AVIARY: TRAINING LANGUAGE AGENTS ON CHALLENGING SCIENTIFIC TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Solving complex real-world tasks requires cycles of actions and observations. This is particularly true in science, where tasks require many cycles of hypothesis, experimentation, and analysis. Language agents hold promise for automating intellectual tasks in science because they can interact with tools via natural language or code. However, their flexibility creates conceptual and practical challenges for software implementations, since agents may comprise non-standard components such as internal reasoning, planning, tool usage, as well as the inherent stochasticity of temperature-sampled language models. Here, we introduce Aviary, an extensible gymnasium for language agents. We formalize agents as policies solving language-grounded partially observable Markov decision processes, which we term language decision processes. We then implement five environments, including three challenging scientific environments: (1) manipulating DNA constructs for molecular cloning, (2) answering research questions by accessing scientific literature, and (3) engineering protein stability. These environments were selected for their focus on multi-step reasoning and their relevance to contemporary biology research. Finally, with online training and inference-time compute scaling, we show that language agents based on open-source, non-frontier LLMs can match and exceed both frontier LLM agents and human experts on multiple tasks at up to 100x lower inference cost.

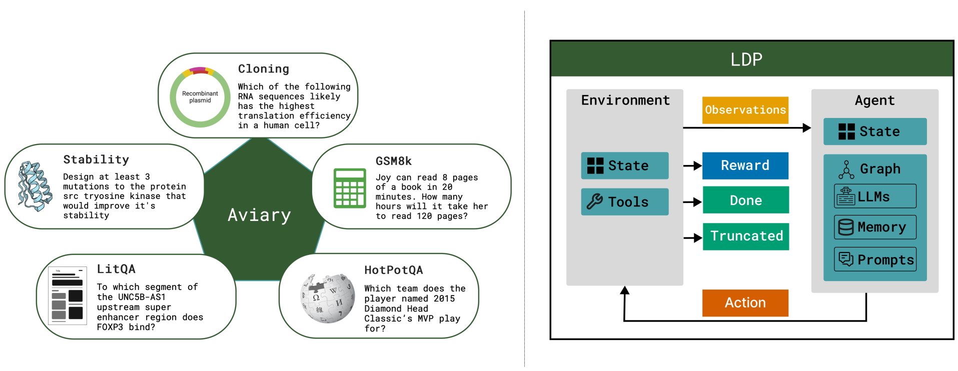


Figure 1: An overview of the five implemented Aviary environments and the language decision process (LDP) framework. The term language decision process here jointly refers to our theoretical description of the class of problems solved by language agents, as well as a software framework for implementing language agents based on a stochastic computation graph that enables training of language agent components such as LLM weights, prompts, memories, and LLM sampling parameters such as temperature.

1 INTRODUCTION

Language agents (Mialon et al., 2023; Xi et al., 2023; Gao et al., 2023; Sumers et al., 2024) are AI agents (Russell & Norvig, 2016) that integrate LLMs (Brown et al., 2020; Achiam et al., 2023; Bowman, 2023) as core components. LLMs excel at zero-shot generalization (Zeng et al., 2023a;

Szot et al., 2024), providing a notable advantage over traditional AI agents, such as those based on handcrafted rules or reinforcement learning, which often struggle to generalize to new environments (Lake et al., 2017). While LLMs can exhibit flawed reasoning and logic when used in isolation (Creswell et al., 2023; Frieder et al., 2024b;a), constructing a language agent by grounding LLMs in an environment with observational feedback can mitigate these issues. Early work on language agents used LLMs to directly output actions in the external environment (Brohan et al., 2023; Huang et al., 2022; Dasgupta et al., 2022), while more recently, language agents have been augmented with internal reasoning (Yao et al., 2023; Shinn et al., 2024) and planning procedures (Hao et al., 2023; Yao et al., 2024), as well as long-term memory storage (Park et al., 2023; Wang et al., 2024a).

An emergent research challenge is to pose a theoretical description of the learning problem solved by language agents (Sumers et al., 2024; Zhuge et al., 2024) and to develop efficient methods to optimize the components of a language agent (Zhuge et al., 2024; Yuksekgonul et al., 2024; Cheng et al., 2024). Here, we define common language agent tasks as language decision processes (LDPs) and frame language agents as stochastic computation graphs (Schulman et al., 2015) that may be trained to solve LDPs. We show that pre-existing agents (Yao et al., 2023; Shinn et al., 2024; Yao et al., 2024) can be implemented within our stochastic computation graph framework and introduce a simple and extensible software package named LDP that enables modular interchange of environments, agents, and optimizers, simplifying experimentation across a variety of settings.

In the problems we consider, we use the term optimization of language agents in the reinforcement sense to encompass procedures that yield iterative improvement of the language agent over time through feedback from an environment. An example of one such optimization algorithm is expert iteration (EI) (Anthony et al., 2017; Anthony, 2021; Havrilla et al., 2024) which achieves learning through successive rounds of supervised fine-tuning on (self-) generated trajectories from a progressively stronger language agent.

In what follows, we introduce our definition of an environment, a language decision process, and optimization of agents within a stochastic computation graph. We recast popular benchmarks such as GSM8K (Cobbe et al., 2021) and HOTPOTQA (Yang et al., 2018) as environments and integrate three scientific environments related to challenging tasks in the natural sciences. The scientific environments are (1) DNA construct engineering, where the task is to answer questions pertaining to molecular cloning (Laurent et al., 2024), (2) scientific literature question answering, where the task is to answer a multiple choice question by finding a specific passage from the scientific literature (Lála et al., 2023; Skarlinski et al., 2024), and (3) protein design, where the goal is to propose mutations to improve the stability of a given protein sequence (Gao et al., 2020; Khakzad et al., 2023). On the DNA construct design and scientific literature question answering environments, we demonstrate that language agents based on the small, open-source `Llama-3.1-8B-Instruct` model, when trained with expert iteration and using inference-time majority vote sampling, can exceed the performance of both human experts and frontier LLMs.

The environment framework described in this work, Aviary, is available at the anonymous GitHub link aviary and the stochastic computation graph framework together with language agent implementations and training code is available at the link ldp. The relationship between the Aviary and LDP frameworks is illustrated in Figure 1.

2 Related Work

Language Agent Formalisms Although language agents have achieved impressive empirical performance across a range of applications (Mialon et al., 2023; Skarlinski et al., 2024; Huang et al., 2024a), there is still no universally agreed upon theoretical framework for defining a language agent. In terms of conceptual models, the cognitive architectures for language agents (CoALA) framework (Sumers et al., 2024), inspired by ideas from production systems and cognitive architectures, taxonomizes agents according to their information storage (working and long-term memories), decision-making procedures e.g. planning, and action space (divided into internal and external actions). Similarly, in Weng (2023), the author describes language agents as consisting of memory, planning, and tool usage components. Theoretically, many works represent language agents as partially observable Markov decision processes (POMDPs) (Carta et al., 2023; Christianos et al., 2023; Wen et al., 2024b;a; Nguyen et al., 2024; Zhai et al., 2024; Song et al., 2024) yet differ in their treatment of the action space e.g. in Christianos et al. (2023) the authors partition the action space into internal and external actions in a similar fashion to CoALA where internal actions are a family of functions that operate on the agent's memory and external actions elicit an interaction with the environment. By contrast, in Wen et al. (2024b) the authors do not make a distinction between internal and external actions. In Chen et al. (2024c) the authors introduce a general framework for studying the design and analysis of LLM-based algorithms based on a computational graph where they assume LLM nodes are stateless, leaving consideration of aspects of language agents such as memory to future work.

Language Agent Optimization Frameworks Optimization of language agents may entail the learning of prompts, parametrized tools, LLM weights, inference hyperparameters, as well as more exotic parameters such as edges between nodes in computation graphs. Frameworks such as LangChain (Chase, 2022) and LlamaIndex (Liu, 2022) support manual optimization of prompts via human editing. Optimizers such as EcoOptiGen (Wang et al., 2023a) leverage black-box optimization schemes to learn LLM inference hyperparameters. Prompt optimization comprises the optimization of white-box LLMs and black-box LLMs (API-based models that cannot be differentiated through). In white-box prompt optimization (Shin et al., 2020; Li & Liang, 2021; Jia et al., 2022; Chen et al., 2022) numerical gradients can be taken over soft prompts (Qin & Eisner, 2021), the embedding representation of the text-based 'hard' prompt. In black-box prompt optimization a multitude of techniques have been applied to overcome the absence of gradients (Guo et al., 2024a; Ma et al., 2024a; Zhang et al., 2024; Cheng et al., 2023; Yang et al., 2024; Lin et al., 2024a; Hu et al., 2024b; Wu et al., 2024c; Lin et al., 2024b; Chen et al., 2024b; Zhou et al., 2023; Pryzant et al., 2023; Sabbatella et al., 2024; Chen et al., 2023b; Wang et al., 2024c; Mañas et al., 2024; Do et al., 2023; Sordoni et al., 2024; Sabbatella et al., 2023; Wen et al., 2024c; Ye et al., 2023; Wu et al., 2024a). Tool learning (Qu et al., 2024; Schick et al., 2024) can be attempted through in-context demonstrations (Qin et al., 2023) or by finetuning LLM weights on example demonstrations of appropriate tool usage (Havrilla et al., 2024; Yin et al., 2024) using techniques such as expert iteration (Anthony et al., 2017; Anthony, 2021). We discuss further related work on language agent optimization frameworks and benchmarks in Appendix D.

Our principal contributions are: (1) A precise definition of language decision processes (LDPs) for language agent tasks that encompass many proposed agent architectures as stochastic computation graphs. (2) We introduce Aviary, a gym framework that emphasizes multi-step reasoning and tool usage featuring five gym implementations (including three for scientific tasks). (3) We demonstrate that non-frontier LLMs, trained online with inference time sampling, can match or exceed the performance of frontier models on these tasks with a modest compute budget. (4) We release Aviary and our LDP framework as open-source software libraries to enable broader use and experimentation.

3 Methodology

Below we describe novel aspects of our methodology in Aviary and LDP. In Appendix B we provide background on pre-existing methods such as behavior cloning, expert iteration, and inference compute scaling that we employ for our experiments. Appendix F includes an example of an Aviary environment and rollout with an LDP agent.

3.1 Language Decision Processes

A language decision process (LDP) is a Partially-Observable Markov Decision Process (POMDP) (Åström, 1965) whose action and observation spaces are represented in natural language. More concretely, a LDP can be defined using the tuple $(\mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma)$. Here, $\mathcal{V}$ is a non-empty alphabet[1], $\mathcal{S}$ is the state space, $\mathcal{A} \subseteq \mathcal{V}^*$ is the action space[2], $T(s'|s,a) : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{P}(\mathcal{S})$ is the transition function, $R(s,a) : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{P}(\mathbb{R})$ is the reward function, $\mathcal{O} \subseteq \mathcal{V}^*$ is the observation space, $Z(o|s') : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{P}(\mathcal{O})$ is the observation function[3], and $\gamma \in [0,1]$ is the discount factor.

[1]In all LDPs we consider, $\mathcal{V}$ is the set of unicode characters, since Aviary is implemented in Python 3.

[2]Where $\mathcal{V}^* \stackrel{\text{def}}{=} \bigcup_{n=0}^{\infty} \mathcal{V}^n$ is the Kleene closure of a set $\mathcal{V}$ (Kleene, 1956; Meister et al., 2023).

[3]In all LDPs we consider, a state $s' \in \mathcal{S}$ uniquely defines an observation $o \in \mathcal{O}$. Unless otherwise specified, we omit the observation function Z.

Unlike traditional reinforcement learning agents, feedback for language agents is "grounded" in the sense that environment observations must be converted to text (Wu et al., 2024b). As such, the alphabet $\mathcal{V}$ is an important component of the LDP definition and follows other works (Carta et al., 2023; Wen et al., 2024b;a). The solution to an LDP is a policy $\pi_\theta : \mathcal{O} \mapsto \mathcal{A}$, where θ denotes the set of policy parameters we wish to learn. The parameter set θ is abstract and encapsulates any optimizable parameter of the language agent that may impact the action chosen such as LLM weights, inference hyperparameters such as temperature, as well as parametrized procedures such as internal reasoning.

In contrast to previous works which demarcate between internal and external actions (Sumers et al., 2024; Christianos et al., 2023), where internal actions include reasoning and memory retrieval, in our problem framing we consider the action space to strictly constitute interactions with the external environment, allowing our parameter set θ to subsume optimizable procedures that are internal to the language agent such as memory retrieval and internal reasoning. Practically, it is worth noting that the complexity of our environments is such that we do not expect to obtain the globally optimal π_θ^*. Our more modest goal is to be able to optimize θ in a direction that improves π_θ over time.

In all environments we consider, observations are deterministic functions of the state and so the reader may assume $Z = 1$ henceforth. For example, the environment may involve code execution where the observation consists of side-effects of the code. In this case, the state $\mathcal{S}$ includes all information necessary to induce the Markov property of the transition function such as the file system, package versions, environment variables, and hardware. However, the observation is simply the output message of the executed code.

3.2 Stochastic Computation Graphs

In the general case, a language agent may include both stochastic and deterministic operations. We build on the formalism of stochastic computation graphs (SCG) (Schulman et al., 2015): directed, acyclic graphs with nodes corresponding to computations and edges corresponding to arguments.

A deterministic node v corresponds to a function f_v, and the node's output $o(v)$ is defined as:

$$o(v) = f_v(\{o(w) \,|\, w \in \text{parents}(v)\}). \tag{1}$$

A stochastic node u is a distribution p_u, with output:

$$o(u) \sim p_u(\,\cdot \,|\, \{o(w) \,|\, w \in \text{parents}(v)\}). \tag{2}$$

Note that inputs to the SCG are treated as constant, deterministic nodes; outputs are leaf nodes.

A language agent's policy is an SCG with a string input (the observation) and string output (the action). Language agent architectures can easily be expressed as SCGs by combining deterministic and stochastic nodes. The SCGs of some common agents are provided in Appendix A.

4 Environments

Here, we provide an overview of the environments implemented in Aviary. Further details may be found in Appendix C.

4.1 GSM8K

The GSM8K environment is based on the GSM8K dataset introduced in Cobbe et al. (2021), which consists of linguistically diverse grade school math word problems designed to assess multi-step mathematical reasoning. The dataset comprises a train set (7,473 questions) and a test set (1,319). The environment exposes a calculator tool.

4.2 HotpotQA

The HotpotQA environment is based on the HotpotQA dataset introduced in Yang et al. (2018), which was subsequently extended to a language agent environment in Yao et al. (2023). The dataset comprises 112,779 question-answer pairs. We evaluate agent performance on the 7,405 question eval subset. The environment provides tools to search for and extract information from Wikipedia articles.

4.3 PaperQA

PaperQA (Lála et al., 2023; Skarlinski et al., 2024) was developed for literature research and question answering. By pairing agentic retrieval augmented generation with reranking and contextual summarization, PaperQA attained superhuman-level precision and human-level accuracy on the LitQA2 dataset (Laurent et al., 2024).

We refactored PaperQA into an Aviary environment, modifying the toolset as needed. We modified the `paper_search` tool to center on local storage of PDF, text, and HTML files and omitted the `citation_traversal` tool for this local setting. A `complete` tool was added to support agents that require at least one tool selection, and allow the agent to declare if the answer addresses all parts of the question.

For the sake of comparison against Skarlinski et al. (2024), we obtained LitQA2's private test questions through correspondence with the authors[4], and used an 80\20 split on the 199 public questions for train and eval splits.

4.4 Molecular Cloning

Molecular cloning is a fundamental technique for manipulating DNA in biomedical science, enabling basic research into gene function, transgenic models, and recombinant proteins (Bertero et al., 2017; Sharma et al., 2014). The molecular cloning process results in a DNA "construct," which is a general term for DNA that encodes for the desired biologic molecule or genes.

The molecular cloning environment comprises the main tools used by laboratory scientists: (1) an annotation tool for predicting the function of plasmid segments (2) a natural language search tool that retrieves sequences given text, and (3) tools required to plan protocols. A complete list of tools is provided in Appendix C.

The specific tasks used for evaluation come from the SeqQA benchmark (Laurent et al., 2024), which consists of textbook-style multiple-choice questions on molecular cloning. The SeqQA train set comprises 500 questions from Laurent et al. (2024) as well as 150 new test questions we introduce specifically for the Aviary SeqQA environment[5].

4.5 Protein Stability

We introduce the protein stability environment as a sandbox for training agents to integrate knowledge from physics-based models, biochemical principles, and pre-trained protein models, with the potential to leverage experimental results. We assess the language agent's performance on forty proteins randomly selected from the megascale protein stability dataset, excluding any that are mentioned in the text of Tsuboyama et al. (2023). Proposed mutations are evaluated using the Rosetta cart_ddg protocol (Frenz et al., 2020). Note that we only perform inference time evaluation of language agents on the protein stability task and as such, we do not maintain a train set.

5 Experiments

We assess the capabilities of tool-equipped language agents to solve problems in the aforementioned environments. We subsequently explore behavior cloning and expert iteration (described in Appendix B) to train agents on specific tasks in environments. Finally, we explore the effect of majority vote sampling at inference-time.

An overview of the models and their performance is provided in Figure 2. The trained (described below) and frontier language models versions are `claude-3-5-sonnet-20241022`, `gpt-4o-08-06`, and `Llama-3.1-8B-Instruct`. Our agents are:

[4]The test set is private to prevent leakage to frontier models. The test split may be made available to the reviewers upon request pending permission from the original authors.

[5]These questions are maintained in a private repository to prevent test set leakage to frontier LLMs and can be made available to the reviewers upon request.

- Zero-shot Claude 3.5 Sonnet: The LLM is prompted to solve the tasks without access to the environment. No example output or formatting instructions are given.
- Claude 3.5 Sonnet agent: A language agent prompted to call environment tools until the task is solved. It uses the Anthropic API tool-calling schema (Anthropic, 2024).
- LDP-trained language agent: An LDP agent trained to solve tasks using environment tools. This can either be based on fine-tuned GPT-4o (GSM8K, HOTPOTQA) or Llama-3.1-8B-Instruct (SeqQA, LitQA2).
- Majority voting: We sample 32 trajectories from an agent using their consensus as the task solution. For protein stability, we do oracle-verification/pass@k, as in protein engineering one typically tests a batch and only keeps the most successful (Brown et al., 2024).

The set of existing benchmarks and closed-source models were chosen to demonstrate the flexibility of the Aviary software. Claude 3.5 Sonnet was the best frontier LLM across tasks, and was thus used as the benchmark for comparison. With the exception of GSM8K, all agents improve over the zero-shot baseline when given access to the environment. In the case of GSM8K, we hypothesize that a sequence of calculator calls (with no intermediate reasoning) is out-of-distribution with respect to the LLMs' training data, which also contains math word problems. This is consistent with recent findings (Mirzadeh et al., 2024), where modifying elements of the original questions or adding irrelevant information caused performance degradation, as such changes similarly introduce a distribution shift.

Training LDP agents improves performance over untrained agents of the same architecture. On challenging tasks (SeqQA, LitQA2), a relatively small model (Llama-3.1-8B-Instruct) can be trained to match performance of a much larger frontier model (Claude 3.5 Sonnet). Majority voting yields a further large gain at the cost of increased inference compute. The protein stability task sees a large improvement for pass@16, which is a well-known effect for oracle-verified problems(Brown et al., 2024). These results are described in detail below.

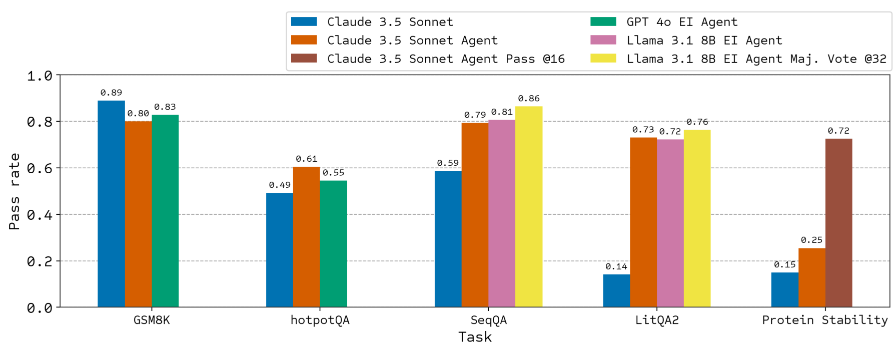


Figure 2: Ability of LLMs and language agents to solve tasks using Aviary environments. All LDP-trained agents are optimized using behavior cloning and expert iteration. For GSM8K and HOTPOTQA, EI is performed on GPT-4o; SeqQA and LitQA2 use Llama-3.1-8B-Instruct (see subsection 5.1). The difference in GSM8K zero-shot reported here (89%) vs Anthropic benchmarks (Anthropic, 2024) (96.5%) is likely because Anthropic's use of chain-of-thought prompting, which we did not use. All agents are rolled out on the environment for a maximum of 10 steps, with the exception of PaperQA (18 steps) and protein stability (20).

5.1 Behavior Cloning and Expert Iteration

Using LDP, we train language agents in the environments described in section 4. Since these environments are challenging, expert iteration initially rejects the majority of trajectories, leading to very slow learning. We therefore begin with behavior cloning, using high-quality trajectories collected by rejection-sampling from a larger LLM. Once the language agent can solve a reasonable fraction of training problems, we switch to expert iteration. All experiments are conducted with Llama-3.1-8B-Instruct (Grattafiori et al., 2024) as the base language model, using Nvidia A100 GPUs.

In Figure 3A, we show the results of training an agent (Llama-3.1-8B EI) to solve SeqQA tasks using the molecular cloning environment. Expert iteration is seeded with 2841 valid trajectories (behavior cloning), followed by 8 further EI epochs. Behavior cloning provides a large initial jump in performance, with a further 14% (absolute) improvement from EI. In Appendix E we study the distribution of trajectories explored by a trained language agent.

In Figure 3B, we show the results of a similar procedure applied to LitQA2 questions in the PaperQA environment. In this case, the untrained Llama-3.1-8B agent has non-trivial performance (30% accuracy), but still significantly improves from behavior cloning (430 trajectories). We aimed to focus training on the more difficult LitQA2 questions, so during expert iteration, we sample trajectories from each task in the dataset with probability:

$$P(\text{task } k) = \frac{w_k}{\sum_j w_j}; \quad w_k = M \cdot (1 - f^k_{\text{pass}}), \tag{3}$$

where f^k_{pass} is a moving average of task k's pass rate as the agent is trained and M is a scaling factor (set to 20). With this, EI produces a small improvement beyond behavior cloning, up to 72% on the test set.

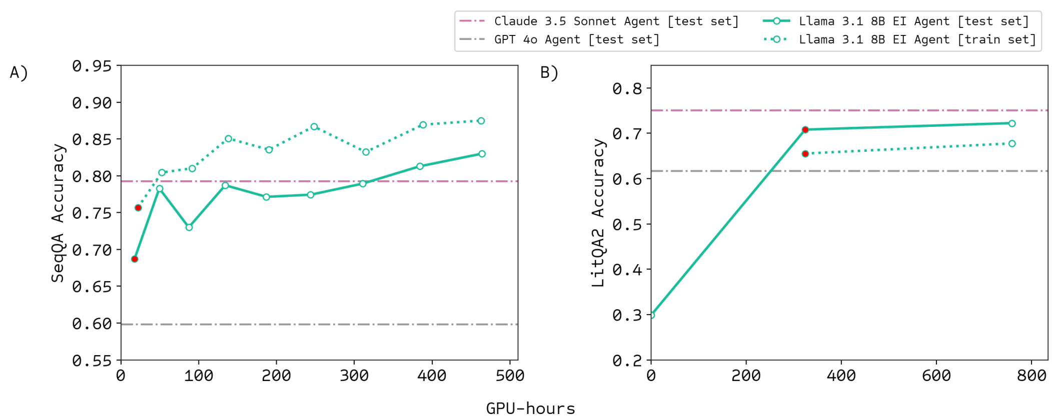


Figure 3: Training language agents to solve (A) SeqQA tasks using the molecular cloning environment and (B) LitQA2 questions using the PaperQA environment. The first epoch (red points) is behavior cloning, followed by expert iteration. These experiments use multiple workers to asynchronously collect trajectories and train the model, so `GPU-hours` measures the total time spent sampling and training. An untrained Llama-3.1-8B agent solves 1% of SeqQA tasks, so we omit the data point at `GPU-hours=0` in panel A.

5.2 Inference Compute Scaling

We assess majority voting on two sets of multiple-choice tasks: SeqQA, and LitQA2, with the results in Figure 4. Majority voting generally affords large improvements, achieving ∼20 percentage points (p.p.) of accuracy gain. Figure 4B demonstrates that non-agentic LLMs benefit as well, with a ∼10 p.p. gain for zero-shot Claude.

Figure 4C shows that majority voting on LitQA2 with a Claude 3.5 Sonnet agent reaches 89% accuracy on the test set, significantly exceeding previously reported scores of 67% from Skarlinski et al. (2024) and the human performance reported in Laurent et al. (2024). The Llama-3.1-8B EI agent performs well, matching human and previously reported best at only a single sample. Three samples exceeds those marks, but cannot match the Sonnet agent if it also uses majority voting on more than one sample. Nevertheless, exceeding a frontier LLM in the single sample setting on unseen data with a small model is a surprising result.

Figure 4A shows that majority voting with the Llama-3.1-8B EI agent significantly exceeds a Claude 3.5 Sonnet agent across all sample counts. SeqQA is more structured than LitQA2, requiring more consistent and longer tool call sequences. The Llama-3.1-8B EI agent can be sampled from cheaply, and so we run 945 rollouts (Figure 4A inset). We observe improvement up to 100s of

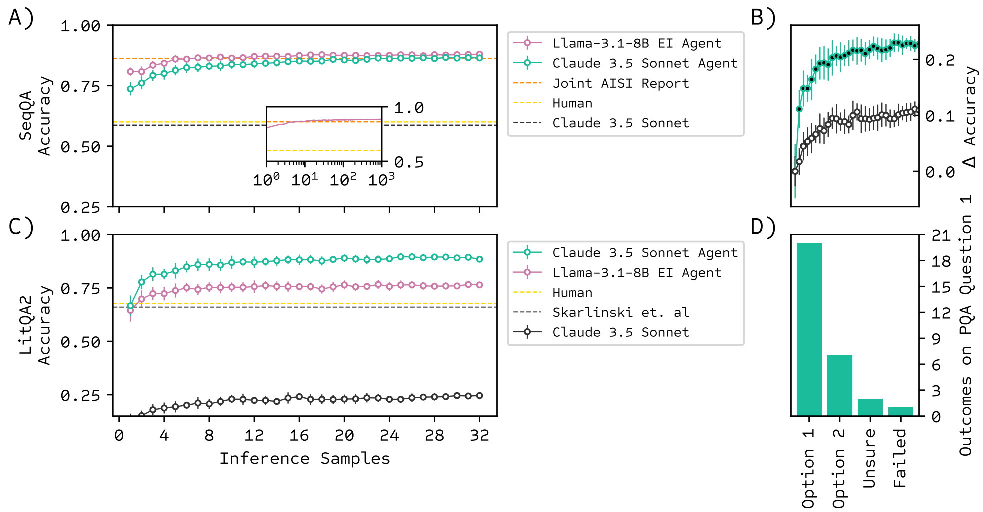


Figure 4: A) majority voting accuracy in SeqQA as a function of number of sampled trajectories. With 32 samples, performance exceeds previously-reported scaffolded agents (Joint AISI Report (US AI Safety Institute, 2024)). The "Claude 3.5 Sonnet" line refers to the zero-shot setting. The inset shows further gains from 0.86 to 0.89 from 32 to 945 samples. B) shows SeqQA improvement from majority voting with an LLM without tools vs. a language agent. C) majority voting accuracy on LitQA2. Both agents significantly exceed previously-measured human and agent performance (Skarlinski et al., 2024). Claude 3.5-Sonnet Agent plateaus at 90% accuracy. D) shows an example question voting on LitQA2 (question id `3e6d7a54`). Option 1 is the correct response, option 2 is incorrect, and failure is because the agent did not submit an answer prior to trajectory termination. Error bars are computed by bootstrap resampling.

samples, yielding a final accuracy of 89%. The highest previously reported result was from a joint technical report from the US and UK AI Safety Institutes on pre-deployment evaluation of `claude-3.5-sonnet-20241022` at 87% accuracy.

5.3 Inference Cost Scaling

The results of the previous sections demonstrate how the performance of different agents scales as training time and sampled trajectories are increased. In this section, we consider a more practical metric: inference cost. This becomes especially relevant in a high-throughput setting, in which agents are tasked to solve thousands of problems in parallel. We focus our comparison on the Claude 3.5 Sonnet agent versus the Llama-3.1-8B EI agent. At the time of writing, Claude 3.5 Sonnet's input tokens cost \$3/1M and output tokens cost \$15/1M [6]. We price Llama 3.1-8B inference at \$0.03/1M input and output tokens, typical in the LLM inference market[7], which we use as a reasonable estimate for EI-trained model inference. In Figure 5, we report performance and inference cost on SeqQA and LitQA2. While majority voting with the Claude 3.5 Sonnet agent clearly outperforms other settings, this requires $\mathcal{O}(\$1)$ per task. We reach the same SeqQA accuracy using the Llama-3.1-8B EI agent for 100x less cost. While this was not achievable for LitQA2, we note that majority voting with Llama-3.1-8B EI still exceeds single-rollout with Sonnet at 3x less cost.

6 Discussion and Limitations

We were motivated to design Aviary and LDP by (a) a need to implement language interfaces to complex scientific environments and (b) a need to optimize agents in these environments. In this work, we have focused on benchmark tasks that are easy to evaluate across five different environments. A surprising, but welcome outcome of our experiments is that trained agents based on relatively

[6] https://www.anthropic.com/pricing#anthropic-api
[7] https://lambdalabs.com/inference#pricing

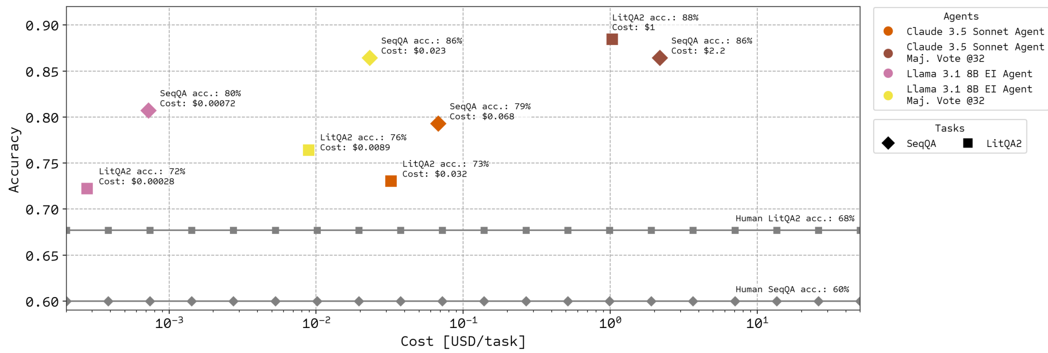


Figure 5: Accuracy vs. inference cost of two agents (Claude 3.5 Sonnet and `Llama-3.1-8B` EI) on the SeqQA and LitQA2 tasks. For both agents, the single-rollout and majority-voting settings are considered. Note that all inference settings here outperform human performance (reported in Skarlinski et al. (2024) and Laurent et al. (2024)).

small language models can compete with or even beat frontier model-based agents. While out of scope for this paper, we anticipate that future work can extend these conclusions by leveraging more sophisticated policy optimization methods (compared to EI) and inference-time scaling (compared to majority voting). To that end, we hope that LDP serves as a useful framework for experimenting with such algorithms.

Small, trained agents reduce inference costs substantially. For reference, our SeqQA tasks require 7-10 LLM calls (Figure 7) and cost $0.07 on average per trajectory with Claude 3.5 Sonnet and $0.00066 with `Llama-3.1-8B` EI. The human PhD contractors that represent the human data series in Figure 4 cost between $4 and $12 per question (see Laurent et al. (2024) for details). In summary, trained agents can exceed the accuracy of human and frontier models at 100x cheaper cost.

There are some limitations in this work. Several of our benchmarks are new or introduce new data and so we are intentionally gating access to the test sets to avoid leakage into pre-training corpora. Our human performance comparison comes with the caveat that human evaluators do not always have access to the exact same toolset. Laurent et al. (2024) gave incentives for correct answers, ample time, and only restricts the use of AI tools. Nevertheless, it is always possible that humans could have been given more expansive or precise technology for the task. Ultimately, the test of these language agents is their ability to make novel scientific discoveries and not simply to achieve high scores on benchmarks.

7 Conclusion

We have presented Aviary, a gymnasium for language agents. Aviary currently contains five environments, three of which focus on challenging scientific tasks. Language agents implemented in these environments exceed the performance of zero-shot frontier LLMs on the SeqQA, HOTPOTQA, LitQA2, and protein stability tasks. Language agents also exceed human performance on SeqQA and LitQA2.

We have introduced the language decision process (LDP) framework for formally describing language agent tasks and showed that language agents can be cast as stochastic computation graphs. Through behavior cloning, expert iteration, and inference-time sampling, we demonstrated that trained `Llama-3.1-8B` EI agents can match and exceed the performance of humans and frontier LLMs in the LitQA2 and SeqQA benchmarks at significantly lower cost. Thus, we have demonstrated that modest compute budgets and model sizes can be competitive at solving realistic scientific tasks. The reported trained `Llama-3.1-8B` EI agents are compute efficient and exceed human-level performance, enabling high-throughput automation of meaningful scientific tasks across biology.

Both the Aviary (`aviary`) and LDP (`ldp`) frameworks are open source and should serve as useful libraries for implementing environments and language agents.

REFERENCES

310.ai. 310 copilot. 2024. URL https://310.ai.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 5366–5376, 2017.

Thomas William Anthony. *Expert iteration*. PhD thesis, UCL (University College London), 2021.

Anthropic. Introducing the next generation of Claude, 2024. https://www.anthropic.com/news/claude-3-family.

Karl Johan Åström. Optimal control of Markov processes with incomplete state information I. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.

Michael Bain and Claude Sammut. A framework for behavioural cloning. *Machine Intelligence*, 15: 103–129, 1995.

Bebop. Poly: A library for DNA sequence design. https://github.com/bebop/poly, 2025. Accessed: 29-Jan-2025.

Alessandro Bertero, Stephanie Brown, and Ludovic Vallier. *Methods of Cloning*, pp. 19–39. Elsevier, 2017. ISBN 9780128030776. doi: 10.1016/b978-0-12-803077-6.00002-3. URL http://dx.doi.org/10.1016/B978-0-12-803077-6.00002-3.

Samuel R Bowman. Eight things to know about large language models. *arXiv preprint arXiv:2304.00612*, 2023.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as I can, not as I say: grounding language in robotic affordances. In *Conference on Robot Learning*, pp. 287–318. PMLR, 2023.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pp. 3676–3713. PMLR, 2023.

Roger A Chambers and Donald Michie. Man-machine co-operation on a learning task. *Computer Graphics: Techniques and Applications*, pp. 179–186, 1969.

Harrison Chase. LangChain, October 2022. URL https://github.com/langchain-ai/langchain.

Angelica Chen, Samuel D Stanton, Robert G Alberstein, Andrew M Watkins, Richard Bonneau, Vladimir Gligorijevi, Kyunghyun Cho, and Nathan C Frey. LLMs are highly-constrained biophysical sequence optimizers. *arXiv preprint arXiv:2410.22296*, 2024a.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023a.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. InstructZero: Efficient instruction optimization for black-box large language models. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=rADFNrIss3.

Xiang Chen, Ningyu Zhang, Xin Xie, Shumin Deng, Yunzhi Yao, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. Knowprompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction. In *Proceedings of the ACM Web conference 2022*, pp. 2778–2788, 2022.

Yanxi Chen, Yaliang Li, Bolin Ding, and Jingren Zhou. On the Design and Analysis of LLM-Based Algorithms. *arXiv preprint arXiv:2407.14788*, 2024c.

Yuyan Chen, Zhihao Wen, Ge Fan, Zhengyu Chen, Wei Wu, Dayiheng Liu, Zhixu Li, Bang Liu, and Yanghua Xiao. MAPO: Boosting large language model performance with model-adaptive prompt optimization. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023b. URL https://openreview.net/forum?id=paUJOst3OE.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the New AutoDiff–unlocking efficient optimization of computational workflows. *arXiv preprint arXiv:2406.16218*, 2024.

Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang. Black-box prompt optimization: Aligning large language models without model training. *arXiv preprint arXiv:2311.04155*, 2023.

Filippos Christianos, Georgios Papoudakis, Matthieu Zimmer, Thomas Coste, Zhihao Wu, Jingxuan Chen, Khyati Khandelwal, James Doran, Xidong Feng, Jiacheng Liu, et al. Pangu-Agent: A fine-tunable generalist agent with structured reasoning. *arXiv preprint arXiv:2312.14878*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-inference: exploiting large language models for interpretable logical reasoning. In *The Eleventh International Conference on Learning Representations*, 2023.

Ishita Dasgupta, Christine Kaeser-Chen, Kenneth Marino, Arun Ahuja, Sheila Babayan, Felix Hill, and Rob Fergus. Collaborating with language models for embodied reasoning. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Xuan Long Do, Yiran Zhao, Hannah Brown, Yuxi Xie, James Xu Zhao, Nancy F Chen, Kenji Kawaguchi, Michael Shieh, and Junxian He. Prompt optimization via adversarial in-context learning. *arXiv preprint arXiv:2312.02614*, 2023.

Brandon Frenz, Steven M Lewis, Indigo King, Frank DiMaio, Hahnbeom Park, and Yifan Song. Prediction of protein mutational free energy: Benchmark and sampling improvements increase classification accuracy. *Front. Bioeng. Biotechnol.*, 8:558247, October 2020.

Simon Frieder, Jonas Bayer, Katherine M Collins, Julius Berner, Jacob Loader, András Juhász, Fabian Ruehle, Sean Welleck, Gabriel Poesia, Ryan-Rhys Griffiths, et al. Data for mathematical copilots: Better ways of presenting proofs for machine learning. *arXiv preprint arXiv:2412.15184*, 2024a.

Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Petersen, and Julius Berner. Mathematical capabilities of ChatGPT. *Advances in Neural Information Processing Systems*, 36, 2024b.

Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. Large language models empowered agent-based modeling and simulation: A survey and perspectives. *arXiv preprint arXiv:2312.11970*, 2023.

Wenhao Gao, Sai Pooja Mahajan, Jeremias Sulam, and Jeffrey J. Gray. Deep learning in protein structural modeling and design. *Patterns*, 1(9):100142, December 2020. ISSN 2666-3899. doi: 10.1016/j.patter.2020.100142. URL http://dx.doi.org/10.1016/j.patter.2020.100142.

Alireza Ghafarollahi and Markus J Buehler. ProtAgents: protein discovery via large language model multi-agent collaborations combining physics and machine learning. *Digital Discovery*, 2024.

Adi Goldenzweig and Sarel J Fleishman. Principles of protein stability and their application in computational design. *Annu. Rev. Biochem.*, 87:105–129, June 2018.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, and Aiesha Letman et. al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Antoine Grosnit, Alexandre Maraval, James Doran, Giuseppe Paolo, Albert Thomas, Refinath Shahul Hameed Nabeezath Beevi, Jonas Gonzalez, Khyati Khandelwal, Ignacio Iacobacci, Abdelhakim Benechehab, et al. Large language models orchestrating structured reasoning achieve Kaggle grandmaster level. *arXiv preprint arXiv:2411.03562*, 2024.

Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran Pan, Teng Wu, Jiaqian Yu, et al. BLADE: Benchmarking language model agents for data-driven science. In *Empirical Methods in Natural Language Processing*, 2024.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=ZG3RaNIsO8.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-Agent: Automated data science by empowering large language models with case-based reasoning. *arXiv preprint arXiv:2402.17453*, 2024b.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024a.

Wenyang Hu, Yao Shu, Zongmin Yu, Zhaoxuan Wu, Xiangqiang Lin, Zhongxiang Dai, See-Kiong Ng, and Bryan Kian Hsiang Low. Localized zeroth-order prompt optimization. *arXiv preprint arXiv:2403.02993*, 2024b.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. InfiAgent-DABench: Evaluating agents on data analysis tasks. In *Forty-first International Conference on Machine Learning*, 2024c.

Kaixuan Huang, Yuanhao Qu, Henry Cousins, William A Johnson, Di Yin, Mihir Shah, Denny Zhou, Russ Altman, Mengdi Wang, and Le Cong. CRISPR-GPT: An LLM agent for automated design of gene-editing experiments. *arXiv preprint arXiv:2404.18021*, 2024a.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Forty-first International Conference on Machine Learning*, 2024b.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pp. 9118–9147. PMLR, 2022.

Quickwit Inc. tantivy, October 2024. URL https://github.com/quickwit-oss/tantivy.

John B. Ingraham, Maxim Baranov, Zak Costello, et al. Illuminating protein space with a programmable generative model. *Nature*, 623:1070–1078, 2023. doi: 10.1038/s41586-023-06728-8.

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. DISCOVERYWORLD: A Virtual Environment for Developing and Evaluating Automated Scientific Discovery Agents. In *Advances in Neural Information Processing Systems*, 2024.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *European Conference on Computer Vision*, pp. 709–727. Springer, 2022.

Zhijing Jin, Yuen Chen, Felix Leeb, Luigi Gresele, Ojasv Kamal, Zhiheng LYU, Kevin Blin, Fernando Gonzalez Adauto, Max Kleiman-Weiner, Mrinmaya Sachan, and Bernhard Schölkopf. CLadder: A benchmark to assess causal reasoning capabilities of language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=e2wtjx0Yqu.

Hamed Khakzad, Ilia Igashov, Arne Schneuing, Casper Goverde, Michael Bronstein, and Bruno Correia. A new age in protein design empowered by deep learning. *Cell Systems*, 14(11):925–939, 2023.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. *arXiv preprint arXiv:2212.14024*, 2022.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, et al. DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines. In *The Twelfth International Conference on Learning Representations*, 2024.

SC Kleene. Representation of events in nerve nets and finite automata. *Automata Studies: Annals of Mathematics Studies. Number 34*, 34:3, 1956.

Josef Laimer, Heidi Hofer, Marko Fritz, Stefan Wegenkittl, and Peter Lackner. MAESTRO–multi agent stability prediction upon point mutations. *BMC Bioinformatics*, 16(1):116, April 2015.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.

Jakub Lála, Odhran O'Donoghue, Aleksandar Shtedritski, Sam Cox, Samuel G Rodriques, and Andrew D White. PaperQA: Retrieval-augmented generative agent for scientific research. *arXiv preprint arXiv:2312.07559*, 2023.

Timothy M Lauer, Neeraj J Agrawal, Naresh Chennamsetty, Kamal Egodage, Bernhard Helk, and Bernhardt L Trout. Developability index: a rapid in silico tool for the screening of antibody aggregation propensity. *J. Pharm. Sci.*, 101(1):102–115, January 2012.

Jon M Laurent, Joseph D Janizek, Michael Ruzo, Michaela M Hinks, Michael J Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D White, and Samuel G Rodriques. LAB-Bench: Measuring capabilities of language models for biology research. *arXiv preprint arXiv:2407.10362*, 2024.

Jinyang Li, Nan Huo, Yan Gao, Jiayi Shi, Yingxiu Zhao, Ge Qu, Yurong Wu, Chenhao Ma, Jian-Guang Lou, and Reynold Cheng. Tapilot-Crossing: Benchmarking and evolving llms towards interactive data analysis agents. *arXiv preprint arXiv:2403.05307*, 2024.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Xiaoqiang Lin, Zhongxiang Dai, Arun Verma, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Prompt optimization with human feedback. *arXiv preprint arXiv:2405.17346*, 2024a.

Xiaoqiang Lin, Zhaoxuan Wu, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Use your INSTINCT: INSTruction optimization for LLMs using neural bandits coupled with transformers. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=RLENZ8pNnn.

Jerry Liu. LlamaIndex, November 2022. URL https://github.com/jerryjliu/llama_index.

Xiao Liu, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. Are LLMs capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data. *arXiv preprint arXiv:2402.17644*, 2024.

Ruotian Ma, Xiaolei Wang, Xin Zhou, Jian Li, Nan Du, Tao Gui, Qi Zhang, and Xuanjing Huang. Are large language models good prompt optimizers? *arXiv preprint arXiv:2402.02101*, 2024a.

Yubo Ma, Zhibin Gou, Junheng Hao, Ruochen Xu, Shuohang Wang, Liangming Pan, Yujiu Yang, Yixin Cao, and Aixin Sun. SciAgent: Tool-augmented language models for scientific reasoning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024b.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. DiscoveryBench: Towards data-driven discovery with large language models. *arXiv preprint arXiv:2407.01725*, 2024.

Oscar Mañas, Pietro Astolfi, Melissa Hall, Candace Ross, Jack Urbanek, Adina Williams, Aishwarya Agrawal, Adriana Romero-Soriano, and Michal Drozdzal. Improving text-to-image consistency via automatic prompt optimization. *arXiv preprint arXiv:2403.17804*, 2024.

Clara Meister, Tiago Pimentel, Gian Wiher, and Ryan Cotterell. Locally typical sampling. *Transactions of the Association for Computational Linguistics*, 11:102–121, 2023.

Grégoire Mialon, Roberto Dessi, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Roziere, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. *Transactions on Machine Learning Research*, 2023.

D. Michie, M. Bain, and J. Hayes-Michie. *Cognitive models from subcognitive skills*, chapter Chapter 5, pp. 71–99. The Institution of Engineering and Technology, 1990. doi: 10.1049/PBCE044E_ch5.

Adrian Mirza, Nawaf Alampara, Sreekanth Kunchapu, Benedict Emoekabu, Aswanth Krishnan, Mara Wilhelmi, Macjonathan Okereke, Juliane Eberhardt, Amir Mohammad Elahi, Maximilian Greiner, et al. Are large language models superhuman chemists? *arXiv preprint arXiv:2404.01475*, 2024.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

Dang Nguyen, Viet Dac Lai, Seunghyun Yoon, Ryan A Rossi, Handong Zhao, Ruiyi Zhang, Puneet Mathur, Nedim Lipka, Yu Wang, Trung Bui, et al. DynaSaur: Large language agents beyond predefined actions. *arXiv preprint arXiv:2411.01747*, 2024.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Dean A Pomerleau. ALVINN: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=WRYhaSrThy.

Guanghui Qin and Jason Eisner. Learning how to ask: Querying lms with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5203–5212, 2021.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. Tool learning with foundation models. *arXiv preprint arXiv.2304.08354*, 10, 2023.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*, 2024.

Mayk Caldas Ramos, Christopher Collison, and Andrew D White. A review of large language models and autonomous agents in chemistry. *Chemical Science*, 2024.

Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

Antonio Sabbatella, Andrea Ponti, Antonio Candelieri, Ilaria Giordani, and Francesco Archetti. A Bayesian approach for prompt optimization in pre-trained language models. *arXiv preprint arXiv:2312.00471*, 2023.

Antonio Sabbatella, Andrea Ponti, Ilaria Giordani, Antonio Candelieri, and Francesco Archetti. Prompt optimization in large language models. *Mathematics*, 12(6):929, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in Neural Information Processing Systems*, 28, 2015.

Kamal Sharma, Ajay Kumar Mishra, Vikram Mehraj, and Ganesh Selvaraj Duraisamy. Advances and applications of molecular cloning in clinical microbiology. *Biotechnology and Genetic Engineering Reviews*, 30(1):65–78, 2014. ISSN 2046-5556. doi: 10.1080/02648725.2014.921501. URL http://dx.doi.org/10.1080/02648725.2014.921501.

Roger A Sheldon and John M Woodley. Role of biocatalysis in sustainable chemistry. *Chemical reviews*, 118(2):801–838, 2018.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4222–4235, 2020.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Arnav Singhvi, Manish Shetty, Shangyin Tan, Christopher Potts, Koushik Sen, Matei Zaharia, and Omar Khattab. DSPy assertions: Computational constraints for self-refining language model pipelines. *arXiv preprint arXiv:2312.13382*, 2023.

Michael D Skarlinski, Sam Cox, Jon M Laurent, James D Braza, Michaela Hinks, Michael J Hammerling, Manvitha Ponnapati, Samuel G Rodriques, and Andrew D White. Language agents achieve superhuman synthesis of scientific knowledge. *arXiv preprint arXiv:2409.13740*, 2024.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for LLM agents. *arXiv preprint arXiv:2403.02502*, 2024.

Alessandro Sordoni, Eric Yuan, Marc-Alexandre Côté, Matheus Pereira, Adam Trischler, Ziang Xiao, Arian Hosseini, Friederike Niedtner, and Nicolas Le Roux. Joint prompt optimization of stacked LLMs using variational inference. *Advances in Neural Information Processing Systems*, 36, 2024.

Martin Steinegger and Johannes Söding. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, 35(11):1026–1028, 2017.

Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=1i6ZCvflQJ. Survey Certification.

Andrew Szot, Max Schwarzer, Harsh Agrawal, Bogdan Mazoure, Rin Metcalf, Walter Talbott, Natalie Mackraz, R Devon Hjelm, and Alexander T Toshev. Large language models as generalizable policies for embodied tasks. In *The Twelfth International Conference on Learning Representations*, 2024.

Kotaro Tsuboyama, Justas Dauparas, Jonathan Chen, Elodie Laine, Yasser Mohseni Behbahani, Jonathan J. Weinstein, Niall M. Mangan, Sergey Ovchinnikov, and Gabriel J. Rocklin. Mega-scale experimental analysis of protein folding stability in biology and design. *Nature*, 620(7973): 434–444, Aug 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06328-6. URL https://doi.org/10.1038/s41586-023-06328-6.

UK AI Safety Institute US AI Safety Institute. Pre-deployment evaluation of Anthropic's upgraded Claude 3.5 Sonnet. *Technical Report*, 2024.

Chi Wang, Xueqing Liu, and Ahmed Hassan Awadallah. Cost-effective hyperparameter optimization for large language model generation inference. In *International Conference on Automated Machine Learning*, pp. 21–1. PMLR, 2023a.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a.

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M Ni, et al. OpenR: An open source framework for advanced reasoning with large language models. *arXiv preprint arXiv:2410.09671*, 2024b.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024c. URL https://openreview.net/forum?id=22pyNMuIoa.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=1PL1NIMMrw.

Muning Wen, Cheng Deng, Jun Wang, Weinan Zhang, and Ying Wen. Entropy-regularized token-level policy optimization for large language models. *arXiv preprint arXiv:2402.06700*, 2024a.

Muning Wen, Ziyu Wan, Weinan Zhang, Jun Wang, and Ying Wen. Reinforcing language agents via policy optimization with action decomposition. *arXiv preprint arXiv:2405.15821*, 2024b.

Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems*, 36, 2024c.

Lilian Weng. LLM-powered autonomous agents. *lilianweng.github.io*, Jun 2023. URL https://lilianweng.github.io/posts/2023-06-23-agent/.

B Widrow and FW Smith. *Computer and Information Sciences*, chapter Pattern recognising control systems. Clever Hume Press, 1964.

Ruidong Wu, Fan Ding, Rui Wang, Rui Shen, Xiwen Zhang, Shitong Luo, Chenpeng Su, Zuofan Wu, Qi Xie, Bonnie Berger, Jianzhu Ma, and Jian Peng. High-resolution de novo structure prediction from primary sequence. *bioRxiv*, 2022. doi: 10.1101/2022.07.21.500999. URL https://www.biorxiv.org/content/early/2022/07/22/2022.07.21.500999.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. AvaTaR: Optimizing LLM agents for tool-assisted knowledge retrieval. *arXiv preprint arXiv:2406.11200*, 2024a.

Yue Wu, Yewen Fan, Paul Pu Liang, Amos Azaria, Yuanzhi Li, and Tom M Mitchell. Read and reap the rewards: Learning to play Atari with the help of instruction manuals. *Advances in Neural Information Processing Systems*, 36, 2024b.

Zhaoxuan Wu, Xiaoqiang Lin, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Prompt optimization with EASE? Efficient ordering-aware automated selection of exemplars. *arXiv preprint arXiv:2405.16122*, 2024c.

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

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Qinyuan Ye, Maxamed Axmed, Reid Pryzant, and Fereshte Khani. Prompt engineering a prompt engineer. *arXiv preprint arXiv:2311.05661*, 2023.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Agent Lumos: Unified and modular training for open-source language agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12380–12403, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.670. URL https://aclanthology.org/2024.acl-long.670.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

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

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. *arXiv preprint arXiv:2310.12823*, 2023b.

Yuanzhao Zhai, Tingkai Yang, Kele Xu, Feng Dawei, Cheng Yang, Bo Ding, and Huaimin Wang. Enhancing decision-making for LLM agents via step-level q-value models. *arXiv preprint arXiv:2409.09345*, 2024.

Tuo Zhang, Jinyue Yuan, and Salman Avestimehr. Revisiting OPRO: The Limitations of Small-Scale LLMs as Optimizers. *arXiv preprint arXiv:2405.10276*, 2024.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. Symbolic learning enables self-evolving agents. *arXiv preprint arXiv:2406.18532*, 2024.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

A Stochastic Computation Graphs for Selected Language Agent Architectures

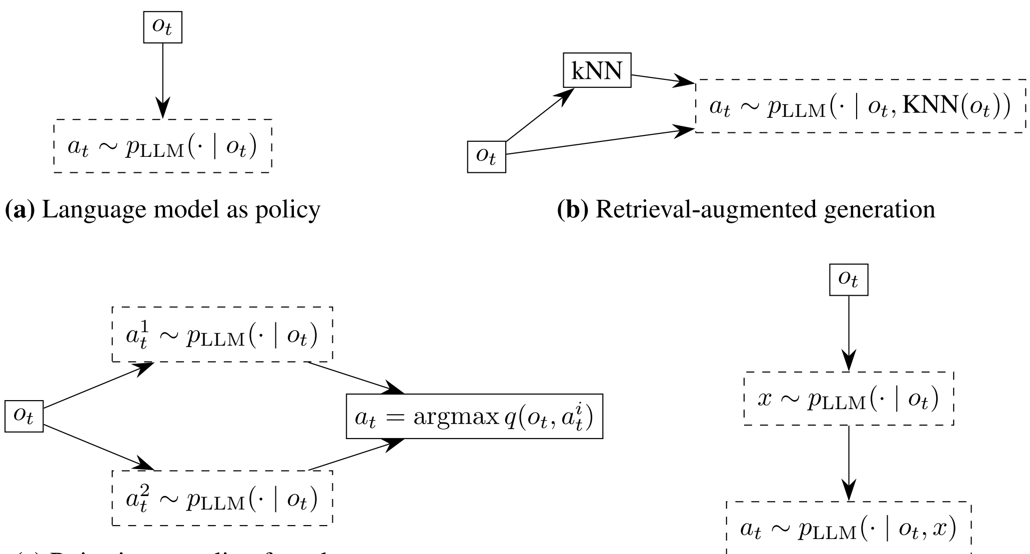
The SCGs of the following simple language agents are presented in Figure 6:

(a) Language model as policy: a single stochastic node corresponding to sampling from the language model.

(b) Retrieval-augmented generation (RAG): a deterministic node (document retrieval) leading to a stochastic node (LLM sampling).

(c) Rejection sampling from LLM: several stochastic nodes (LLM samples), all leading to a deterministic node (selecting the preferred sample).

(d) ReAct (Yao et al., 2023): two consecutive stochastic nodes, corresponding to sampling a reasoning string and an action (tool call).



(a) Language model as policy

(b) Retrieval-augmented generation

(c) Rejection sampling from language model

(d) ReAct

Figure 6: Simple language agent architectures represented as stochastic computation graphs. Deterministic nodes are solid rectangles; stochastic nodes are dashed. Note that we augment the graphs with a deterministic input node to indicate how the observation o_t is consumed.

B Training Methods

Below we describe commonly-used imitation learning (Widrow & Smith, 1964; Chambers & Michie, 1969; Pomerleau, 1988) methods employed to improve language agent performance on our environments. These training methods do not optimize the SCG graph directly, instead we optimize only the language model node in the SCG.

Behavior cloning (BC) BC (Michie et al., 1990; Bain & Sammut, 1995) refers to a general imitation learning technique that derives a policy by supervised learning on high quality trajectories termed expert demonstrations. In the context of language agents this is typically achieved by supervised fine-tuning (SFT) of an LLM on either human trajectories or trajectories generated from a stronger LLM (Christianos et al., 2023; Chen et al., 2023a; Zeng et al., 2023b; Yin et al., 2024; Song et al., 2024). In the context of our experiments, we use BC to initialize the trajectory buffer for an expert iteration loop on `Llama-3.1-8B-Instruct`, due to its inability to self-generate successful trajectories prior to training.

Expert iteration (EI) EI (Anthony et al., 2017; Anthony, 2021; Havrilla et al., 2024) performs behavior cloning in an iterative fashion, improving the demonstration data each iteration. The inputs to the EI algorithm are a base LLM represented as an initial policy π_0 and a trajectory buffer D_0, which may either be empty or consist of an initial set of demonstrations generated by a human expert or a stronger LLM. At each round of EI, first a batch B of trajectories are sampled from the current policy π_i (via rollout). Then these trajectories $\{\tau_i^{(j)}\}_{j=1}^{B}$ are filtered (the rejection sampling step (Yuan et al., 2023)) based on return R exceeding a threshold value ρ. The filtered trajectories are then appended to the trajectory buffer D_i and the current LLM, π_i, is fine-tuned on D_i using cross-entropy loss. Pseudocode for EI is provided in Algorithm 1.

Algorithm 1 Expert Iteration with Rejection Sampling Fine-Tuning

1: **Inputs**: initial policy π_0, iteration rounds N, batch size B, return threshold ρ, trajectory buffer D_0
2: **for** $i = 1, \dots N$ **do**
3: $\quad T_i \leftarrow \text{rollout}(\pi_{i-1})$
4: $\quad D_i \leftarrow D_{i-1} \cup \{(\tau_i^{(j)}, R_i^{(j)}) |\ \tau_i^{(j)} \in T_i, R_i^{(j)} > \rho, j = 1, \dots, B\}$ {Rejection sample trajectories}
5: $\quad \pi_i \leftarrow \text{SFT}(D_i)$ {SFT on updated trajectory buffer}
6: **end for**

Inference Compute Scaling Scaling inference-time compute to improve LLM performance is now a frequently-employed technique (Brown et al., 2024; Wang et al., 2023b). There are two common settings: oracle-verified (pass@k) and majority vote (consensus@k). As shown in Brown et al. (2024), if an oracle verifier can identify any correct solution – namely, if you can obtain just one correct answer among k – then it is possible to scale across multiple orders of magnitude. Without an oracle verifier, majority voting can be used (Wang et al., 2023b). Majority voting is simply the consensus response, which requires some natural binning of responses. Although oracle verification scales to very large numbers of completions (Li et al., 2022), majority voting plateaus more quickly than oracle verification (Brown et al., 2024). In this work, we omit any "unsure" or truncated trajectories (trajectories for which the agent did not submit an answer) from majority voting.

C ENVIRONMENT DETAILS

Below we expand on the details of the environments comprising Aviary including example questions, available tools, and their framing as instances of Language Decision Processes (LDPs). Environment details already mentioned in the main paper are repeated here for clarity. As a reminder, an LDP is defined as follows:

LDP $(\mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma)$

- $\mathcal{V}$ is an alphabet, a non-empty set, consisting of tokens $w \in \mathcal{V}$.
- $\mathcal{S}$ is the state space.
- $\mathcal{A} \subseteq \mathcal{V}^*$ is the action space.[a]
- $T(s'|s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{P}(\mathcal{S})$ is the transition function.
- $R(s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{P}(\mathbb{R})$ is the reward function.
- $\mathcal{O} \subseteq \mathcal{V}^*$ is the observation space.
- $Z(o|s') : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{P}(\mathcal{O})$ is the observation function.[b]
- $\gamma \in [0, 1]$ is the discount factor.

[a] Where $\mathcal{V}^* \stackrel{\text{def}}{=} \bigcup_{n=0}^{\infty} \mathcal{V}^n$ is the Kleene closure of a set $\mathcal{V}$ (Kleene, 1956; Meister et al., 2023).
[b] In all POMDPs we consider, a state $s \in \mathcal{S}$ uniquely defines an observation $o \in \mathcal{O}$. Unless otherwise stated, we will omit Z.

C.1 GSM8K

The GSM8K environment is based on the GSM8K dataset introduced in Cobbe et al. (2021), which consists of linguistically diverse grade school math word problems designed to assess multi-step mathematical reasoning. The GSM8K dataset comprises a training set of 7,473 questions and a test set of 1,319 questions. The GSM8K environment is fully observable and hence reduces to the LDP $(\mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$.

Example Task: Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn?

Tools:

1. Calculator[Expression]: Return the result of a numerical expression.
2. Submit[Answer]: Return the answer.

LDP:

- $\mathcal{V}$: Unicode characters.
- $\mathcal{S}$: GSM8K question, current step in the reasoning process.
- $\mathcal{A}$: {calculator, submit}.
- $\mathcal{T}$: The deterministic transition function.
- R : The reward function:

$$R = \begin{cases} 1 & \text{if the action submits a correct answer,} \\ -1 & \text{if the action is an invalid tool call,} \\ 0 & \text{otherwise.} \end{cases}$$

- γ : Takes on a value of 1.

C.2 HOTPOTQA

The HOTPOTQA environment is based on the HOTPOTQA dataset introduced in Yang et al. (2018), which was subsequently extended to a language agent environment in Yao et al. (2023). The HOTPOTQA dataset comprises 112,779 question-answer pairs. We run evals on the 7,405 eval subset of questions. In the HOTPOTQA environment, the agent is provided with a Wikipedia API and tasked with answering the questions. The agent is not provided with any context and the API supports full access to all articles and sections on Wikipedia.

Example Task: The Battle of Gettysburg was fought in which state and in which year?

Tools:

1. Search[Entity]: Search for an entity on Wikipedia. If a Wikipedia page exists, the first 5 sentences of the page are returned, otherwise the top-5 similar entities from the Wikipedia search engine are returned.
2. Lookup[Keyword]: Return the first sentence featuring the keyword in the current Wikipedia page, simulating Ctrl + F functionality in the web browser.
3. Finish[Answer]: Return the answer.

LDP:

- $\mathcal{V}$: Unicode characters.
- $\mathcal{S}$:{HOTPOTQA question, current Wikipedia page}.
- $\mathcal{A}$: {Search, Lookup, Finish}.
- $\mathcal{T}$: The deterministic transition function.
- R : The reward function:

$$R = \begin{cases} 1 & \text{if the action produces a correct answer,} \\ 0 & \text{otherwise.} \end{cases}$$

- $\mathcal{O}$: The current paragraph (following Search) or sentence (following Lookup) in the Wikipedia page.
- $\mathcal{Z}$: Is 1 in all cases as observations arise deterministically conditioned on the action taken.
- γ : Takes on a value of 1.

C.3 PaperQA

PaperQA2 (Lála et al., 2023; Skarlinski et al., 2024) is a language agent/environment pairing developed for scientific literature research and question answering. The environment exposes tools for full text literature search (`paper_search`), citation traversal (`citation_traversal`), reranking and contextual summarization (`gather_evidence`), and answer given ranked contextual summaries (`generate_answer`). This pairing demonstrated superhuman-level precision and human-level accuracy on version 2 of a multiple choice literature question and answering task, called LitQA2 (Laurent et al., 2024).

To implement this in Aviary, we refactored the agent/environment pairing into a standalone PaperQA environment. This environment is similar to the one used in PaperQA2, with three differences. First, to make it easy for the ML community to use, we modified the `paper_search` tool to center on local storage containing a set of PDF, text, and HTML files using tantivy (Inc., 2024). Second, the `citation_traversal` tool was omitted for this local setting. And third, a `complete` tool was added to allow the agent to declare if the answer addresses all parts of the question.

We evaluate agents on their ability to use the PaperQA environment to solve LitQA2 questions. LitQA2 features 248 questions, 199 of which are publicly available and the remaining 49 were held out as a test set. For the sake of comparison against Skarlinski et al. (2024) we obtain the private test set through correspondence with the authors[8]. The public dataset of 199 questions is randomly partitioned into an 80\20 split with 149 train questions and 40 eval questions.

To build the tantivy search indices, we (1) aggregated all paper search or citation traversal results from many PaperQA2 invocations, (2) binned the results corresponding to each LitQA2 question, and (3) combined bins based our train, evaluation, and test split LitQA2 questions. The end result is

[8]The test set is private to prevent test set leakage to frontier models. The test split may be made available to the reviewers upon request pending permission from the original authors.

18955 DOIs in the train split, 5457 DOIs in the evaluation split, and 5519 DOIs in the test split[9]. The large number of distractor papers in each split's index forms a learnable retrieval task for language agents. To avoid copyright infringement of the underlying papers, we only distribute the paper DOIs, and not the parsed texts used by the environment.

Finally, we note that our local PaperQA environment is capable of tasks beyond LitQA2. For example, it can do literature review writing and contradiction detection as reported in Skarlinski et al. (2024).

Example Task: Which base editor has been shown to be the most efficient for inducing the mutation K352E in CD45 in human T-cells?

A) ABE8e-NG
B) ABE8e–SpRY
C) SPACE-NG
D) ABE8e-SpG
E) Insufficient information to answer this question

Tools:

1. `paper_search(query: str, min_year: int | None, max_year: int | None)` - Full-text semantic search through a local search index.
2. `gather_evidence(question: str)` - Perform LLM reranking and contextual summarization given a question on `paper_search` results.
3. `gen_answer()` - Attempt to answer given the top ranked contextual summaries.
4. `complete(has_successful_answer: bool)` - Terminate using the last proposed answer, with the argument declaring whether the answer addressed all parts of the question.

LDP:

- $\mathcal{V}$: Unicode characters.
- $\mathcal{S}$:{paper chunks, metadata, ranked contextual summaries and final answer}.
- $\mathcal{A}$: {`paper_search`, `gather_evidence`, `gen_answer`, `complete`}.
- $\mathcal{T}$: Nondeterministic, as the `gather_evidence` and `gen_answer` tools rely on LLM completions.
- R :
$$\begin{cases} 1 & \text{correct answer,} \\ -1 & \text{incorrect answer,} \\ 0.1 & \text{unsure answer.} \end{cases}$$
- $\mathcal{O}$: Same as $\mathcal{S}$.
- $\mathcal{Z}$: Controlled by the stochasticity of LLM temperature sampling in `gather_evidence` and `gen_answer`.
- γ : Takes on a value of 0.9.

C.4 Molecular Cloning

Molecular cloning is a fundamental technique for manipulating DNA in biomedical science, enabling basic research into gene function, transgenic models, and recombinant proteins (Bertero et al., 2017; Sharma et al., 2014). The molecular cloning process results in a DNA "construct," which is a general term for DNA that encodes for the desired biologic molecule or genes. Molecular cloning involves assembling DNA fragments, ligating them into vectors, introducing the recombinant DNA into host organisms, and screening for desired clones (Bertero et al., 2017). The steps in molecular cloning are

[9]The train, evaluation, and test split DOIs can be made available to the reviewers upon request.

usually undertaken using a combination of human planning, specialized software, and databases of known purchasable components.

The molecular cloning environment comprises the main tools used by laboratory scientists: (1) an annotation tool for predicting the function of plasmid segments (2) a natural language search tool that retrieves sequences given text, and (3) tools required to plan protocols. Applications for protocol-specific tools include PCR primer design, ligation, codon optimization, Gibson or Golden gate assembly, and fetching genes from standard organisms. Many implementations use or are derived from the Go Poly library (Bebop, 2025). The annotation tools were built using MMSeqs2 (Steinegger & Söding, 2017).

The specific tasks used for evaluation come from the SeqQA benchmark (Laurent et al., 2024), which consists of textbook-style multiple-choice questions broken down into 15 subtasks designed to cover diverse properties of DNA, RNA, and protein sequences, as well as common tasks in molecular biology workflows such as restriction digests and polymerase chain reactions (PCR). The SeqQA train set comprises 500 questions from Laurent et al. (2024) as well as 150 new test questions we introduce specifically for the Aviary SeqQA environment[10]. The SeqQA task is solvable using only a subset of the tools, and the tools have *not* been engineered specifically for the conventions of SeqQA. For example, SeqQA questions assume 1-indexing, but the tools are 0-indexed and thus the language agent needs to learn to convert indices. Another example illustrating that that the tools are not engineered specifically for SeqQA is that SeqQA only considers coding open reading frames, but the tools may consider both reading frames.

In the molecular cloning environment, it is worth nothing that the combination of tools for manipulating DNA constructs, annotating sequences or plasmids, and searching for DNA components in databases facilitates tasks beyond SeqQA. The environment also supports working with "CloningScenarios," which is a multiple-choice benchmark for working with DNA constructs derived from real laboratory notebooks (Laurent et al., 2024). One can also perform more normative plasmid tasks, an example prompt being, "Clone the given protein [protein] into [plasmid] to express in yeast with a GFP fusion (check annotations above, plus GFP in correct relative orientation)."

Example Task:

Which of the following RNA sequences contains an ORF that is most likely to have high translation efficiency in a human cell?

A) [RNA sequence 1]
B) [RNA sequence 2]
C) [RNA sequence 3]
D) [RNA sequence 4]
E) Insufficient information to answer this question

Tools:

1. `search(query: str)` - Search Plasmid and NCBI nucleotide databases.
2. `annotate(sequence: Sequence)` - Annotate proteins, ORFs, restriction sites in a DNA sequence.
3. `gibson(sequences: list[Sequence])` - Simulate gibson assembly.
4. `goldengate(sequences: list[Sequence], enzyme: str)` - Simulate golden gate assembly.
5. `simulate_pcr(sequence: Sequence, forward_primer: Sequence | None, forward_primer_name: str | None)` - Simulate polymerase chain reaction.
6. `optimize_translation(sequence: Sequence, cg_content: int, codon_table: int, min_repeat_length: int)` - Codon optimization.
7. `separate(sequences: list[Sequence])` - Simulate gel electrophoresis.

[10]These questions are maintained in a private repository to prevent test set leakage to frontier LLMs and can be made available to the reviewers upon request.

8. `enzyme_cut(sequence: Sequence, enzyme: str)` - Simulate restriction digest.
9. `search(query: str)` - Search Plasmid and NCBI nucleotide databases.
10. `annotate(sequence: Sequence)` - Annotate proteins, ORFs, restriction sites in a DNA sequence.
11. `gibson(sequences: list[Sequence])` - Simulate gibson assembly.
12. `goldengate(sequences: list[Sequence], enzyme: str)` - Simulate golden gate assembly.
13. `simulate_pcr(sequence: Sequence, forward_primer: Sequence | None, forward_primer_name: str | None)` - Simulate polymerase chain reaction. Primers can be sequence (by ref) or name of enzyme or sequence value.
14. `optimize_translation(sequence: Sequence, cg_content: int, codon_table: int, min_repeat_length: int)` - Codon optimization.
15. `separate(sequences: list[Sequence])` - Simulate gel electrophoresis.
16. `enzyme_cut(sequence: Sequence, enzyme: str)` - Simulate restriction digest.
17. `find_sequence_overlap(sequence1: Sequence, sequence2: Sequence, reverse: bool)` - Find overlapping regions between two sequences.
18. `find_orfs(sequence: Sequence, min_length: int, codon_table: int, strand: int)` - Find open reading frames in a DNA sequence.
19. `design_primers(sequence: Sequence, target_tm: float, forward_overhang_name: str, reverse_overhang_name: str)` - Design PCR primers for a sequence.
20. `merge(sequences: list[Sequence])` - Combine multiple sequences, needed to do assembly simulation.
21. `add(sequence1: Sequence, sequence2: Sequence)` - Add two sequences together.

Tools Continued:

22. `slice_sequence(sequence: Sequence, start: int, end: int, name: str)` - Extract a subsequence.
23. `view_translation(sequence: Sequence)` - View the amino acid translation of a DNA sequence.
24. `view_sequence_stats(sequence: Sequence)` - View sequence statistics.
25. `view_restriction_sites(sequence: Sequence)` - View restriction enzyme cut sites.
26. `view_sequence(sequence: Sequence)` - View the raw sequence.
27. `submit_answer(answer: str)`

LDP:

- $\mathcal{V}$: Unicode characters.
- $\mathcal{S}$: A set of reference DNA sequences, provided by the task or obtained from previous actions. A protocol with the steps taken so far. Current reward.
- $\mathcal{A}$: A text sequence generated using the vocabulary, which should specify a tool to use and its parameters, though not all sequences will necessarily form valid or usable actions.
- $\mathcal{T}$: A transition function that executes the selected action and as a result can add more reference DNA sequences, extend the protocol, or propose a final answer.
- R :
$$\begin{cases} 1 & \text{correct answer,} \\ -1 & \text{incorrect answer,} \\ 0.1 & \text{unsure answer.} \end{cases}$$
- $\mathcal{O}$: A list of names and statistics of sequences in $\mathcal{S}$.
- $\mathcal{Z}$: 1.0 in all cases as observations arise deterministically from the state.
- γ : Takes on a value of 1.0.

C.5 Protein Stability

Engineering proteins with increased stability is a crucial task in protein engineering, with broad applications in enzyme engineering and drug design (Sheldon & Woodley, 2018). Protein engineering remains challenging, however, due to the complex interplay of sequence, structure, and biological factors (Goldenzweig & Fleishman, 2018). Integrating sequence and structure-based methods, including tools such as Rosetta, affords a more comprehensive pipeline for improving protein stability (Laimer et al., 2015).

We introduce the protein stability environment as a framework for training agents that can effectively integrate knowledge from physics-based models, biochemical principles, and pre-trained protein models with the potential to leverage experimental results to improve protein stability. The protein stability environment consists of tools commonly used by human experts for analyzing protein sequences and structures, including (1) a biochemical description tool to identify residue bond types, (2) a sequence property tool to calculate molecular weight, aromaticity, instability index, isoelectric point, sequence charge, and hydropathy, (3) a secondary structure annotation tool, and (4) a Rosetta-based tool to calculate the aggregation propensity scores per residue (Lauer et al., 2012). We assess the language agent's performance on 40 proteins randomly selected from the megascale protein stability dataset, excluding any that are mentioned in the text of Tsuboyama et al. (2023). Proposed mutations are evaluated using the Rosetta cart_ddg protocol (Frenz et al., 2020). Note that we only perform inference time evaluation of language agents on the protein stability task and as such, we do not maintain a train set.

Prior work on utilizing LLMs for protein design include 310.ai (2024), which introduced a chat-based interface for protein design, and ProtAgents (Ghafarollahi & Buehler, 2024), a multi-agent platform for protein design that integrates deep learning models trained on protein structure data (Ingraham et al., 2023; Wu et al., 2022) with physics-based simulations. LLMs have also proven effective as biological sequence optimizers (Chen et al., 2024a).

Example Task: Design at least 3 mutations and a maximum of 7 mutations to the protein sequence MKVMIRKTATGHSAYVAKKDLEELIVEMENPALWGGKVTLANGWQLEL-PAMAADTPLPITVEARKL that would improve its stability. The sequence of this protein is provided in the text file located at {input_txt_path}, and the structure of the protein can be found in the PDB file located at {input_pdb_path}.

Tools:

1. `get_bond_types_between(residues: list[int], bond_type: str)` - Describes all instances of the specified bond type among a given list of residues as outlined in the function description.
2. `get_secondary_structure(pdb_string: str)` - Describes secondary structure elements found in the protein structure by residue.
3. `get_sequence_properties(mutations: list[str], return_wt: bool)` - Describes properties like instability index, molar extinction coefficient, fraction of charged residues, iso-electric point.
4. `get_distance_between_residues(mutation: list[str])` - Get pairwise distances between list of residues.
5. `get_residue_at_position(residues: list[int])` - Returns the residue present at a specific position and describes whether it is acidic or basic or charged, polar or aliphatic or aromatic.
6. `get_hydrophobicity_score(local_pdb_file: str)` - Calculates aggregation propensity by residue using Rosetta.
7. `get_mutant_protein_sequence(mutations: list[str])` - Returns the sequence of the protein after the mutations are applied to the sequence.
8. `complete(mutations: list[str])` - Terminate after proposing mutations to the protein sequence

LDP:

- $\mathcal{V}$: Unicode characters.
- $\mathcal{S}$: path to a .pdb file containing the protein structure renumbered from 1, path to a .txt file containing the protein sequence and list of mutations proposed.
- $\mathcal{A}$: {`get_bond_types_between`, `get_secondary_structure`, `get_sequence_properties`, `get_distance_between_residues`, `get_residue_at_position`, `get_hydrophobicity_score`, `get_mutant_protein_sequence`, `complete`}
- $\mathcal{T}$: A transition function that executes the selected action and as a result can update the proposed stabilizing mutations to the protein sequence.
- R :

$$\begin{cases} 1 & \text{if Rosetta } \Delta\Delta G < 0, \\ 0 & \text{otherwise.} \end{cases}$$

- $\mathcal{O}$: sequence or structure descriptions of wild type sequence or proposed mutations
- $\mathcal{Z}$: 1.0 in all cases as observations arise deterministically from the state.
- γ : Takes on a value of 1.0.

D FURTHER RELATED WORK

Below we discuss further related work on language agent optimization frameworks and benchmarks.

Language Agent Optimization Frameworks Continued Amongst methods that jointly optimize language agent components, the TextGrad framework (Yuksekgonul et al., 2024) backpropagates feedback received from an LLM. Zhou et al. (2024) also backpropagate textual feedback by creating natural language simulacra of weights, losses, and gradients. Hu et al. (2024a) uses a metaprompt to encourage an LLM to perform discrete optimization of an agent architecture. The OptoPrime optimizer in the Trace framework (Cheng et al., 2024) passes code execution traces and uses an LLM to perform updates. DSPy (Khattab et al., 2022; Singhvi et al., 2023; Khattab et al., 2024) parametrizes a computational graph for language agents and automatically generates useful demonstrations for in-context learning. In the multi-agent setting, GPTSwarm (Zhuge et al., 2024) introduces a computation graph and performs binary edge-level optimization and node-level optimization over prompts. Lastly, OpenR (Wang et al., 2024b) is a framework for LLM reinforcement learning and inference-time scaling, but is targeted at token-level optimization, not tool usage.

Language Agent Benchmarks Existing language agent benchmarks feature a broad range of applications including machine learning tasks (Huang et al., 2024b), data science (Guo et al., 2024b; Grosnit et al., 2024), data analysis (Hu et al., 2024c; Li et al., 2024), quantitative reasoning (Liu et al., 2024), and causal reasoning (Jin et al., 2023). In Aviary, we place particular focus on scientific tasks. Relevant work in this area has included DiscoveryBench, a benchmark for data-driven hypothesis generation (Majumder et al., 2024), ChemBench (Mirza et al., 2024) which focuses on chemistry tasks, BLADE (Gu et al., 2024) which is concerned with data-driven science, SciAgent (Ma et al., 2024b) a benchmark for scientific reasoning, DISCOVERYWORLD (Jansen et al., 2024) which concentrates on cycles of scientific discovery, and ScienceWorld (Wang et al., 2022) which is concerned with scientific reasoning. For a review focused on scientifically-relevant agents the reader is directed to Ramos et al. (2024). In Aviary, we focus on sequential decision-making tasks that necessitate multiple steps of agent-environment interactions. We construct environments from the pre-existing datasets such as GSM8K (Cobbe et al., 2021), HOTPOTQA (Yang et al., 2018), and LitQA2 (Skarlinski et al., 2024) by casting them as parametrizable tools manipulating an environment state.

E DISTRIBUTION OF TRAINED LANGUAGE AGENT TRAJECTORIES

In Figure 7, we study the distribution of SeqQA trajectories explored by a trained language agent in a Sankey diagram of the tool call patterns. The demonstration trajectories (all successful) heavily feature assembly simulations and are relatively long. The trained agent was initially cloned from the demonstrations, but through online learning discovered significantly different ways to solve SeqQA tasks. The agent's trajectories are generally shorter and less diverse, suggesting that self-training tends to converge on a subset of possible paths.

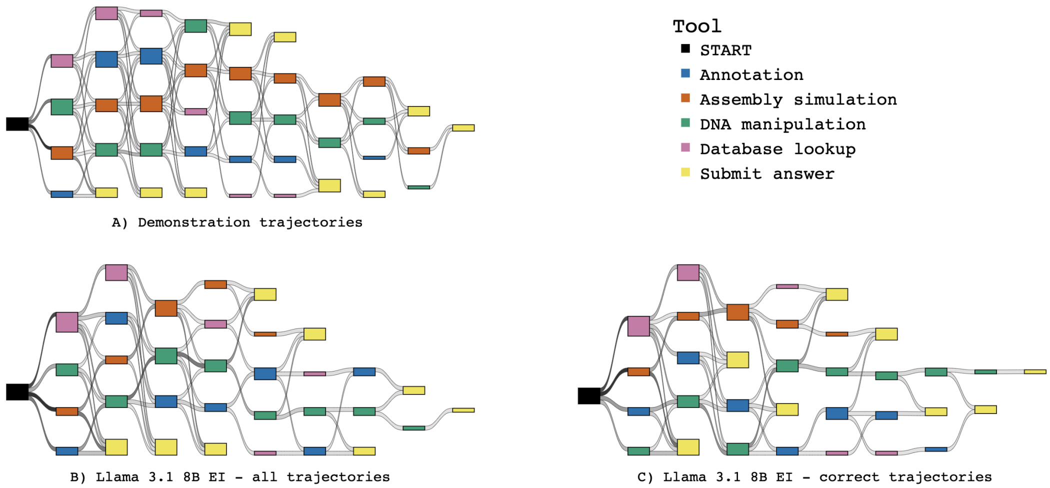


Figure 7: Patterns of tool calls across trajectories. Colored boxes represent different tool categories, and edges between boxes represent consecutive actions taken in a trajectory, with darker edges implying a greater number of trajectories. In panel (A), we show the demonstration trajectories used for behavior cloning. Panels (B) and (C) show the trajectories sampled from the `Llama-3.1-8B` EI agent after expert iteration.

F CODE EXAMPLES

Below is an example of a simple Aviary environment that maintains an integer counter. More realistic and complex examples are provided in the codebase and associated documentation.

```python
from collections import namedtuple
from aviary.core import Environment, Message, ToolRequestMessage, Tool

# State in this example is simply a counter
CounterEnvState = namedtuple('CounterEnvState', ['count'])

class CounterEnv(Environment[CounterEnvState]):
    """A simple env that allows an agent to modify a counter."""

    async def reset(self):
        self.state = CounterEnvState(count=0)
        self.tools = [
            # Parse signatures and docstrings
            # into Tool objects housing tool schemae
            Tool.from_function(self.incr),
            Tool.from_function(self.decr),
        ]
        return [Message(content=f"counter={self.state.count}")], self.tools

    async def step(self, action: ToolRequestMessage):
        obs = self.exec_tool_calls(action)
        reward = self.state.count ** 2
        # Returns observations, reward, done, truncated
        return obs, reward, reward < 0, False

    def incr(self):
        """Increment the counter."""
        self.state.count += 1
        return f"counter={self.state.count}"

    def decr(self):
        """Decrement the counter."""
        self.state.count -= 1
        return f"counter={self.state.count}"
```

Building on this, below is an example of a simple LDP agent illustrating how to sample trajectories in an environment.

```python
from ldp.agent import Agent
from ldp.graph import LLMCallOp
```

```python
from ldp.runners import RolloutManager

class AgentState:
    def __init__(self, messages, tools):
        self.messages = messages
        self.tools = tools

class SimpleAgent(Agent):
    def __init__(self, **kwargs):
        self.llm_call_op = LLMCallOp(**kwargs)

    async def init_state(self, tools):
        return AgentState([], tools)

    async def get_asv(self, agent_state, obs):
        action = await self.llm_call_op(
            config={"model": "gpt-4o", "temperature": 0.1},
            msgs=agent_state.messages + obs,
            tools=agent_state.tools,
        )
        new_state = AgentState(
            messages=agent_state.messages + obs + [action],
            tools=agent_state.tools,
        )
        # Return action, state, value (hence get_asv)
        return action, new_state, 0.0

agent = SimpleAgent(config={"model": "my_llm_endpoint"})
runner = RolloutManager(agent=agent)
trajectories = await runner.sample_trajectories(
    environment_factory=CounterEnv,
    batch_size=2,
)
```