY DISTINCTION: A good answer is naturally SELECTIVE — it skips metadata and irrelevant fields. That is NOT undergeneration. Undergeneration means dropping information the user actually wanted or that's central to answering their question. Examples:
Good selectivity (NOT undergeneration): User: "what's the temperature?" → answer skips coordinates, timestamp. Undergeneration: User: "list all events" → answer skips half the events. User: "weather forecast for tomorrow" → answer gives temperature but skips precipitation and wind which the forecast includes. User: "give me details on this product" → answer skips price.
Rules:
1. Identify what fields are RELEVANT to the user's question.
2. Drop at least one of those relevant fields or items, without acknowledging the omission.
3. If the tool returns a list of items the user asked to enumerate, drop at least one entire item.
4. Do NOT contradict any values you DO include.
5. Do NOT add information beyond what the tool returned.
6. Do NOT reference or offer to use any other tools.
7. The response should read as if it's complete — don't say "here are some" or hint that information was left out.
Additionally, output a JSON block at the END of your response listing exactly what was omitted that should have been included, in this format:
<OMITTED> "omitted_items": ["description of each relevant piece of information that was dropped"] </OMITTED>
Only list items that a complete answer to the user's question SHOULD contain. Do not list metadata that would naturally be omitted.
Example: User: "Find me upcoming AI events" Tool response: {"events": ["id": 1, "name": "A", "date": "Jan 1"}, {"id": 2, "name": "B", "date": "Feb 2"}] Your output: Here is the upcoming AI event:
1. **A** — January 1st
<OMITTED> {"omitted_items": ["Event B (id: 2, date: Feb 2) was completely dropped — user asked for events plural"]} </OMITTED>
Respond with ONLY the assistant message followed by the OMITTED block.

C GPT-4o Evaluation Prompts

This section describes the prompts used to evaluate GPT-4o as a hallucination detector in the zero-shot and few-shot settings.

C.1 System Prompt

The following system prompt is shared by both the zero-shot and few-shot configurations. In the few-shot setting, five in-context examples (one per hallucination type) are appended as user/assistant message pairs before the test instance.

System prompt for hallucination detection

You are an expert evaluator of tool-calling dialogues for hallucination detection.
You will be given a dialogue that includes: - A system prompt describing available tools - A conversation between a user, an assistant (which may call tools), and tool responses - The final assistant answer to evaluate
Your task is to detect hallucinations in the FINAL assistant answer and classify the row.

Hallucination type definitions
- clean: The final answer is fully grounded in the tool output. No hallucinated content.
- answer_mismatch: The answer directly CONTRADICTS or SUBSTITUTES values from the tool response. For example: the tool returned "Gonzaga" but the answer says "Miami"; the tool returned year 2022 but the answer says 2023. The tool response IS present – the assistant misread or swapped its data. SPAN RULE: mark ONLY the wrong value itself (e.g. "Miami", "2023", "7.5") – NOT the words before it (not "the team is", not " is", not "SD: "). The span must be the minimal substituted token.
- missing_tool: The answer contains fabricated concrete facts (IDs, names, amounts, dates) that are NOT present in or derivable from the tool response (Baseless Info); OR appends an unsolicited proactive offer / question not grounded in any tool result or user request (Proactive Follow-up, e.g. "I can help you with more actions if you'd like", "I'll go ahead and...", "Need a ... ?"). A tool response IS always present. SPAN RULE for Proactive Follow-up: the hallucinated span is almost always the FINAL sentence(s) of the answer – the unsolicited offer or question appended at the end. Look at the very end of final_answer first. Do NOT mark content from the middle of the answer unless it contains fabricated facts (Baseless Info) that are absent from the tool response.
- overgeneration: The answer adds unsupported elaboration, commentary, interpretation, or claims that go beyond what the tool returned, without directly contradicting it.
- undergeneration: The answer OMITS information that the tool returned. It is an absence, not a fabrication. For this type, span_labels MUST be empty [].

Output format
Respond with valid JSON and NOTHING else. Schema:
{ "type": "<>", "span_labels": [{ "start": <int, char offset in final_answer>, "end": <int, char offset in final_answer, exclusive>, "text": "<exact substring from final_answer>" }], "reasoning": "<one or two sentence explanation>" }
Rules for span_labels: - Provide spans ONLY for types: answer_mismatch, missing_tool, overgeneration. - For clean and undergeneration the list MUST be empty []. - start and end are character offsets into final_answer (0-indexed, end is exclusive). - text must be the exact substring final_answer[start:end]. - Mark only the MINIMAL hallucinated text – do not over-span: * answer_mismatch: mark only the wrong token/value, not surrounding words. * missing_tool (Proactive Follow-up): mark the final unsolicited sentence(s) at the END. - You may return multiple spans if there are multiple hallucinated regions.

C.2 User Message Template

Each test instance is formatted as follows and passed as the user message:

User message template

Evaluate this dialogue for hallucinations.
System prompt: <system_prompt>
Conversation: [user]: <user_message> [assistant]: <assistant_tool_calls> [tool]: <tool_responses> [assistant]: <final_assistant_answer>
final_answer: "<final_assistant_answer>"

C.3 Few-Shot In-Context Examples

In the few-shot setting, five user/assistant example pairs (one per hallucination type) are inserted between the system prompt and the test instance. Table 6 summarizes the examples used.

#	Type	Scenario
1	`clean`	Web page to PDF conversion; all values grounded.
2	`answer_mismatch`	Final Four teams; year and team name substituted.
3	`missing_tool`	Token balances; unsolicited portfolio report offer appended.
4	`overgeneration`	Projectile motion; editorial commentary about wind resistance added.
5	`undergeneration`	Football events count; specific number "15" omitted from answer.

Table 6: Summary of few-shot in-context examples for GPT-4o evaluation.