Tools Fail: Detecting Silent Errors in Faulty Tools

Jimin Sun[1*] **So Yeon Min**[2] **Yingshan Chang**[2] **Yonatan Bisk**[2]

[1]CohereAI [2]Carnegie Mellon University

jimin@cohere.com

Abstract

Tools have become a mainstay of LLMs, allowing them to retrieve knowledge not in their weights, to perform tasks on the web, and even to control robots. However, most ontologies and surveys of tool-use have assumed the core challenge for LLMs is choosing the tool. Instead, we introduce a framework for tools more broadly which guides us to explore a model's ability to detect "silent" tool errors, and reflect on how to plan. This more directly aligns with the increasingly popular use of models as tools. We provide an initial approach to failure recovery with promising results both on a controlled calculator setting and embodied agent planning.[1]

1 Introduction

Tools offer a convenient way to augment capabilities beyond text-based reasoning, from executing code to incorporating recent data through web search, and even facilitating multimodal interactions. While the term "tool" is often interpreted to mean offloading specific deterministic functions to external APIs, as tasks grow more complex, the definition is expanding to include learned modules such as translators and object detectors, as well as heuristics-based policies like search algorithms and robotic skills. LLMs themselves are also being used as tools, particularly as task planners in robotics, chained with vision models and robot policies to perform navigation and manipulation (Ahn et al., 2022; Huang et al., 2022a,b; Liang et al., 2022; Singh et al., 2022a; Li et al., 2023; Xu et al., 2023; Zeng et al., 2023).

As tools take on more responsibilities, assessing and ensuring their reliability becomes crucial; a failure in one tool can trigger a cascade of errors, leading to complete task failure. Recent studies have suggested recovery mechanisms, such as correcting inputs based on API error messages (Pan et al., 2023a; Zhang et al., 2023; Chen et al., 2023b; Pan et al., 2023b). However, most methods rely on two underlying assumptions: that accurate inputs guarantee flawless outputs, and that errors are accompanied by explicit signals. Yet, real-world scenarios challenge the premises, as failures often arise from unpredictable environmental dynamics and inherent inaccuracies of tools themselves.

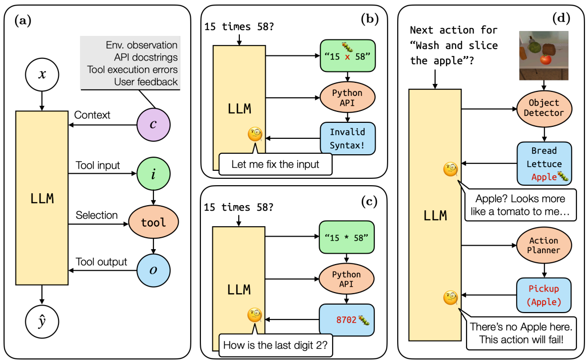

Figure 1: **(a) Tool-use Overview**: Starting from an input x, the LLM generates inputs i for the selected tool, and incorporates tool outputs o to predict the final task output $\hat{y}$. The context c is used throughout the task. **(b) Correct Calculator** Incorrect tool inputs from the LLM leads to tool failure. The error messages can be leveraged for correction (Refine). **(c) Broken Calculator** Tool inputs are correct, but the tool itself silently produces false outputs. **(d) ALFRED** The first tool, Object Detector, misidentifies the Tomato in the image as an Apple, leading to error cascades in the next tool, the Action Planner.

This paper introduces a taxonomy to categorize sources of tool-related errors and recovery methods. We shed light on the often overlooked case: "tool-based" failures. As opposed to input-based errors which are often accompanied by error messages, most tool failures are "silent." This poses unique reasoning challenges for the LLM, which must actively 1. detect the failure, 2. infer the source, and 3. plan recovery strategies. In this paper, we focus on the first step, detection, as it is the prerequisite for downstream fault assignment and recovery.

We investigate tool errors in two distinct set-

[1]Code and data are released: https://github.com/jiminsun/tools-fail

*Work done while at Carnegie Mellon University.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 14272–14289
November 12-16, 2024

tings (Fig. 1) – a controlled environment where an LLM solves arithmetic problems using a broken calculator, and a more natural "broken" tool setting involving a multimodal instruction-following agent. We investigate whether LLMs can detect incorrect tool outputs without explicit error signals, to observe overtrusting of tools. Motivated by how humans detect tool failures based on internal expectations of correct outputs, we devise three in-context interventions, and find that LLMs *can* learn to doubt tools and detect mistakes. Following the taxonomy, we further examine how much and what type of deviation is necessary to trigger the LLM's recognition of the tool error in each setting.

2 Related Work

Tools Text-based tools help compensate for LLMs' relative weakness in world knowledge and computational precision (Lewis et al., 2020; Parisi et al., 2022; Gao et al., 2023; Schick et al., 2023; Yao et al., 2023). Multimodal tools allow LLMs to receive inputs from other modalities and generate grounded answers (Gupta and Kembhavi, 2023; Wu et al., 2023; Yang et al., 2023; Zeng et al., 2023). Outputs of Vision-Language models (Radford et al., 2021), Object Detectors, OCR models, and speech-to-text APIs (Zeng et al., 2023) have been added to the LLM's prompt, enabling zero-shot inference on multimodal tasks.

Agents Research on LLM agents spans multi-step tasks in gaming (Wang et al., 2023a; Wu et al., 2024), web navigation (Qin et al., 2023; Shinn et al., 2023; Yao et al., 2023), and code generation (Shinn et al., 2023; Yao et al., 2023). Most focus on the selection and utilization of tools (Wang et al., 2023a; Qin et al., 2023; Wu et al., 2024), and enhancing reasoning through self-evaluation and feedback (Shinn et al., 2023; Wang et al., 2023a; Chen et al., 2023a; Xu et al., 2023; Madaan et al., 2024).

Adapting LLMs to tool-use Existing works use in-context learning (ICL) (Lu et al., 2023; Shen et al., 2024), finetuning (Schick et al., 2023), and trial-and-error (Wang et al., 2024) to adapt LLM to tool-use. However, the focus has been on adapting to "newer" tools, from demonstrations or documentations, and the question of tool reliability and recovering from "unreliable" tools has not been actively investigated. While malfunctioning APIs are preemptively filtered out in API-centric environments (Qin et al., 2023), the strategies for addressing ineffective learned tools, as in games (Wang et al., 2023a; Wu et al., 2024) or multimodal tasks (Zeng et al., 2022), have been less explored. Overall, existing approaches tend to amalgamate various tool failure modes under the umbrella term "reasoning," focusing primarily on the most salient aspect of failure within their specific domain. In contrast, we distinctly identify and thoroughly analyze errors related to tool arguments, the tools themselves, and the alignment with environmental dynamics.

3 Background

Notation We outline a typical tool-use scenario in Fig. 1a with the following notation:

x : task input	i : tool input
$\hat{y}$: predicted task output	o : tool output
c : context information	t_θ : tool

The LLM first selects tools and constructs tool-specific arguments i from the task input x. Based on the tool result o, the final task prediction $\hat{y}$ is made. Notably, the flexibility of LLMs as an interface allows tool inputs to be constructed based on enriched context information c throughout the task. c may include task specifics, API docstrings, any external feedback like error messages, or even previous action trajectories in interactive tasks.

Additionally, we denote the oracle values of the input, output, context as i^*, o^*, and c^*. The tool input i and output o may contain inaccuracies since they are essentially outputs of preceding LLM/-tool calls. Fig. 1b demonstrates a scenario where i contains a mistake (`15 x 58` should be `15 * 58`). The context c can also be incomprehensive or noisy, as they are approximations of the real world. Moreover, the tool t_θ can be suboptimal in multiple dimensions. For deterministic APIs, a suboptimal tool may have been chosen by an LLM (Schick et al., 2023). For learned tools, the tool itself is an inherently imperfect parameterized model, thus t_θ.

Defining Error The suboptimality of i, c, and t_θ manifest as suboptimal tool outputs o, that deviate from o^*. The deviation can be as critical and explicit, leading to error messages in Fig. 1b, or weakly wrong like the Object Detector output in Fig. 1d. In fact, the severity of a tool error depends on how critically the mistake impacts downstream task performance. In Fig. 1d, the Object Detector misidentifying the `Tomato` as an `Apple`, is crucial to the task, but mistaking objects like `Bread` would not hinder the task as much. As the high-level goal is task success rather than perfect tool utilization, it is important to rectify critical mistakes, whereas harmless mistakes can be disregarded.

To formalize this notion of "task-critical" tool-use mistakes, we introduce an error threshold ϵ to define a range of tool outputs that are not "critically" wrong. Intervention is only necessary when the deviation between the tool output and the oracle, $d(o, o^*)$, is larger than ϵ, thereby degrading the performance/quality of the final task output $\hat{y}$.

$$d(o, o^*) > \epsilon \implies s_{\text{task}}(\hat{y}|o) < s_{\text{task}}(\hat{y}|o^*) \quad (1)$$
$$\text{where } s_{\text{task}} := \text{task performance metric}$$

This is analogous to how humans approach errors; the goal is not a perfect world model but to accomplish a task. As long as we can grab the apple, we do not need to know its exact shape or coordinates.

4 Error sources

The tool output o is accurate if and only if:

1. The context c is correct and sufficient.
2. The tool inputs i are accurate.
3. The tool t_θ makes correct predictions.

Formally, to obtain o with deviation smaller than ϵ, $d(o, o^*)$, is a union of component error bounds:

$$d(o, o^*) < \epsilon \quad (2)$$
$$\Leftarrow \underbrace{d(c, c^*) < \epsilon_c}_{\text{context}} \wedge \underbrace{d(i, i^*) < \epsilon_i}_{\text{tool input}} \wedge \underbrace{d(t_\theta, t_{\theta^*}) < \epsilon_t}_{\text{tool correctness}}$$

If any condition above is not met, output errors will lead to task failure. The following sections discuss each condition, and a table of corresponding real-world error scenarios is presented in App. A.

4.1 Context: $d(c, c^*) > \epsilon_c$

LLMs are employed to choose tools and generate tool inputs, based on context information represented in natural language. However, context information that fits into the prompt is often an impoverished textual approximation of all of the information needed (e.g., API docstrings, few-shot examples, world knowledge) to construct perfect tool inputs. Even for LLMs with human-level reasoning capability, tool proficiency is bottlenecked if the provided context is insufficient. In interactive task settings, this is often inevitable earlier in the planning trajectory, due to partial observability of the surrounding environment. For instance, a web agent might need to scroll through the page and explore hyperlinks. Similarly, an embodied agent may need to explore hidden objects in closed receptacles through trial-and-error, in order to obtain enough information pertinent to the task.

4.2 Input: $d(i, i^*) > \epsilon_i$

Even when the provided context is sufficient, LLMs are prone to generate incorrect tool inputs. Imperfect tool inputs often result from incorrect outputs from a prior tool, like errors in LLM-generated code or noisy images. For deterministic tools (e.g., code interpreters), most errors are due to tool inputs, and malformed inputs typically trigger an error message. However, well-formed inputs with incorrect content (e.g., ambiguous queries for search APIs) can produce erroneous outputs that inadvertently propagate through subsequent steps.

4.3 Tool: $d(t_\theta, t_{\theta^*}) > \epsilon_t$

Tools themselves can make mistakes, even when the input or context is perfect. This situation is especially prominent as learnable tools are becoming more widely adopted in practice. LLMs are prone to generating factually incorrect statements even when reference documents are provided through context (Krishna et al., 2024). Search APIs might fail not because of the input query's clarity, but due to an imperfect database/dense retrieval method. The tool's precision can also contribute to failure – heuristic-based search/manipulation robot policies can fall apart when they lack the precision needed to address the complexity of real-world scenarios.

Due to the absence of explicit error signals, tool-based errors require the tool-using model to reason over indirect cues. In easier cases, errors can be recognized based on well-calibrated confidence scores. Much harder cases, however, arise when a tool confidently produces errors. In such scenarios, a broader context may help identify these hidden errors. Multiple tools presenting conflicting evidence (e.g., fact verification tool vs search API), disagreement between different modalities (Lee et al., 2021), or prediction inconsistencies over multiple trials (Kadavath et al., 2022; Wang et al., 2023c) or timesteps (Chaplot et al., 2020), may help surface potential limitations of the tool.

5 Recovery behaviors

Next, we categorize current recovery methods from previous literature into two groups: **Refine** and **Re-**

place, and advocate for meta-cognitive reasoning.

5.1 Refine: $i \to i^*, c \to c^*$

Recovering from tool failures often involves refining the tool input. This is particularly effective when the failure is followed by explicit feedback signals that indicate "what" to fix. Inputs can be rewritten guided by API error messages and human/LLM feedback (Madaan et al., 2023; Shinn et al., 2023; Wang et al., 2023b). In the planning literature (e.g., TAMP (Garrett et al., 2021; Ding et al., 2023)), this is referred to as "closed-loop planning," where plans are continuously updated by new observations, task progress, or clarification questions (Huang et al., 2022b; Singh et al., 2022a; Song et al., 2022). Augmenting the context based on increased observability changes the input's interpretation. Refine methods are well-suited to LLMs as they can flexibly accept varying lengths of text-based feedback. In contrast, corrections to other modalities (e.g. image lighting or non-verbal communication) remain open challenges for VLMs.

5.2 Replace: $t_\theta \to t_{\theta^*}$

When errors originate from the tool itself, the aim is to move t_θ closer to t_{θ^*}, aligning it more closely with the final task. Mitigation strategies vary based on how easily the tool can be fixed at inference time. For LLMs, in-context examples are used to elicit specific task capabilities from more generic reasoning abilities, a method further enhanced by retrieving samples that are more pertinent to the specific test example (Rubin et al., 2022; Song et al., 2022). Ensembles over multiple predictions also offer a non-invasive way to improve tool performance (Anil et al., 2023; Wang et al., 2023c; Chen et al., 2024). Test-time adaptation methods (Wang et al., 2021) can be useful, though application requires access to the tool's internal parameters. The aforementioned strategies focus on improving the tool's performance in isolation, which may not translate to better task performance. In Fig. 1d, better ImageNet performance does not guarantee detecting the Tomato. Understanding the interplay between tools and task performance remains an open question of system dynamics and credit assignment.

When improving the tool is not viable or when adjustments are insufficient, the best strategy can be to switch to a different tool. Research on assistance-seeking agents implicitly model this behavior, with agents identifying when to delegate the action to a human/oracle (Singh et al., 2022b; Xie et al., 2022). In NLP, Krishna et al. (2024) introduce a fact-checking tool that edits unsupported claims in LLM-generated summaries, advocating for the strategic use of alternative tools to ensure quality and reliability.

5.3 LLMs as a Meta-Reasoner: $\epsilon_i, \epsilon_c, \epsilon_t \uparrow$

For humans, the tools we employ are not perfect. But tools can err because humans can fix incorrect outputs – misrecognized card numbers through an OCR system are corrected ad-hoc by the user. Similarly, imbuing LLMs with the ability to recognize and handle errors flexibly allows for tools to make mistakes, effectively increasing the permissible error thresholds of the tool components $\epsilon_i, \epsilon_c, \epsilon_t$ in Eq. 2. An LLM's meta-cognitive ability to reason over uncertainty and realize its knowledge limits have received some attention (Kadavath et al., 2022; Kuhn et al., 2023). The next step is to jointly reason over their uncertainty/knowledge and that of another tool or agent. This compounds in multi-tool or multi-LM settings. Existing recovery methods that presuppose the cause and tweak a single knob may not yield overall improvement unless limitations of the right variables are resolved.
In summary, we identify three challenges:

1. **Failure Detection**: Recognizing failures and assessing their severity – $d(o, o^*) > \epsilon$?
2. **Fault Assignment**: Identifying which tool caused the error (in multi-tool settings), with the exact source – i, c, or t_θ?
3. **Recovery Planning**: Selecting the most effective recovery strategy. Refine vs Replace?

Explicit error signals (though rare) can obviate all three problems. More importantly, silent tool errors are the opposite case, where even detection is not straightforward although the problem is pervasive. In this work, we delve into "silent" tool errors, a relatively overlooked area in tool-error research, focusing on the foremost problem: error detection.

6 A broken calculator

Humans use tools with a rough expectation of what correct results should look like, allowing them to spot outputs that are obviously wrong. For example, for multiplying 120 by 131, we can expect a result around 10,000 and ending in zero, even if we don't know the exact answer. If the tool makes arithmetic mistakes, can LLMs also detect faulty outputs?

```
# Task
What is the answer to: (2 + 3) * 5?

Refer to the tool output below.
# Calculator API
result = (2 + 3) * 5
result
25  # broken tool setting -> 21 / 205 / -25

# Format
Return your answer in this format:
Thought: Your reasoning process
Answer:
...

# Answer
```

Figure 2: Prompt for a math problem using tool outputs. The result 25 is perturbed in the Broken scenario: Digit replacement, Magnitude shift, or Sign inversion.

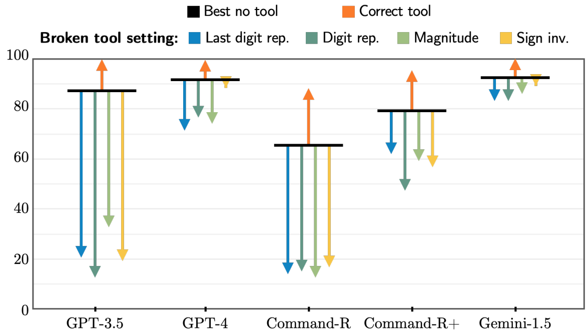

Figure 3: Math accuracy of models. The black bar indicates the best accuracy *without* tool-use; upward orange/downward arrows respectively indicate performance with correct/broken tool-use.

6.1 Task setting

We devise a controlled setting where an LLM answers simple math problems with an external tool, a calculator. In this case, the calculator is broken and returns incorrect outputs.

First, we programmatically generate 300 equations that involve two random operators from $\{+, -, \times\}$ and three random integers (e.g., $9 \times (20 + 7)$). The equations have three levels of difficulty, which is determined by the range that the integers are sampled from: easy $[-20, 20]$, medium $[-100, 100]$, and hard $[-1000, 1000]$. We give the incorrect tool output to the model, and test whether models are able to recognize the error. We compare five different models: GPT-3.5 and GPT-4, Command-R and Command-R+, Gemini-1.5.

6.2 Preliminary experiments

We begin by estimating the models' capabilities to solve math problems on their own, to better understand the downstream implication of having a credible/broken calculator in the loop. Specifically, we query the LLM with five different prompts – three non-tool and two tool-use prompts.

Non-tool setting The non-tool settings serve as a proxy to gauge the model's task capability, providing a basis to compare the effects of incorporating tools with varying levels of credibility. We ask the model to solve the math problems on its own, with three different prompting methods:

1. Direct: Asking the equation directly (e.g., `"What is the answer to (2+3)*5?"`)
2. Chain-of-Thought (CoT): Asking to explain its reasoning step-by-step prior to answering.
3. CoT Few-Shot: In addition to reasoning, the model is provided five in-context examples.

Tool-use setting We assume two types of calculators – Correct and Broken. Fig. 2 shows the tool-use prompt, where the model is asked to answer the question referring to the tool output (**bold**). For Correct tool, the ground truth answer is provided as the tool result. For Broken tool, we give a perturbed answer using one of the following three:

1. Digit replacement: One digit is replaced with a different number (e.g., 25 → 21)
2. Magnitude shift: Digits are inserted/removed, resulting in magnitude shifts in the range 10^{-2} and 10^{3} (e.g., 25 → 205)
3. Sign inversion: The sign is flipped, changing positive numbers to negative and negative numbers to positive (e.g., 25 → −25)

Inspired by Wei et al. (2022); Yao et al. (2023), we specify a "Thought" section, to encourage the model to generate its reasoning prior to answering.

Results We report the preliminary experiment results in App. B and Fig. 3. When the tool is broken, the accuracy drops drastically for all perturbation categories, with the exception of Sign Inversion on GPT-4 and Gemini-1.5. With broken tools, performance drops far below the best no-tool setting's performance, up to 47%. We find that models tend to overtrust tools – copying the incorrect output (with hallucinated justification) rather than ignore the tool in favor of its own answer.

6.3 In-context intervention strategies

Humans leverage various contextual cues like prior tool failures to calibrate the level of trust associated with their tools. Further, AI chatbots include disclaimers like "The model can make mistakes"

Model	ZST Obl.	ZST Disc.	ZST Conf.	ZST Check.	CoT Obl.	CoT Disc.	CoT Conf.	CoT Check.	CoT+FST Obl.	CoT+FST Disc.	CoT+FST Conf.	CoT+FST Check.
GPT-3.5	23	**53**	44	46	46	**81**	79	80	87	**89**	86	84
GPT-4	76	82	**85**	**85**	86	89	89	**91**	90	**91**	88	89
Command-R	16	14	**16**	14	29	42	44	**47**	11	23	**53**	46
Command-R+	57	76	79	**81**	60	**84**	82	76	71	82	**86**	78
Gemini-1.5	84	**90**	76	87	93	**95**	95	90	**94**	**94**	94	94

Table 1: Accuracy of models on math equations with in-context intervention methods against broken tools

Model	ZST Obl.	ZST Disc.	ZST Conf.	ZST Check.	CoT Obl.	CoT Disc.	CoT Conf.	CoT Check.
GPT-3.5	79	**86**	**86**	83	70	67	**83**	75
GPT-4	92	**95**	94	91	96	**97**	96	94
Command-R	62	64	**67**	60	59	68	**80**	71
Command-R+	83	**89**	87	77	73	78	**81**	77
Gemini-1.5	92	94	94	**96**	95	**96**	**96**	89

Table 2: Accuracy of models on the Accept/Reject task on calculator outputs.

to ensure answers are scrutinized. Can LLMs also leverage such information effectively?

We test three types of contextual cues that can raise the awareness towards potential tool mistakes: a simple disclaimer, prediction confidence scores, and a checklist of criteria to look out for. For each method, we evaluate the prediction accuracy on both perturbed and non-perturbed tool outputs, in Zero-shot, CoT, and Few-shot settings. We compare four different prompts:

- **Oblivious (Obl.)** does not mention any indications that the tool can cause errors Fig. 2.

- **Disclaimer (Disc.)** includes a simple disclaimer: *"The tool can sometimes give incorrect answers. Please verify the correctness of the tool output."*

- **Confidence (Conf.)** includes the confidence score of the tool's prediction, in addition to the disclaimer. Since the calculator is not a probabilistic model, we devise a score [0,1] based on the string edit distance between the ground truth and the perturbed output. For learned tools, model confidence can be used.

- **Checklist (Check.)** is motivated by heuristics that humans use, which includes a list of criteria to check the tool output, based on the perturbation. For the math task, the checklist consists of:

  1. Is the positive or negative sign correct?
  2. Is the magnitude of the number correct?
  3. Is the last digit correct?
  4. Are all the digits correct?

Results Table 1 shows how effectively each method helps the LLM notice and correct mistakes. For most models, even a simple disclaimer prevents naively believing perturbed answers, boosting accuracy up to 30%. As humans, LLMs can better detect mistakes when provided the context that tools can be wrong. Chain-of-thought prompting and in-context examples further help models recover performance, nearly to the best no-tool scores.

7 Detecting tool-based mistakes

The results in §6 suggest that it is challenging for LLMs to simultaneously detect and override faulty outputs, even for capabilities that are decently performed without tools. Thus, next we narrow the LLM's responsibility to "detecting" mistakes.[2]

Results The models are often able to identify the incorrect outputs (Table 2) despite not being able to produce the correct answer – even in conditions where they would have without a tool present. Smaller models (GPT-3.5, Command-R) are more sensitive to in-context information. Where in Oblivious, most small model errors are due to overtrusting tools, and with in-context intervention, the prediction skews heavily towards rejecting outputs, leading to high false positive rates. In contrast, errors occur in similar rates for the larger models.

Surprisingly, CoT does not always improve performance over Zero-shot. We find that the majority of CoT errors are the model falsely rejecting correct outputs – caused by failure in faithfully copying the original equation's terms in its reasoning steps. Incorrect reasoning cases are more frequently observed in the CoT setting, contradicting Table 1 where CoT outperformed Zero-shot. While more investigation is needed, we speculate that the effectiveness of CoT might depend on task complexity, because the model is burdened to simultaneously

[2] We reformulate the calculator setting into a binary Accept/Reject task (Fig. 8). We balance the 300 perturbed equations in §6.2 with 300 correct samples to account for false positives.

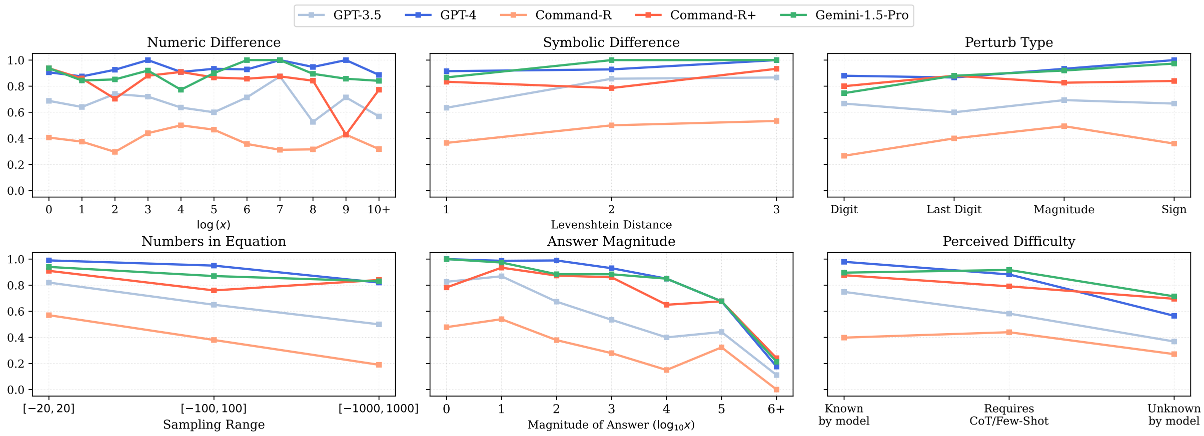

Figure 4: The rejection rate on the perturbed calculator outputs with respect to six features.

1. solve the equation and 2. spot mistakes in the Detection+CoT setting. A two-step process where the LLM first generates its answer, then compares its own answer to tool outputs in a second call may alleviate this issue, which we leave to future work.

7.1 When are mistakes easier to detect?

For humans, whether a mistake is detected might depend on the type of mistake (blatant vs subtle), the difficulty of the original question, or the answerer's task proficiency. Are some mistakes, past a certain level of deviation, just more obvious than others? Does the property of the question matter? Or does it relate to the model's internal knowledge – do you need to "know" the answer to detect errors? In Fig. 4, we analyze the models' rejection rate on the perturbed outputs with respect to six features:

Numeric Difference The absolute difference between the correct and perturbed answer.

Symbolic Difference The string edit (Levenshtein) distance. Smaller symbolic deviations are expected to be less noticeable. Symbolic difference only loosely correlates with numeric differences (ρ = 0.49). For example, 123 to −123 vs 119.

Perturbation Type Digit replacement, Magnitude shift, and Sign inversion from §6.2. We separate last digit replacement as it is easier for humans to detect than other digit positions by mental math.

Magnitude in Equation Equations are binned into three difficulty levels (§6.1), based on the magnitude of numbers involved in the equation. Relatedly, LLMs have been shown to find larger numbers harder to reason over (Nogueira et al., 2021; Lee et al., 2023; An et al., 2023; Duan and Shi, 2024).

Answer Magnitude The magnitude of the correct answer, in log scale ($\log_{10}|x|$). This is similar to "Magnitude in Equation" above, but provides more fine-grained measurements.

Perceived Difficulty This is inferred via the model's ability to answer the equation in §6.2. The categories are: The model (1) answered correctly with a "Direct" prompt, (2) required CoT or Few-Shot examples, and (3) gets the equation wrong even after applying these methods. The number of samples vary for each bin, depending on the model.

Numeric/String Difference and Perturbation Type attribute the rejection rate to the error's "wrongness." Magnitude is associated with the question itself, and Perceived Difficulty targets the model's internal knowledge.

7.2 Analysis

Numeric vs Symbolic Unlike numeric difference, symbolic deviations appear highly correlated with rejection rates. This aligns with literature that LLMs are not performing arithmetic "reasoning," but memorizing strings (Chang and Bisk, 2024).

Perturbation Types For humans, Sign Inversion and Last Digit are likely the easiest to spot. LLMs also find some perturbation types more obvious than others – Sign Inversion for GPT-4 and Gemini, Magnitude for Command-R and GPT-3.5, and Last Digit Replacement for Command-R+. Most models find Last Digit Replacements easier to spot than other digits. Sensitivity is likely attributable to differing representations/tokenization (Nogueira et al., 2021; Liu and Low, 2023).

Large Numbers Models struggle with large values in both Numbers in Equation and Magnitude.

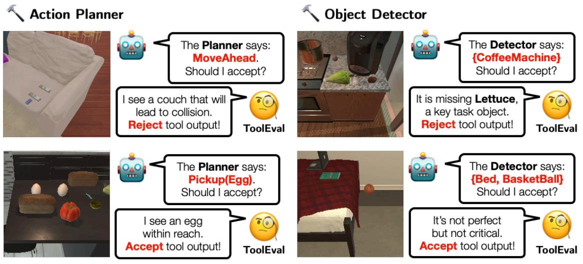

Figure 5: Evaluating two tool outputs in ALFRED – Action Planner (Left) and Object Detector (Right). The LLM is asked whether to Accept/Reject the tool output, based on the provided image and task context.

	VLM	ZST				CoT			
		Obl.	Disc.	Conf.	Check.	Obl.	Disc.	Conf.	Check.
Action Planner	GPT-4o	43	42	40	**44**	57	55	52	<u>**60**</u>
	Gemini	49	55	50	**63**	64	64	62	<u>**65**</u>
Object Detector	GPT-4o	**68**	**68**	66	67	68	<u>**69**</u>	66	68
	Gemini	60	60	56	**62**	<u>**67**</u>	66	65	66

Table 3: F1 score on the Accept/Reject task on two tool outputs in ALFRED. We compare interventions (Disclaimer, Confidence, Checklist) with "Oblivious."

Equations with large numbers can be easier depending on the operations involved. For instance, $(1000-998)\times 2 = 4$ is easier than $10\times 11\times 12 = 1320$. Notably, the rejection rate for answers larger than 10^6 drops sharply for all models.

Perceived Difficulty Problems that are more easily answered by the model, are also more easily detected when exposed to errors. While this might raise a question on the utility of imperfect tools, we find that the larger models (GPT-4, Gemini-1.5-Pro, Command-R+) can "detect" the mistake for the majority of questions, even for ones that it were not able to answer correctly. This sheds light on the feasibility of using LLMs as a tool planner that evaluates the credibility of tools and reroutes functions accordingly to alternative tools. Smaller models, however, tend to overtrust the tool and allow errors to pass.

8 Natural tool errors: ALFRED

We now consider a setting where tool-based errors occur more naturally via ALFRED (Shridhar et al., 2020), an embodied instruction following benchmark. Involving language understanding, perception, spatial reasoning, and action planning capabilities, a common approach is to incorporate multiple specialized modules (Blukis et al., 2022; Min et al., 2022), as opposed to end-to-end training.

Multiple modules, or tools collaborating with each other in ALFRED offer a unique opportunity to study the robustness of LLMs to various tool errors. As in Fig. 1d, the object detector's mistakes are silently passed on to subsequent tools, leading to error cascades in the Action Planner. In such scenarios, LLMs that can detect tool errors help improve the system's robustness, by correcting some obvious semantic anomalies (Elhafsi et al., 2023) or delegating operations to other tools or humans.

In this section, we investigate whether LLMs can detect these realistic, multimodal tool errors arising from individual modules used in the FILM architecture (Min et al., 2022). Specifically, we test the LLM's fault detection capability on two distinct tools – the object detector and the action planner.[3]

8.1 Multimodal tool-error detection dataset

We create a classification task where the model Accept/Rejects the tool output, based on the current context. For the action planner, the model has to assess the feasibility of the predicted action, and reject actions that are to fail (e.g., facing an obstacle for `MoveAhead`, Fig. 5). For the object detector, the LLM evaluates the correctness of the detection results with respect to the image, and reject outputs that mistaken important task objects. We note that imperfect outputs can still be labeled as "Accept" if only containing task-irrelevant errors.

We collect agent trajectories from the ALFRED validation set with actions and API responses whether the action succeeded. For the object detector, we gather RGB images with detection results and groundtruth semantic information. We provide detailed statistics of each dataset in App. C.1.

8.2 Experimental setting

We test tool evaluation accuracy against the two best closed-source Vision-Language Models: GPT-4o and Gemini-1.5-Pro-latest. As in the calculator, we evaluate models on Zero-Shot (ZST) and Chain-of-Thought (CoT) settings. The prompt includes the task state (e.g., current subgoal, steps taken), tool docstrings (e.g., possible actions, object categories), and the current tool output. We provide

[3] Object detection uses a finetuned MaskRCNN model. Action planning is done by the Fast Marching Method (Sethian, 1996), a heuristic-based algorithm.

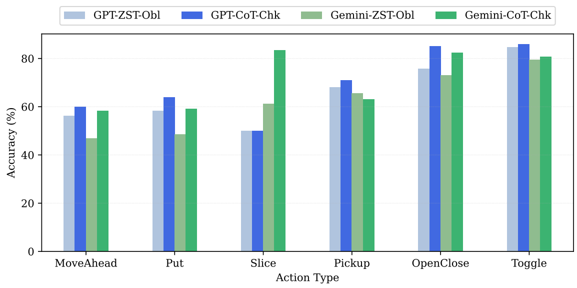

Figure 6: Tool evaluation accuracy on the action planner output binned by action types. We plot the baseline (Zero-shot+Oblivious) with the best performing setting (CoT+Checklist) of the two models.

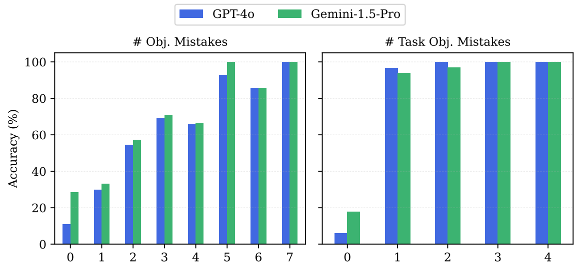

Figure 7: Tool evaluation accuracy on the object detector output binned by the number of detector mistakes on all objects (Left) and task-relevant objects (Right).

example prompts in the Appendix: Action Planner (C.2), Object Detector (C.3).

8.3 Results

Models are able to reach 60-70 F1 scores with raised awareness through in-context learning and CoT prompting (Tab. 3). In particular, specifying the potential failure modes in the Checklist prompt is effective for evaluating the action planner, where the error modes are more diverse than the Object Detector. In contrast, giving the raw confidence scores is not as helpful, as it demands additional interpretation. As these results are all zero-shot evaluations, we expect further improvements in few-shot or finetuning scenarios.

Action Planner In Figure 6, we further analyze the tool evaluation accuracy per different action type. Actions require different preconditions to succeed. Also for assessing feasibility, different actions require varying levels of spatial reasoning, object/scene detection, and task understanding. For MoveAhead, the agent needs to perform spatial reasoning, looking out for obstacles in its path. For interaction actions, more conditions are needed – successful Pickup demands the target to be visible from the agent, located within reachable distance, while the agent's hand is empty.

One might expect navigation actions like MoveAhead to be the easiest to infer feasibility, as it relies mainly on spatial reasoning of obstacles, compared to interaction actions which may demand more preconditions. Somewhat surprisingly, we observe the opposite – because evaluating MoveAhead depends "solely" on spatial information, it is in fact harder to evaluate compared to other interaction actions, as the model has less hints to utilize. For interaction actions, models were able to predict tool success based on objects, which compensated for the LLMs' limited capability in spatial reasoning.

Object Detector In Figure 7, we plot the LLM's evaluation accuracy with respect to the number of mistakes made by the detector, which is one indicator of the deviation, $d(o, o^*)$. We differentiate "crucial" mistakes with less crucial ones, by plotting the number of detection mistakes for all objects (Left) and task-relevant objects (Right) separately. The more mistakes the tool makes, regardless of their task relevance, it is easier for models to reject tool outputs. Both models are also able to spot "task-relevant" mistakes 90% of the time when they occur (#Task Obj Mistakes > 1).

However, models tend to over-reject many acceptable tool outputs even when the mistake is not crucial. While the desired behavior is to accept when the number of task-object mistakes is zero (i.e., no mistakes), models incorrectly reject most outputs (Acc $< 20\%$). Models seem to understand when the tool is wrong, but still struggle with telling apart task-critical vs tolerable tool mistakes, indicating the challenge of relying on in-context learning to steer complex reasoning abilities of LLMs.

9 Conclusion

We characterize the trust dynamics of modern LLMs with respect to tool usage. By establishing an extensive taxonomy of tool-related errors and recovery strategies, we identify fundamental challenges associated with integrating learned tools. Our experiments span both synthetic and natural tool failures, and affirms current LLMs' ability to identify silent tool failures. This work paves the way for future research on harnessing LLMs as sophisticated tool-reasoners.

10 Limitations

This study, while comprehensive in its scope, has certain limitations regarding the diversity and breadth of the models and datasets used. Firstly, for the calculator experiments, we employ five LLMs, mostly closed-source. Including smaller, open-source models, and models specifically fine-tuned for tool-use would have offered more insights into the models' tool trusting behavior. In the experiments involving embodied agents, we limited our focus to only two API-based Vision-Language Models (VLMs). Incorporating smaller, open-source VLMs would have offered opportunities to explore into the models' internal workings, revealing additional nuances in how models handle unreliable tools.

Secondly, the action planner and object detection dataset we constructed based on ALFRED trajectories is fairly small in size – Action Planner (490) and Object Detector (214). In terms of diversity, running multiple models/agents in addition to FILM would have enabled collecting a wider array of failure modes. Moreover, the action's success or failure is highly dependent on the affordances provided by the AI2-THOR framework which may not accurately reflect real-world scenarios. For example, a "Put" action might fail due to the system perceiving a surface as cluttered, even when there is visibly sufficient space available. A dataset encompassing a wider variety of scenarios and higher diversity would potentially provide deeper insights into the practical applications and limitations of current AI systems in navigating real-world environments.

Acknowledgements

We would like to thank the anonymous reviewers, friends at CLAW lab and SLAB for their valuable comments and feedback.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Jayant Joshi, Ryan C. Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego M Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, F. Xia, Ted Xiao, Peng Xu, Sichun Xu, and Mengyuan Yan. 2022. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*.

Jisu An, Junseok Lee, and Gahgene Gweon. 2023. Does chatgpt comprehend the place value in numbers when solving math word problems. In *Proceedings of the Workshop" Towards the Future of AI-augmented Human Tutoring in Math Learning" co-located with The 24th International Conference on Artificial Intelligence in Education (AIED 2023), Tokyo, Japan*, volume 3491, pages 49–58.

Gemini Team Google Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, et al. 2023. Gemini: A family of highly capable multimodal models. *ArXiv*, abs/2312.11805.

Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. 2022. A persistent spatial semantic representation for high-level natural language instruction execution. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 706–717. PMLR.

Yingshan Chang and Yonatan Bisk. 2024. Language models need inductive biases to count inductively. *arXiv preprint arXiv:2405.20131*.

Devendra Singh Chaplot, Helen Jiang, Saurabh Gupta, and Abhinav Gupta. 2020. Semantic curiosity for active visual learning. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part VI*, volume 12351 of *Lecture Notes in Computer Science*, pages 309–326. Springer.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. 2024. Are more llm calls all you need? towards scaling laws of compound inference systems. *Preprint*, arXiv:2403.02419.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. 2023a. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023b. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. 2023. Task and motion planning with large language models for object rearrangement. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2086–2092. IEEE.

Shaoxiong Duan and Yining Shi. 2024. From interpolation to extrapolation: Complete length generalization for arithmetic transformers.

Amine Elhafsi, Rohan Sinha, Christopher Agia, Edward Schmerling, Issa A D Nesnas, and Marco Pavone. 2023. Semantic anomaly detection with large language models. *Auton. Robots*, 47(8):1035–1055.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2021. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14953–14962.

Wenlong Huang, P. Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *ArXiv*, abs/2201.07207.

Wenlong Huang, F. Xia, Ted Xiao, Harris Chan, Jacky Liang, Peter R. Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022b. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zachary Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, John Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom B. Brown, Jack Clark, Nicholas Joseph, Benjamin Mann, Sam McCandlish, Christopher Olah, and Jared Kaplan. 2022. Language models (mostly) know what they know. *ArXiv*, abs/2207.05221.

Kundan Krishna, Sanjana Ramprasad, Prakhar Gupta, Byron C Wallace, Zachary C Lipton, and Jeffrey P Bigham. 2024. Genaudit: Fixing factual errors in language model outputs with evidence. *arXiv preprint arXiv:2402.12566*.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations*.

Michelle A. Lee, Matthew Tan, Yuke Zhu, and Jeannette Bohg. 2021. Detect, reject, correct: Crossmodal compensation of corrupted sensors. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 909–916.

Nayoung Lee, Kartik Sreenivasan, Jason D Lee, Kangwook Lee, and Dimitris Papailiopoulos. 2023. Teaching arithmetic to small transformers. *arXiv preprint arXiv:2307.03381*.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Boyi Li, Philipp Wu, Pieter Abbeel, and Jitendra Malik. 2023. Interactive task planning with language models. *ArXiv*, abs/2310.10645.

Jacky Liang, Wenlong Huang, F. Xia, Peng Xu, Karol Hausman, Brian Ichter, Peter R. Florence, and Andy Zeng. 2022. Code as policies: Language model programs for embodied control. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500.

Tiedong Liu and Bryan Kian Hsiang Low. 2023. Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks. *arXiv preprint arXiv:2305.14201*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

So Yeon Min, Devendra Singh Chaplot, Pradeep Kumar Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. 2022. FILM: following instructions in language with modular methods. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2021. Investigating the limitations of transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023a. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2023b. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *arXiv preprint arXiv:2308.03188*.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning to retrieve prompts for in-context learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2671, Seattle, United States. Association for Computational Linguistics.

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

J A Sethian. 1996. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Neural Information Processing Systems*.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2022a. Progprompt: Generating situated robot task plans using large language models. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530.

Kunal Pratap Singh, Luca Weihs, Alvaro Herrasti, Jonghyun Choi, Aniruddha Kembhavi, and Roozbeh Mottaghi. 2022b. Ask4help: Learning to leverage an expert for embodied tasks. *Advances in Neural Information Processing Systems*, 35:16221–16232.

Chan Hee Song, Jiaman Wu, Clay Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2022. Llm-planner: Few-shot grounded planning for embodied agents with large language models. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2986–2997.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. Llms in the imaginarium: tool learning through simulated trial and error. *arXiv preprint arXiv:2403.04746*.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. 2021. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023b. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *Preprint*, arXiv:2309.10691.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023c. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference* *on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *Preprint*, arXiv:2303.04671.

Yue Wu, So Yeon Min, Shrimai Prabhumoye, Yonatan Bisk, Russ R Salakhutdinov, Amos Azaria, Tom M Mitchell, and Yuanzhi Li. 2024. Spring: Studying papers and reasoning to play games. *Advances in Neural Information Processing Systems*, 36.

Annie Xie, Fahim Tajwar, Archit Sharma, and Chelsea Finn. 2022. When to ask for help: Proactive interventions in autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 35:16918–16930.

Mengdi Xu, Peide Huang, Wenhao Yu, Shiqi Liu, Xilun Zhang, Yaru Niu, Tingnan Zhang, Fei Xia, Jie Tan, and Ding Zhao. 2023. Creative robot tool use with large language models. *Preprint*, arXiv:2310.13065.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. Mm-react: Prompting chatgpt for multimodal reasoning and action. *Preprint*, arXiv:2303.11381.

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

Andy Zeng, Maria Attarian, brian ichter, Krzysztof Marcin Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael S Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. 2023. Socratic models: Composing zero-shot multimodal reasoning with language. In *The Eleventh International Conference on Learning Representations*.

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. 2022. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*.

Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. 2023. Self-edit: Fault-aware code editor for code generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada. Association for Computational Linguistics.

Appendix

A Overview of Tool Errors

In Table 4, we compile a list of tools that support various modalities, with respective real-world tool-error scenarios. We categorize specific error scenarios by its source of failure – the tool input, the tool itself, or context information.

B Math problems

Table 5 reports the accuracy of models on "answering" math equations, plotted in Figure 3. The numbers in the parentheses indicate the relative gain/loss compared to the best no-tool setting (in **bold**). In short, Chain-of-Thought prompting improves arithmetic performance, which is further enhanced by few-shot in-context examples. Correct tool-use yields strongest results, supporting existing literature that employ reliable tools.

We share an example prompt for Accept/Reject task for the calculator setting in Figure 8. This is comparable to Figure 2, where the task inputs are identical, but the primary task is to "answer" the equation rather than "evaluating" the tool output.

C ALFRED

C.1 Dataset

For the dataset used for action planner evaluation, we plot the histogram of actions and task types in Figure 9. For **Action Type** (Left), `Pickup` and `Put` are the most frequent actions, as most task types necessitate these actions for object interaction. `Toggle` and `OpenClose` are merged from the canonical actions `ToggleOn+ToggleOff`, and `OpenObject+CloseObject`, respectively. We note that `ToggleOff` and `CloseObject` was always successful for the FILM agent, as these actions are attempted at the same location where the preconditioning action (`ToggleOn`, `OpenObject`) was successful. Merging the related actions help balance out the Accept/Reject label distribution per action category.

Similarly in Figure 10, we describe object frequencies in the object detector evaluation dataset. Large receptacle objects like `CounterTop` and `Cabinet` are observed the most frequently.

C.2 Action Planner Evaluation

Figure 11 shows an example prompt used for action planner evaluation. The prompt consists of general task instructions, a docstring explaining how the Planner API works, the agent's status on task progress. For the Disclaimer setting, it is informed that the planner can make mistakes. In the Confidence setting, a confidence score is provided alongside the predicted action, which is the success rate of the past five actions. We additionally note that this confidence score may not always align well with tool success rates in this setting, which might be one reason why the Confidence prompt underperforms the Oblivious prompt in Table 3. The Checklist lists the common failure modes of the planner suggested action. The previous four actions and their success/failure is also presented. Our analysis into the reasoning steps of the LLM shows that models are capable of inferring the robot's state based on this information (e.g., `[(Open, Fail), (MoveAhead, Success), (Open, Fail), (MoveAhead, Success)] -> Reasoning: ... the previous attempts suggest that the robot might have been trying to open the microwave from too far away`).

C.3 Object Detector Evaluation

Figure 12 shows an example prompt used for object detector evaluation. Similar to the action planner prompt in Figure 11, general instructions, tool docstring, robot states are given. The robot state here additionally includes the remaining subgoals, as it is helpful in determining which objects are task relevant or not. For instance, while the current subgoal is `('Pickup', 'Apple')`, correcting detection mistakes for `'Microwave'` would be beneficial, as it is needed in future subgoals. For Oblivious, Disclaimer, and Checklist, the tool output is given in a nested dictionary format, where objects are binned into `'detected'` and `'filtered'`, based on the detector threshold. For the Confidence setting, the detection results are provided in a single dictionary, with objects and their respective raw confidence scores. The instruction mentions that objects with score below 60 will be filtered out. Based on the raw scores, the LLM has to interpret whether specific objects will be kept or discarded.

Modality	Capability	Tool	Source of failure		
			Tool input	**Tool itself**	**Context**
Text	Mathematical computation	Calculator Code interpreter	- API syntax error - Incorrect content	NA	NA
	Code validation	Code interpreter	- Code syntax error - Version updates (e.g., deprecated functions) - Incorrect content	NA	NA
	World knowledge	Search API	- Ambiguous query	- Incomplete DB - Irrelevant results (e.g., different word sense)	
	Task planning	LLM/VLM	- Prompt includes non-existent objects due to previous perception errors	- API call failure - Plan includes unsupported actions/objects - Incorrect steps	- Invalid plan due to partial observability (e.g., closed receptacles)
Image	Text recognition	OCR model	- Blurry/noisy image	- Parsing mistakes	
	Visual perception	Vision-Language Models (CLIP) Semantic segmentation (Fast-RCNN) Object detectors (M-DETR)	- Camera noise - Poor lighting	- Unknown object - Detection failure - Hallucination - Wrong categories - Bad segmentation mask	
		Depth estimators		- Estimation errors	
Sensory Perception	Pose Estimation, Map building	SLAM	- Sensor drift	- Algorithmic errors	- Environmental interference (e.g. moving humans, key object change)
Audio	Auditory perception	Speech-to-text API (Socratic Models)	- Audio noise	- Recognition errors	
Action	Navigation	Path-planning algorithms (A*, Fast Marching Method)		- Collision - Circling with no progress	- Change in obstacle locations
	Manipulation	Skills		- Grip failure	

Table 4: **Overview of Tool Errors.** API syntax errors are a shared case of input-based failures across tools. Similarly, network issues are shared across tools as environmental failures.

```
# Task
You are given the equation: (2 + 3) * 5. The task is to
evaluate the result of the equation provided by the tool.

Refer to the tool output below.
# Calculator API
result = (2 + 3) * 5
result
-25 # broken tool setting -> 21 / 205 / -25

# Format
Return your answer in this format:
Thought: Your reasoning process
Evaluation: Accept/Reject
...

# Answer
```

Figure 8: Example Accept/Reject prompt for the output of the calculator. The modified Fig. 2 instructions are in **bold**. We color-code the three perturbation methods as: Digit replacement, Magnitude shift, Sign inversion.

Model	Direct	CoT	CoT-FS	Correct tool	Broken tool
GPT-3.5	61.0	79.7	**85.3**	98.7 (+13.4)	22.7 (-62.6)
GPT-4	64.0	89.0	**89.7**	97.7 (+8.0)	76.0 (-13.7)
Command-R	34.3	52.3	**63.3**	86.3 (+23.0)	16.0 (-47.3)
Command-R+	62.0	75.7	**77.3**	93.7 (+16.4)	56.7 (-20.6)
Gemini-1.5	86.7	**90.3**	88.7	98.3 (+8.0)	83.7 (-6.6)

Table 5: Average accuracy of models on math equations based on various prompting methods.

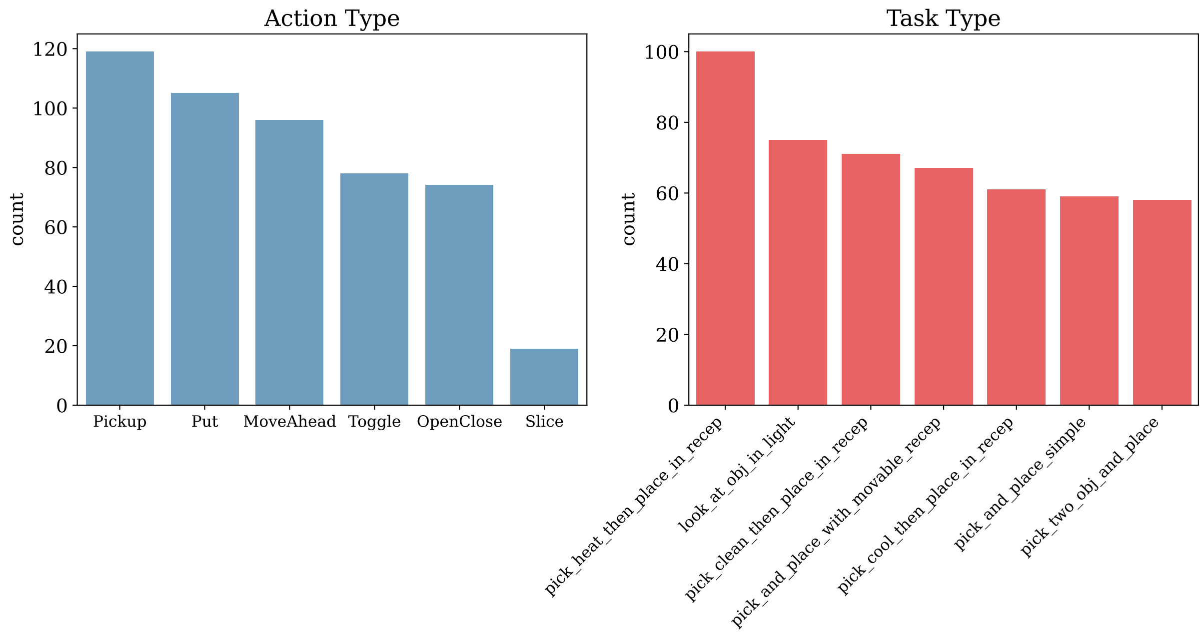

Figure 9: Histogram of actions (left) and task types (right) in the dataset

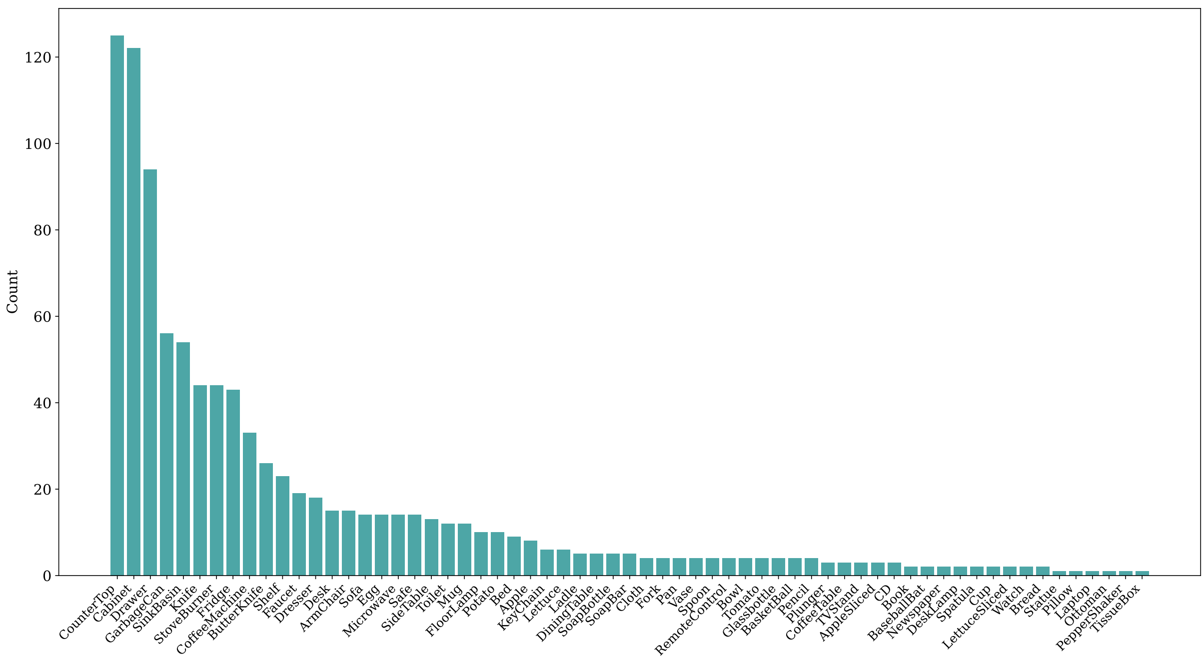

Figure 10: Histogram of objects appearing in all scenes in the dataset

```
A robot is working on household tasks in a simulator environment. The robot follows a series of low-level actions to
accomplish the task. The robot uses an external tool, a low-level action planner, which predicts the next action to follow.
 The provided image is a first-person view from the robot's perspective. Refer to the tool suggested action below and
decide whether to accept or reject the tool output, based on your judgement of whether the action would succeed/fail.

The tool can sometimes give incorrect answers. Please cross-check the output based on the image and robot state, to verify the correctness
and feasibility of the planner's output.
The tool's prediction confidence (between 0 and 1) is also provided, which may hint the correctness of the output. Confidence is based on previous
action attempts and success/failure.

The following are some scenarios where the Planner action might fail.
1. Interaction actions might fail if the object is too far from you. In this case, you need to approach closer to the object.
2. Interaction actions might fail when you do not have a good view of the object.
3. If another object is in your path, MoveAhead will fail due to collision. In this case, you need to walk around the obstacle.
4. If a receptacle is occupied with another object, Put will fail.

# Tool: Planner API
The Planner API provides a function that takes the task_state, observed_state as input and returns the next suggested
action. The action is computed based on the agent and target object's location, based on the robot's internal spatial map.

## Task
possible_actions = ['MoveAhead', 'Open(Receptacle)', 'Close(Receptacle)', 'Pickup(Object)', 'Put(Object, Receptacle)', '
ToggleOn(Object)', 'ToggleOff(Object)', 'Slice(Object)']

## Robot state
task_state = {
    'task_description': "Pick up a pillow and turn a lamp on.",
    'completed_subgoals': [],
    'current_subgoal': "Pickup Pillow",
    'num_steps_taken': 56
}
print(observed_state)
Current room has: Bed, Pillow on a Bed, Cabinet, Drawer, Dresser, GarbageCan, Shelf, SideTable, Sofa, Pillow on a Sofa.
Previous action attempts: [(MoveAhead, Success), (MoveAhead, Success), (MoveAhead, Success), (MoveAhead, Success)]

## Planner output at current step
output = Planner(task_state, observed_state)
print(output)
Pickup(Pillow), 0.8

# Format
Return your answer in this format:
Tool output: [ACTION]
Thought: Your reasoning process
Evaluation: Accept/Reject

The evaluation is a single word indicating whether you accept or reject the tool output. Do not provide any reasoning in
the evaluation. Provide your reasoning in the thought section.

# Answer
```

Figure 11: **Example Prompt for Planner Error Detection** The model is provided instructions to evaluate the output of the Planner and decide whether to Accept or Reject. We denote the instructions specific to the different types of in-context interventions as Disclaimer, Confidence, and Checklist).

```
A robot is working on household tasks in a simulator environment. The provided image is a first-person view from the robot'
s perspective. The robot uses an external tool, an object detector to identify which objects are in the current scene.
Refer to the tool output below and evaluate the correctness of the detector with respect to the provided image, and decide
whether to accept or reject the tool output. If objects important to the task are ignored by the detector, the tool output
should be rejected. Mistakes with regard to task-irrelevant mistakes are acceptable.

The tool can sometimes give incorrect answers. Please cross-check the output based on the image and robot state, to verify the correctness of the
detector's output.
The tool's prediction confidence (between 0 and 100) is also provided, which may hint the correctness of the output. Keep in mind that objects with
confidence scores below 60 will be ignored.

The following are common examples where the detector mistakes may hinder the robot's ability to accomplish the task. Consider these cases in your
reasoning steps.
1. Missing task-relevant objects in the scene. In particular, small objects (e.g., keys, credit card) are prone to be missed.
2. Hallucinating task-relevant objects that are not in the scene. For example, objects that are similar in shape or color (e.g., apple vs tomato) may
be mistaken.

# Tool: Object Detector API
The Detector API provides a function that takes the current_image as input and returns the list of objects detected in the
image. The obj_categories and receptacles are predefined as below. The prediction consists of two parts: the predicted
objects and the filtered objects. The 'filtered' objects are object detections ignored as the detection confidence was
lower than the threshold. Only the 'detected' objects will be passed on.

Detector.obj_categories = ['AlarmClock', 'Apple', 'AppleSliced', 'BaseballBat', 'BasketBall', 'Book', 'Bowl', 'Box', 'Bread
', 'BreadSliced', 'ButterKnife', 'CD', 'Candle', 'CellPhone', ... ]
Detector.receptacles = ['ArmChair', 'BathtubBasin', 'Bed', 'Cabinet', 'Cart', 'CoffeeMachine', 'CoffeeTable', 'CounterTop',
 'Desk', 'DiningTable', 'Drawer', 'Dresser', 'Fridge', ... ]

## Robot state
task_state = {
    'task_description': "Place a cooked apple into the sink.",
    'completed_subgoals': [('Pickup', 'Apple')],
    'remaining_subgoals': [('Open', 'Microwave'), ('Put', 'Microwave'), ('Close', 'Microwave'), ('ToggleOn', 'Microwave'),
('ToggleOff', 'Microwave'), ('Open', 'Microwave'), ('Pickup', 'Apple'), ('Close', 'Microwave'), ('Put', 'SinkBasin')],
    'num_steps_taken': 235
}

## Detector output on current image
Detector(current_image)
# {'Apple': 3.09, 'Knife': 0.55, 'CounterTop': 63.31, 'DiningTable': 47.09} for Confidence
# other prompting methods:
{
    'detected': {'CounterTop'},
    'filtered': {'DiningTable', 'Apple', 'Knife'}
}

# Format
Return your answer in this format:
Thought: Your reasoning process on the provided information (image, task_state and tool_output)
Evaluation: Accept/Reject

The evaluation is a single word indicating whether you accept or reject the tool output. Do not provide any reasoning in
the evaluation. Provide your reasoning in the thought section.

# Answer
```

Figure 12: **Example Prompt for Object Detector Error Detection** The model is provided instructions to evaluate the output of the Object Detector and decide whether to Accept or Reject. We denote the instructions specific to the different types of in-context interventions as Disclaimer, Confidence, and Checklist.