

# AI Agents That Matter

Sayash Kapoor\*, Benedikt Strobel\*, Zachary S. Siegel, Nitya Nadgir, Arvind Narayanan  
Princeton University  
July 2, 2024

## Abstract

All agents are an exciting new research direction, and agent development is driven by benchmarks. Our analysis of current agent benchmarks and evaluation methods reveals several challenges that must be overcome for downstream applications. First, there is a narrow focus on accuracy without attention to other metrics. As a result, SOTA agents are necessarily complex and costly, and the community has reached a point where the cost of maintaining them is prohibitive. One focus on cost in addition to accuracy motivates the need of jointly optimizing the two metrics. We design and implement one such optimization, showing its potential to provide significant cost reductions while maintaining accuracy. Second, the benchmarking needs of model and downstream developers have been conflicting, making it hard to identify which agent would be best suited for a particular application. Third, many existing benchmarks have become brittle over time. This led to a lack of standardization in various ways. We prescribe a principled framework for avoiding overfitting. Finally, there is a lack of standardization in evaluation practices, leading to a lack of reproducibility. We describe a list of best practices that ensure different types of held-out samples are needed based on the desired level of generality. Without proper hold-outs, agent developers can take shortcuts, even unintentionally. We illustrate this with a case study of downstream evaluation.

## 1 Introduction

Computer AI systems, or AI agents, are becoming an important research direction. Zaharia et al. [63] argue that “compound AI systems will likely be the best way to maximize AI results in the future, and might be one of the most impactful trends in AI in 2024”. Over a dozen new benchmarks have been proposed, quickly adopted by the community [27, 62, 63, 64, 65, 66, 67, 68, 69]. Many benchmarks developed for LLM evaluation have also been used for agent evaluation.

Agent evaluation differs from language model evaluation in fundamental ways. Agents can be used on tasks that are harder, more realistic, have more real-world utility, and usually don’t have a single correct answer. For example, agents can use the command line to carry out tasks. SOTA-Agent even includes a command-line interface. Agents can cost much more than a single model cell. For example, the authors of SOTA-Agent capped a run of the agent at 64 USD, which translates to hundreds of thousands of language model tokens.

As a result, agent benchmarking comes with distinct challenges. This paper empirically demonstrates these challenges and provides recommendations for addressing them. Specifically, we make five contributions:

1. **AI agent evaluations must be cost-controlled (Section 2).** The language models underlying most AI agents are stochastic. This means simply calling the underlying model multiple times can increase accuracy [27, 6, 26]. We introduce three new simple baseline agents and empirically

\*Equal Contribution. Contact: {sayashk, strobelb}@princeton.edu

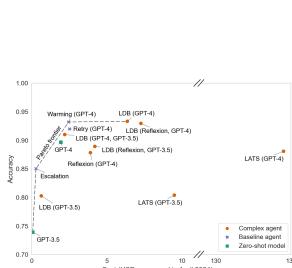


Figure 1: Some baseline designs for multi-step reasoning on GPT-3.5 agents. We mark each five times and report the mean accuracy and the mean total costs on the HumanEval problems. Where results for LDB have two models/agents in parenthesis, they indicate the language model or agent used to generate the code, followed by the language model used to debug the code. Where they have just one model/agent in parenthesis, they indicate the language model used to generate the code. All non-standard axes; in Appendix A, we show our results with the full y-axis as all error bars and provide additional details. Robustness checks are contained in Appendix A.2. Section 4 explains why we measure dollar costs instead of using proxies for cost such as the amount of compute used.

will be stable if model costs change). Meanwhile, the cost of accuracy strictly improves as we train and report less than half of LDB (0.5).

**Lack of evidence that System 2 approaches are responsible for performance gains.** Since papers proposing new agents haven’t adequately tested simple baselines, it has led to widespread beliefs in the community that complex ideas like planning, reflection, and debugging are responsible for accuracy gains. Based on our findings, the question of whether debugging, reflection, and other such “System 2” ideas [21] are responsible for performance gains is still open. We leave this question to future findings [50, 18]. In addition, the over-optimism about System 2 approaches is exacerbated by a lack of reproducibility and standardization that we report in Section 6. Failing to properly validate an approach can lead to a lack of confidence in its validity.

It is possible that System 2 techniques will be useful on harder programming tasks than those represented in HumanEval, such as SWBench [21].

To summarize this section, useful agent evaluations must control for cost – even if we ultimately don’t care about cost and only about identifying innovative agent designs. Accuracy alone cannot identify progress because it can be improved by scientifically meaningless methods such as retraining.

## 3 Jointly optimizing cost and accuracy can yield better agent designs

Visualizing the cost and accuracy of agents as a Pareto frontier opens up a new space for agent design: jointly optimizing cost and accuracy, which can lead to agents that cost less while maintaining accuracy. This is something that is fully attributable to other researchers in the field.

The total cost of running an agent includes fixed and variable costs. Fixed costs are non-linear expenses incurred when the agent’s hyperparameters (temperature, prompt, etc.) for a given task. Variable costs are incurred each time the agent is run and depend on the number of input and output tokens. The more an agent is used, the more the variable cost dominates (Appendix B).

4

show that they outperform many SoTA complex agent architectures on HumanEval [64, 65, 44] while costing much less. Therefore, agent evaluations must be cost-controlled; otherwise it will encourage researchers to develop extremely costly agents just to claim they topped the leaderboard.

2. **Jointly optimizing accuracy and cost can yield better agent design (Section 3).** Visualizing evaluations as a Pareto frontier allows us to jointly optimize both metrics. This is a hallmark of agent design: jointly optimizing the two metrics. We modify the DPSV framework [24] for joint optimization, lowering cost while maintaining accuracy on HotPotQA [60].
3. **Model developers and downstream developers have distinct benchmarking needs (Section 4).** There is a subtle difference between the two types of evaluation that model evaluation can be misleading when used for downstream evaluation. We argue that downstream evaluation should account for dollar costs, rather than proxies for cost such as the number of model parameters.
4. **Agent benchmarks enable shared knowledge (Section 5).** We show that different types of testing to agents are not equivalent. We show that a lack of generality in agent benchmarks means that different types of held-out samples are needed based on the desired level of generality. Without proper hold-outs, agent developers can take shortcuts, even unintentionally. We illustrate this with a case study of downstream evaluation.
5. **Agent evaluations lack standardization and reproducibility (Section 6).** We found pervasive shortcomings in the reproducibility of WebArena and HumanEval benchmarks (Table A6). These errors inflate accuracy estimates and lead to overoptimism about agent capabilities.

The overarching goal of our work is to stimulate the development of agents that are useful in the real world and not just accurate on benchmarks. (1) and (2) above do this by incorporating metrics beyond accuracy into agent evaluation and optimization; (4) and (5) do so by improving precision about what a benchmark aims to measure and ensuring that it actually measures that; and (3) does both.

### 1.1 What is an AI agent?

In traditional AI, agents are defined as entities that perceive and act upon their environment [40]. In the LLM era, the agents are defined as a language model that is trained to act as an agent. This is a subtle but important distinction. Most researchers have yet to formalize this language model as an agent. Many of them view it as a spectrum – sometimes denoted by the term “agentic” [38] – rather than a binary definition of an agent. We believe that there are many definitions, we do not propose a new, but rather identify the fact that each of them is to be considered more agentic according to existing definitions. We found three clusters of factors:

- **Environment and goals.** The more complex the environment – e.g. range of tasks and domains, multi-stakeholder, long time horizon, unexpected changes – the more AI systems operating in that environment are agents [41, 14]. Systems with more complex goals without being instructed on how to achieve them are also agents [41, 4, 14].
- **User interface and supervision.** AI systems that can be instructed in natural language and act autonomously on the user’s behalf are more agentic [14]. In particular, systems that require less supervision are more agentic [41, 4, 14]. We discuss the user supervision aspect in more detail in Section 3.2.
- **System design:** Systems that use design patterns such as tool use (e.g., web search, programming) or planning (e.g., reflection, subgoal decomposition) are more agentic [37, 38]. Systems whose control flow is driven by an LLM, and hence dynamic, are more agentic [57].

### 2. AI agent evaluations must be cost-controlled

#### 2.1 Maximizing accuracy can lead to unbounded cost

Calling language models repeatedly and taking a majority vote can lead to non-trivial increases in accuracy across benchmarks like GSM-8K, MATH, and MMLU [26, 6, 48].

When the agent environment has easy signals to check if an answer is correct, repeatedly retrying can lead to even more compelling performance gains [51]. Li et al. [27] showed that the accuracy

2

Joint optimization allows us to trade off the fixed and variable costs of running an agent. By spending more upfront on the one-time optimization of agent design, we can reduce the variable cost of running an agent (e.g., by finding shorter prompts and few-shot examples while maintaining accuracy).

As an illustration of the power of joint optimization, we verify the DPSV framework [24] and evaluate on the HumanEval benchmarks. We show HotPotQA [60] as an example of the benchmarks used to illustrate the effectiveness of DPSV in the original paper and has been featured in several official tutorials by the developers. We used the Optuna hyperparameter optimization framework [2] to search for few-shot examples to be included with an agent that minimizes cost while maintaining accuracy. Note that we expect joint optimization approaches to vastly outperform our approach. Our results are only a starting point intended illustrate the vast underexplored design space in agent design enabled by joint optimization.

#### 3.1 HotPotQA evaluation setup

We implement several agent designs to evaluate performance on multi-hop question-answering using DPSV. For retrieval, we use ColBERTv2 to query Wikipedia based on the HotPotQA task specification [60]. Performance is evaluated by comparing whether the agent successfully retrieved all ground-truth answers to a subset of the HotPotQA task. We use the samples from the HotPotQA training set to optimize the DPSV framework [24] and then the evaluation set to evaluate the results (this is consistent with the implementation of the DPSV pipelines provided by the developers to illustrate efficacy at multi-hop retrieval). We evaluate five agent architectures:

- **Unoptimized:** Do not optimize the agent’s prompt or include instructions on how to spend the budget on the one-time optimization of agent design.
- **Formating instructions only:** This is the same as the unoptimized baseline, but we add instructions on how to format generated outputs for saving retrieval queries.
- **Few-shot:** We use DPSV to identify effective few-shot examples using all 100 samples from the training set. We include formating instructions. Few-shot examples are selected based on success rate and average cost.
- **Random Search:** We use DPSV’s random search optimizer on a subset of the training data (50 of 100 samples) to select the best few-shot examples based on its performance on the remaining 50 samples. We include formating instructions.
- **Joint optimization:** We iterate over half of the training set (50 of 100 samples) to collect a set of few-shot examples that improve the agent’s accuracy. We use the other 50 samples for validation to measure accuracy and minimize the cost of the few-shot examples included in the prompt using parameter search. We implement parameter search using Optuna [2]. We search over the following parameters to find Pareto-optimal agent designs: (a) the temperature of the prompt, (b) the number of examples, (c) the budget for the selection of specific examples, and (d) whether to add formating instructions. Of the candidate agents selected by Optuna, we pick the one with the best accuracy on the development set as our joint optimization model.

We test all five of the above agent designs on two underlying models: Llama-3-70B and GPT-3.5.

#### 3.2 HotPotQA results: Joint optimization reduces cost while maintaining accuracy

Fig. 2 shows the results of joint optimization on the HumanEval test set. We find that joint optimization is significantly more cost-effective than the unoptimized baseline. For example, for Llama-3-70B, joint optimization leads to 53% lower variable cost with similar accuracy compared to both default DPSV implementations. Similarly, for Llama-3-70B, it leads to a 41% lower cost while maintaining the same accuracy.

**Tradeoffs between fixed and variable costs for agent design.** Our joint optimization formulation provides a way to trade off fixed and variable costs (Appendix B). In particular, we find that it is used for HotPotQA tasks, the Llama-3-70B as well as the GPT-3.5 joint optimization model both become cheaper (in terms of total cost) compared to the default DPSV implementation, after 1,350 tasks

5

of AlphaCode increases from close to 0% zero-shot to over 15% with 1,000 retries and over 30% with a million retries (accuracy is measured by how often one of the top 10 answers generated by the model is correct). Thus, there is seemingly no upper bound on the number of retries that can increase, which increases the generation budget but has been shown to improve accuracy in various applications [56]. Coding optimizations often include signals of correctness, such as test cases to check if a given solution is correct. Agent developers can keep sampling from an underlying model until the solution passes the test cases. Our results below show that this is true for HumanEval.

#### 2.2 Visualizing the accuracy-cost tradeoff using a Pareto curve

In the last year, many agents have been claimed to achieve state-of-the-art accuracy on coding tasks. But at what cost? To visualize the tradeoff, we re-evaluated the accuracy of three agents.

Specifically, we included agents from the HumanEval leaderboard with PyTorch/Code that share their code publicly [7]: LDB [64], LATS [65], and Reflexion [44]. These agents rely on the code generation pipeline [44]. They are designed to be production-ready and were provided with a pre-trained model to debug the code [64]. Let’s look at alternative paths in the code generation process [65], or “reflect” on why the model’s outputs were incorrect before generating another solution [44, 65, 64].

We also evaluated the cost and time requirements of running these agents. In addition, we calculated the accuracy of the most expensive agent.

#### • GPT-3.5 and GPT-4 models (zero shot; no agent architecture)

**Results:** We re-implement a model with the same hyperparameters as the baseline set to, up to five times, if it fails the test set provided with the problem description. Retrying makes sense because LLMs aren’t deterministic even at temperature zero (Appendix A.1).

**Warning:** This is the same as the retry strategy, but we gradually increase the temperature of the underlying model with each run, from 0 to 0.5. This increases the stochasticity of the model and, we believe, increases the chance of getting a correct answer.

**Reflexion:** We start with a cheap model (LDB [3-8]) and escalate to more expensive models (GPT-3.5, Llama-3-70B, GPT-4) as we encounter a test case failure.

We use the modified benchmark version of HumanEval provided with the LDB paper [64] since it includes example test cases for all 164 tasks (in the original benchmark, example test cases are provided for only 16 of 164 tasks, as detailed in Section 6).

#### 2.3 Two-dimensional evaluation yields surprising insights

Fig. 1 shows our main results for this section. Note that we are on the Pareto frontier of there is no other agent that has significantly better performance on both dimensions simultaneously (see Appendix A.1).

**“State-of-the-art” agent architectures for HumanEval do not outperform simple baselines.** There is no significant accuracy difference between the agent and the baseline agent architectures. In other words, we are aware of no reason that compares the two agent architectures with any of the last three of our simple baselines on HumanEval (retry, warning, escalation).

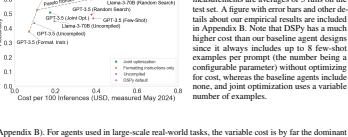
**Agents differ drastically in terms of cost.** Despite substantially similar accuracy, the cost can differ by almost two orders of magnitude. Yet, the cost of running these agents isn’t a top-line metric reported in any of these papers. Reflexion and LDB cost over 50% more than the warning strategy, and LATS over 50 times more (all these costs are entirely or predominantly from calls to GPT-4, as these ratios are the same for all other agents).

**Escalation:** We constrain the agent to be correct, because given two points  $a$  and  $b$  on the graph corresponding to agents  $A$  and  $B$ , we can always linearly interpolate between them by creating a new agent that has  $A$  with probability  $\alpha$  and  $B$  with probability  $1 - \alpha$ . This is the same as the “escalation” strategy in the original HumanEval paper.

**“We constrain the Pareto frontier to be convex, because given two points  $a$  and  $b$  on the graph corresponding to agents  $A$  and  $B$ , we can always linearly interpolate between them by creating a new agent that has  $A$  with probability  $\alpha$  and  $B$  with probability  $1 - \alpha$ .**

**Chen et al. [6]** do compare the effect of reusing the same model multiple times on HumanEval, but they do not test the performance of GPT-4, nor do they argue for a specific complex agent architecture.

3



(Appendix B). For agents used in large-scale real-world tasks, the variable cost is by far the dominant term compared to the fixed cost, by orders of magnitude, as an agent might be used millions of times. In summary, joint optimization allows for efficient agent design. This comes at a small fixed cost for optimizing the design, which is insignificant if the agent is used thousands of times.

## 4 Model and downstream developers have distinct benchmarking needs

All evaluations are used by model developers and AI researchers to identify which changes to the training set or downstream evaluation set are most effective. These evaluations are used by downstream developers to decide which AI systems to use in their products for procurement decisions (downstream evaluation). The difference between model evaluation and downstream evaluation is underappreciated. This has led to many misconceptions about what is used in the real world.

**Model evaluation** is a scientific question of interest to researchers. Here, it makes sense to stay away from dollar costs, because reporting costs breaks many properties of benchmarks that we take for granted: measurement doesn’t change over time (whereas costs tend to increase over time); different models can have the same cost; and the cost of a model is not proportional to scale, leading to lower inferred costs. Because of this, researchers usually pick a different axis for the Pareto curve, such as parameter count or the amount of compute used to train a model.

**Proxy for cost is misleading: downstream evaluation.** In contrast, the proxy for cost is the amount of compute used to train a model. This is a scientific question of interest to downstream developers. Here, it makes sense to stay away from dollar costs, because reporting costs breaks many properties of benchmarks that we take for granted: measurement doesn’t change over time (whereas costs tend to increase over time); different models can have the same cost; and the cost of a model is not proportional to scale, leading to lower inferred costs. Because of this, researchers usually pick a different axis for the Pareto curve, such as parameter count or the amount of compute used to train a model.

Downstream developers don’t care about the number of inference requests they receive over time. They simply care about the dollars spent on accuracy. “Mistral” chose “active parameters” as a proxy, which makes sense because it makes their models look better than older models such as Meta’s Llama and Cohere’s Command R+. If every model developer picked a proxy that makes their model look good, multi-dimensional evaluation would lose its usefulness.

## 5.1 Case study of the STeP agent on WebArena.

WebArena can be evaluated in many capacities: navigating to a website, scrolling, selecting the right web element etc. There are many different types of websites that can be used such benchmarks: e-commerce, social media, information search etc. WebArena is an agent benchmark that aims to evaluate agents on tasks on the web [66]. It includes clones of six different websites (GitHub, Reddit, LinkedIn, YouTube, Netflix, and Wikipedia) and two tools (calculator and scratchpad). It has 812 different tasks that involve interacting with these websites, such as “find the address of all US international airports that are within a driving distance of 60km”, “is Niagara Falls” and “post a question to a subreddit related to New York City”.

We find that it is challenging to model drift. Benchmark developers would need to find changes made to published websites and incorporate those into the test sets. Further, as we saw above, the held-out set would need to be kept secret, since agent developers could otherwise overfit to the specific changes in the benchmark. Still, we view these steps as necessary for meaningful evaluation.

Consider the top agent on the WebArena leaderboard, called STeP [17]. It has an accuracy of 35.8%, more than twice that of the second-best agent. This is a significant achievement for a model that is over 100 times smaller than the next-best agent [13]. Does STeP achieve this high accuracy?

It turns out that STeP handles policies to solve the specific tasks included in WebArena. For example, when WebArena loads tasks involves navigating to a user’s profile. The STeP policy for this task is to look at the current base URL and add a suffix “/user/”.

This is brittle: the policy would no longer be effective if the website updates its URL structure (an example of drift). Even if the probability of an individual policy failing is small, an agent might need to recall many different policies for times for each task. The overall probability of failure compounds quickly.

To be clear, the STeP developer’s goals are orthogonal to the benchmark developers’ goals—creating competitive policies for accomplishing fixed tasks that are known a priori. STeP’s developer’s goals are to learn the specific tasks and then solve them.

Yet the leaderboard accuracy on WebArena is misleading from the perspective of downstream developers, who might be using the WebArena leaderboard to understand the accuracy of web agents’ real-world tasks and make decisions about which agent to adopt. This happens even more frequently if we consider WebArena a domain-general benchmark.

This can be justified on the basis that for previous web benchmarks, the “functionality of many environments is a limited version of their real-world counterparts, leading to a lack of task diversity” [66]. This suggests that the WebArena tasks aim to simulate the development of agents that work in many different tasks on the web.

Unfortunately, WebArena lacks a held-out test set for evaluating whether an agent can perform well on unseen tasks. For example, we can test an agent’s performance on a specific task, but not on a completely different task.

Note that while the held-out set contains different tasks (such as the accuracy scores from newly added websites) compared to those in the training set, the accuracy scores from newly added websites are not included in the held-out set. This motivates the need for held-out tasks and that a held-out set will be added in future editions of the benchmark.

## 5.2 Agent benchmarks don’t account for humans in the loop

The degree of human supervision, feedback, and intervention required for an agent to perform a task can be seen as a spectrum. Consider a data analysis task. On one end of the spectrum, the analyst might use a chart to help with tasks like debugging. Here the user is firmly in control and executes code for certain data

8

Addressing challenges to cost evaluation. As discussed above, there are several hurdles to cost evaluation. Different providers can charge different amounts for the same model, the cost of an API call might change overnight, and cost might vary based on model developer decisions, such as whether to use a specific API or not.

These slowdowns can be partly addressed by making the evaluation results customizable using mechanisms to adjust the cost of running models, i.e., providing users the option to adjust the cost of input and output tokens for their choice of relevance to their task. This is something that downstream developers have been doing for years now [21].

It is possible that System 2 techniques will be useful on harder programming tasks than those represented in HumanEval, such as SWBench [21].

To summarize this section, useful agent evaluations must control for cost – even if we ultimately don’t care about cost and only about identifying innovative agent designs. Accuracy alone cannot identify progress because it can be improved by scientifically meaningless methods such as retraining.

**4.1 Implications for benchmark design using a case study of NovelaQ**

The NovelaQ system model and downstream evaluation can face challenges in evaluating benchmarks, since they are typically a type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is one prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

NovelaQ is a prominent type of shortcut, and a serious problem for agent benchmarks, since they are typically a type of shortcut. This is a more serious challenge for agent benchmarks, since they are typically a type of shortcut. We argue that the challenges faced by downstream evaluation are similar to those faced by the NovelaQ system model.

Nov



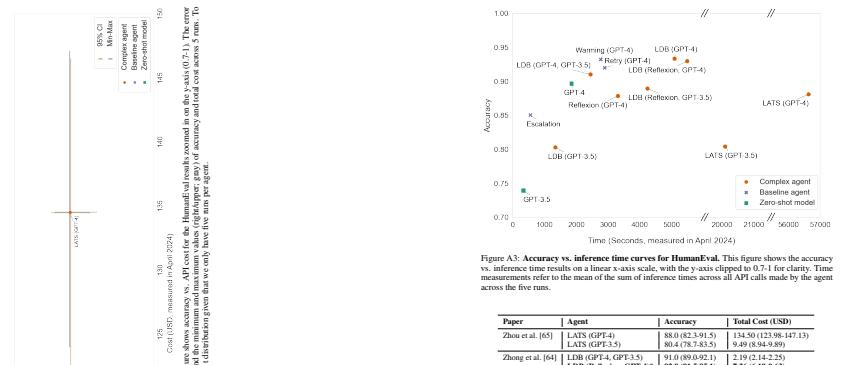


Figure A5: Accuracy vs. inference time curve for HumanEval. This figure shows the accuracy vs. inference time results on a linear x-axis scale, with the y-axis clipped to 0.7 for clarity. Time measurements refer to the mean of the sum of inference times across all API calls made by the agent across the five runs.



Figure A6: Error bar for our HumanEval analysis. The figure shows accuracy vs. API cost for the HumanEval results zoomed in on the y-axis (0-1). The error bars represent the minimum and maximum values (left/cross) and the mean and standard deviation (right/cross). We use the 90% confidence intervals. As we can see, the variance in the minimum and maximum values is very low, indicating that the agents are stable across the five runs.

Paper	Agent	Accuracy	Total Cost (USD)
Zhou et al. [65]	LATS (GPT-4)	88.0 (82.91%)	134.50 (123.98, 147.13)
	LATS (GPT-3.5)	80.4 (87.83%)	94.98 (94.84, 98.89)
Zhong et al. [64]	LDB (GPT-4, GPT-3.5)	91.0 (89.92%)	2.19 (2.14, 2.25)
	LDB (Reflexion, GPT-4)	90.5 (89.82%)	2.19 (2.14, 2.25)
LDB (GPT-4*)	LDB (GPT-4*)	88.9 (86.61%)	4.39 (3.98, 4.37)
	LDB (Reflexion, GPT-4*)	88.9 (86.61%)	4.39 (3.98, 4.37)
LDB (GPT-4, GPT-3.5)	LDB (GPT-4, GPT-3.5)	93.3 (92.14%)	6.36 (5.11, 7.08)
	LDB (Reflexion, GPT-4, GPT-3.5)	93.3 (92.14%)	6.36 (5.11, 7.08)
GPT-4 (baseline)	GPT-4 (baseline)	89.6 (87.80%)	1.59 (1.51, 1.77)
	GPT-3.5 (baseline)	89.6 (87.80%)	1.59 (1.51, 1.75)
Shin et al. [44]	Reflexion (GPT-4)	87.0 (84.84%)	3.80 (3.76, 3.84)
	Reflexion (GPT-4*)	84.2 (82.14%)	3.80 (3.76, 3.84)
Our baselines	Reflexion (GPT-4)	92.0 (91.92%)	2.31 (2.46, 2.56)
	Reflexion (GPT-4*)	92.0 (91.92%)	2.31 (2.46, 2.56)
Escalation	Reflexion (GPT-4)	85.0 (84.84%)	0.27 (0.24, 0.28)
	Reflexion (GPT-4*)	85.0 (84.84%)	0.27 (0.24, 0.28)

Table A1: Accuracy and total cost of HumanEval agents. We run each agent five times and report the mean accuracy and the mean total cost on the 164 HumanEval problems. The minimum and maximum values are included in the parentheses. The paper does not specify that the authors did not evaluate in the original paper. In particular, the original LDB paper does not include tests with GPT-4 as the debugger.

#### A.1 Implementation details

We used gpt-3.5-turbo-0125 for GPT-3.5 implementations and gpt-4-turbo-2024-04-09 for GPT-4 implementations. The underlying prices for GPT-3.5 and GPT-4 were 0.55 (1.55) and 105 (305) per one million input (output) tokens, respectively, as of April 2024. We describe all implementations in

temperature is set to zero. If at any point a solution passes the tests given in the HumanEval problem description, we evaluate this as the final solution of the agent for this problem. This leads to slightly lower accuracy compared to GPT-4, since some solutions from cheaper models might pass the tests but fail our robustness checks.

**Pareto frontier.** In our analysis, Pareto frontiers are employed to evaluate agent designs. We define the Pareto frontier as the set of points (agents) that are non-dominated by any other agent in terms of mean cost and accuracy. The frontier is constrained to be convex, meaning if two agents lie next to each other on the frontier, any linear combination of these agents should also yield a point that lies on the frontier. We provide a simple example implementation of how we compute Pareto frontiers on simulated agent evaluation data.<sup>12</sup>

#### A.2 Robustness checks with June 2023 versions of GPT models

One concern with our analysis is that we use the latest versions of Optuna's April 2024 tuning models, since we conduct our experiments with the June 2023 versions of GPT-3.5 and GPT-4. We conduct additional robustness checks with the June 2023 versions of GPT-3.5 and GPT-4. We find substantially similar results for this version: complex agents are no better than simple agent because they are more expensive. In fact, in most cases, the simple agents are more efficient than GPT-4 is much more expensive than the April 2024 version, leading to the big difference in inference cost. For the LDB agents, there are some significant outliers across tasks for both, LDB (GPT-4) and LDB (GPT-3.5), with some remaining more than 2 times more expensive than LDB (GPT-3.5). Overall, there are more extreme outliers for LATS (GPT-4) and LATS (GPT-4\*).

In the end, we had to exclude one task, i.e., HumanEval(83) from our analysis for LATS (GPT-3.5), which did not stop running even after 5 hours. We marked this task as failed and excluded the cost and accuracy of this task from the results shown above. The reason is one of the tasks excluded from the subset of HumanEval that the authors evaluated the LATS agent on.

One concern with our analysis is that we use the latest versions of Optuna's April 2024 tuning models, since we conduct our experiments with the June 2023 versions of GPT-3.5 and GPT-4. We conduct additional robustness checks with the June 2023 versions of GPT-3.5 and GPT-4. We find substantially similar results for this version: complex agents are no better than simple agent because they are more expensive. In fact, in most cases, the simple agents are more efficient than GPT-4 is much more expensive than the April 2024 version, leading to the big difference in inference cost. For the LDB agents, there are some significant outliers across tasks for both, LDB (GPT-4) and LDB (GPT-3.5), with some remaining more than 2 times more expensive than LDB (GPT-3.5). Overall, there are more extreme outliers for LATS (GPT-4) and LATS (GPT-4\*).

In the end, we had to exclude one task, i.e., HumanEval(83) from our analysis for LATS (GPT-3.5), which did not stop running even after 5 hours. We marked this task as failed and excluded the cost and accuracy of this task from the results shown above. The reason is one of the tasks excluded from the subset of HumanEval that the authors evaluated the LATS agent on.



Table A2: Accuracy and total cost of HumanEval robustness checks with June 2023 versions of GPT models. As for the main analysis, we run each agent five times and report the mean accuracy and the mean total cost on the 164 HumanEval problems. The minimum and maximum values are included in the parentheses. Rows marked with an asterisk indicate that the authors did not evaluate in the original paper.

Paper	Agent	Accuracy	Total Cost (USD)
Zhou et al. [65]	LATS (GPT-4)	81.6	360.02
	LATS (GPT-3.5)	77.4	8.97
Zhong et al. [64]	LDB (GPT-4, GPT-3.5)	88.7 (87.28%)	3.20 (3.14, 3.26)
	LDB (Reflexion, GPT-4)	90.5 (89.92%)	3.20 (3.14, 3.26)
LDB (GPT-4*)	LDB (GPT-4*)	91.2 (90.24%)	9.31 (8.27, 10.26)
	LDB (Reflexion, GPT-4*)	91.2 (90.24%)	9.31 (8.27, 10.26)
GPT-4 (baseline)	GPT-4 (baseline)	86.5 (84.87%)	2.94 (2.92, 2.95)
	GPT-3.5 (baseline)	75.2 (74.76%)	0.04 (0.04, 0.04)
Shin et al. [44]	Reflexion (GPT-4)	80.2 (78.73%)	0.29 (0.28, 0.69)
	Reflexion (GPT-4*)	80.2 (78.73%)	0.29 (0.28, 0.69)
Our baselines	Reflexion (GPT-4)	90.6 (91.92%)	3.33 (3.32, 3.33)
	Reflexion (GPT-4*)	89.8 (87.28%)	3.33 (3.32, 3.33)
Escalation	Reflexion (GPT-4)	87.8 (86.84%)	0.42 (0.40, 0.44)
	Reflexion (GPT-4*)	87.8 (86.84%)	0.42 (0.40, 0.44)

Table A3: Accuracy and two types of cost of agent designs evaluated on HotPotQA. We evaluate each agent five times on the test set and report the mean accuracy. Variable cost refers to the cost per 100 inferences on HotPotQA. Fixed costs are the total costs incurred during the optimization of the agent design. The minimum and maximum accuracy are included in the parentheses.

Model	Agent	Accuracy	Cost (USD, measured in May 2024)
GPT-3.5	RFSW* few-shot	0.495 (0.50-0.485)	0.376 (0.374, 0.376)
	Formatting instructions only	0.47 (0.46-0.48)	0.384 (0.382, 0.385)
GPT-3.5	Joint optimization	0.49 (0.48-0.50)	0.374 (0.372, 0.375)
	Uncoupled	0.377 (0.36-0.39)	0.071 (0.070, 0.0715)
LATS (3-708)	RFSW* few-shot	0.496 (0.49-0.496)	0.426 (0.425, 0.427)
	Formatting instructions only	0.436 (0.39-0.48)	0.094 (0.094, 0.097)

Table A3: Accuracy and two types of cost of agent designs evaluated on HotPotQA. We evaluate each agent five times on the test set and report the mean accuracy. Variable cost refers to the cost per 100 inferences on HotPotQA. Fixed costs are the total costs incurred during the optimization of the agent design. The minimum and maximum accuracy are included in the parentheses.

(a) GPT-3.5

(b) LDB (GPT-4), LDB (GPT-3.5), LATS (GPT-4), LATS (GPT-3.5), and Reflexion (GPT-4).

(c) LDB (Reflexion, GPT-4), LDB (Reflexion, GPT-3.5), LDB (GPT-4), LDB (GPT-3.5), and LATS (GPT-4).

Figure A7: Pareto frontiers returned by joint optimization. This figure shows the Pareto frontier of programs returned by our joint optimization method with linear x-axes and y-axes clipped to 0.14-0.14 for clarity. Note that the x-axes do not follow the same scale. Accuracies are calculated on the development set. Cost measurements refer to API prices as of May 2024.

B Additional details on Section 3: Jointly optimizing cost and accuracy can yield better agent designs

In this section, we include Figure A5, reporting the results of our analysis on HotPotQA along with bars for accuracy and cost, as well as Table A3, which reports our results and inference and optimization cost for our five agent designs. In Fig. A7, we report the Pareto frontiers returned by our joint optimization method for both models.

<sup>12</sup>See: [https://colab.research.google.com/drive/1yb3puyt\\_qhJsd0gfdCQnfPwzFvCP9C57](https://colab.research.google.com/drive/1yb3puyt_qhJsd0gfdCQnfPwzFvCP9C57)

Figure A8: Robustness checks with June 2023 versions of GPT models. This figure shows the results of our robustness checks with linear x-axes and y-axes clipped to 0.71 for clarity. Time measurements refer to the mean of the sum of inference times across all API calls made by the agent across the five runs. Cost and accuracy measurements refer to the mean across five runs.

Figure A9: Accuracy vs. inference time curve for HumanEval. The plot shows accuracy (y-axis, 0.70 to 1.00) versus time in seconds (x-axis, 0 to 5000). Multiple curves represent different agents: GPT-4, GPT-3.5, LDB (GPT-4), LDB (GPT-3.5), LATS (GPT-4), LATS (GPT-3.5), Reflexion (GPT-4), Reflexion (GPT-3.5), Warning (GPT-4), and LDB (Reflexion). A legend indicates: ● GPT agent, □ Baseline agent, ■ Zero-shot model. The plot also includes shaded regions for 90% CI and Max. The curves generally show higher accuracy for GPT-4 and LDB (GPT-4) compared to GPT-3.5 and LDB (GPT-3.5).

Figure A10: Error bars for our HumanEval analysis. The plot shows error bars for cost (USD) and accuracy (%). The x-axis represents error bars for time (seconds). The plot highlights that GPT-4 and LDB (GPT-4) have the lowest error bars, indicating more stable performance.

Figure A11: Accuracy vs. inference time curve for HumanEval. The plot shows accuracy (y-axis, 0.70 to 1.00) versus time in seconds (x-axis, 0 to 5000). Multiple curves represent different agents: GPT-4, GPT-3.5, LDB (GPT-4), LDB (GPT-3.5), LATS (GPT-4), LATS (GPT-3.5), Reflexion (GPT-4), Reflexion (GPT-3.5), Warning (GPT-4), and LDB (Reflexion). A legend indicates: ● GPT agent, □ Baseline agent, ■ Zero-shot model. The plot also includes shaded regions for 90% CI and Max. The curves generally show higher accuracy for GPT-4 and LDB (GPT-4) compared to GPT-3.5 and LDB (GPT-3.5).

Figure A12: Error bars for our HumanEval analysis. The plot shows error bars for cost (USD) and accuracy (%). The x-axis represents error bars for time (seconds). The plot highlights that GPT-4 and LDB (GPT-4) have the lowest error bars, indicating more stable performance.

Figure A13: Accuracy vs. inference time curve for HumanEval. The plot shows accuracy (y-axis, 0.70 to 1.00) versus time in seconds (x-axis, 0 to 5000). Multiple curves represent different agents: GPT-4, GPT-3.5, LDB (GPT-4), LDB (GPT-3.5), LATS (GPT-4), LATS (GPT-3.5), Reflexion (GPT-4), Reflexion (GPT-3.5), Warning (GPT-4), and LDB (Reflexion). A legend indicates: ● GPT agent, □ Baseline agent, ■ Zero-shot model. The plot also includes shaded regions for 90% CI and Max. The curves generally show higher accuracy for GPT-4 and LDB (GPT-4) compared to GPT-3.5 and LDB (GPT-3.5).

Figure A14: Error bars for our HumanEval analysis. The plot shows error bars for cost (USD) and accuracy (%). The x-axis represents error bars for time (seconds). The plot highlights that GPT-4 and LDB (GPT-4) have the lowest error bars, indicating more stable performance.

Figure A15: Error bars for our HotPotQA analysis. The figure shows retrieval accuracy vs. API cost for the HotPotQA. The error bars represent 95% confidence intervals (left/below); and the minimum and maximum values (right/above) of gray bar and total cost and time of 3 runs. To calculate the 95% confidence interval, we use the formula  $\text{CI} = \text{Max} - \text{Min}$ . A detailed analysis shows that we have five runs per agent. Note that the error bars account for the in-sample variance on the test set and do not account for the variability induced during optimization and resampling.

detail below. In addition to our analysis in the main text, we also conducted robustness checks with the June 2023 versions of GPT-3.5 and GPT-4 and found substantially similar results (see Appendix A.2). When we consider our baseline test set in April 2024, the total cost of the baseline is 0.361 (505) per one million input (output) tokens, while the 0.45-0.56 (603) is identified in January 2024 version – 0.55 (1.55) and 105 (305) per one million input (output) tokens, respectively.

**HumanEval version.** In evaluating our baseline agent architectures on the LDB paper [64], we use the most recent version provided in the LDB paper [64]. This version includes internal test cases for all tasks in the original benchmark; test cases were provided for only 161 of 164 tasks, as detailed in the paper.

**GPT-3.5 and GPT-4.** We implement the model baselines using the simple (zero shot; without agent architecture) strategy provided with the LDB paper [64]. This includes a text prompt and the example tests accompanying the HumanEval coding problem as inputs to the model.

**LDB [64].** The LDB agent uses the language models: (i) for generating code and another for debugging and testing. We use the monologue “LDB (Generator, Debugger)” to specify which models were used. If the same model served both functions, we list it only once within parentheses. We kept all parameters as specified in the accompanying code and reflections.

**Reflection [44].** We left all parameters as specified in the original paper corresponding with the original repository.

**Escalation.** We modify the simple strategy by setting the temperature to 0.1, the expansion factor to 3, and the temperature to zero for generating function implementations.

**Retry.** This baseline uses the sample strategy implemented in the LDB paper [64]. We used this strategy to repeatedly pass the same language models, keeping all parameters equal across retrials, as long as the code outputted by the model failed at least one of the example tests. If at any point a solution was found, we increased the temperature to 0.3. The maximum number of internal tests was set to 6 for runs with GPT-3.5 and 4 for runs with GPT-4. The difference in the number of internal tests for GPT-3.5 and GPT-4 was not present in the paper; we learned this from our correspondence with the original paper.

**Reflexion [44].** We left all parameters as specified in the original paper corresponding with the original repository.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Retry baseline.** This baseline uses the sample strategy implemented in the LDB paper [64]. We used this strategy to repeatedly pass the same language models, keeping all parameters equal across retrials, as long as the code outputted by the model failed at least one of the example tests. If at any point a solution was found, we increased the temperature to 0.3. The maximum number of internal tests was set to 6 for runs with GPT-3.5 and 4 for runs with GPT-4. The difference in the number of internal tests for GPT-3.5 and GPT-4 was not present in the paper; we learned this from our correspondence with the original paper.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the expansion factor to 3, and temperature to zero for generating function implementations.

**Setting temperature.** We set the temperature to 0.1, the

