`diff` History for Neural Language Agents

Ulyana Piterbarg [1] **Lerrel Pinto** [1] **Rob Fergus** [1]

Abstract

Neural Language Models (LMs) offer an exciting solution for general-purpose embodied control. However, a key technical issue arises when using an LM-based controller: environment observations must be converted to text, which coupled with history, results in long and verbose textual prompts. As a result, prior work in LM agents is limited to restricted domains with small observation size as well as minimal needs for interaction history or instruction finetuning. In this paper, we introduce `diff` history, a simple and highly effective solution to these issues. By applying the Unix `diff` command on consecutive text observations in the interaction histories used to prompt LM policies, we can both abstract away redundant information and focus the content of textual inputs on the salient changes in the environment. On NetHack, an unsolved video game that requires long-horizon reasoning for decision-making, LMs tuned with `diff` history match state-of-the-art performance for neural agents while needing $1800\times$ fewer training examples compared to prior work. Even on the simpler BabyAI-Text environment with concise text observations, we find that although `diff` history increases the length of prompts, the representation it provides offers a 25% improvement in the efficiency of low-sample instruction finetuning. Further, we show that `diff` history scales favorably across different finetuning dataset sizes. We open-source our code and data to https://diffhistory.github.io.

1. Introduction

Scale is a key driver of performance in classical supervised learning (Kaplan et al., 2020). However, when it comes to supervised learning for sequential decision-making, the picture is more murky: in certain domains, model improvement has been shown to scale log-linearly and sub log-linearly with task-specific demonstration count (Tuyls et al., 2023; Piterbarg et al., 2023). Sequential decision-making data is cumbersome and expensive to collect for most domains, especially for real-world problems, rendering sheer data scaling an untenable solution to improving neural agents.

From the cognitive science literature, we know that humans and animals leverage abstraction to rapidly master new behaviors from limited demonstrations and interactions (Gopnik et al., 2004; Tenenbaum et al., 2011; Allen et al., 2020). Many recent works including "Scratchpad," "Chain-of-Thought," and "Tree-of-Thought" have looked to abstraction as a means of improving the downstream generalization of neural language models (LMs) on complex tasks (Nye et al., 2021; Wei et al., 2022; Yao et al., 2023; Besta et al., 2023; Wu et al., 2023). However, the abstractions behind these works are task-specific.

We propose `diff` history, a method that introduces a highly effective and *task-agnostic* abstraction into LM prompts in order to improve the quality of text generation for sequential decision-making. When finetuning LMs on demonstrations of interaction sequences consisting of interleaved environment observations and expert actions, we substitute *full-text* natural language observations for symbolically annotated but human-interpretable *delta* observations. We refer to such an interaction demonstration sequence as a `diff` history. Computing a `diff` history is easy: just run the Unix command `diff` (Kernighan & Pike, 1984) between consecutive pairs of past full-text observations. During inference, we iteratively prompt LMs to generate natural language actions conditioned on recent `diff` history at each timestep of environment interaction.

We conduct a series of experiments testing instruction finetuning with `diff` history in two different environments: the low-dimensional, multi-task BabyAI-Text environment (Carta et al., 2023) and the high-dimensional, as-of-yet unsolved video game NetHack (Küttler et al., 2020). Both of these environments are procedurally-generated, enabling us to directly probe the generalization of neural language agents in our study. We find that `diff` history has several benefits:

[1]Dept of Computer Science, Courant Institute, NYU. Correspondence to: Ulyana Piterbarg <up2021@cims.nyu.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

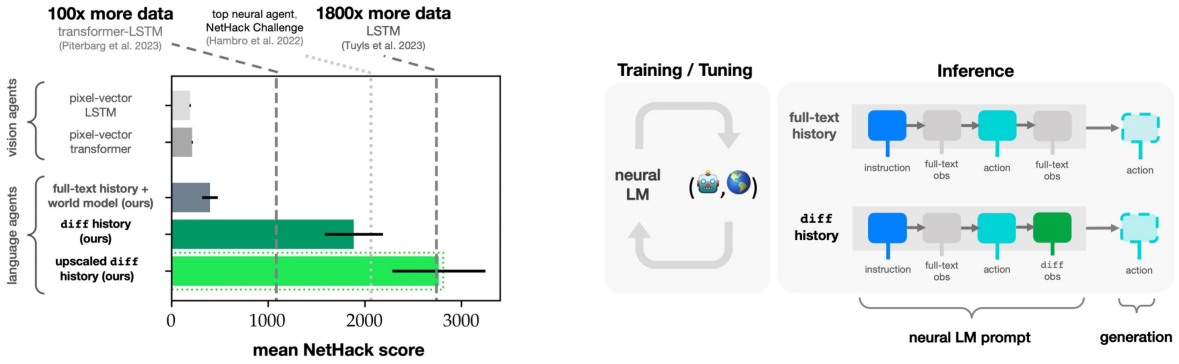

Figure 1. **Instruction finetuning LMs for sequential decision-making with `diff` history**: We summarize our key results on the unsolved video-game NetHack (left) and illustrate our proposed approach (right). Neural language agents with `diff` history match state-of-the-art performance on NetHack while training on orders of magnitude fewer data. `diff` history improves the quality of generations in low-resource decision-making domains by replacing full-text observations in interaction history data provided in-context to LMs as "memory" with richly structured *deltas*. These deltas are computed using an off-the-shelf implementation of the Unix `diff` operation.

1. It summarizes relational changes between text observations at consecutive timesteps of interaction, providing a dense learning signal for finetuning (Figure 2, Section 3.3, Appendix E).

2. It can function as a "soft compression" mechanism for interaction histories in high-dimensional environments, providing an increase in the memory horizon available to language agents (Section 3.4, Section 4.1.3).

3. In BabyAI-Text, neural language agents with full-text history require 33% more gradient updates to match the performance of `diff` history agents (Section 4.2.1).

4. In NetHack, neural language agents with `diff` history attain state-of-the-art neural agent performance despite finetuning on 1800× fewer data (Figure 1). Ablating `diff` observations from finetuning reduces neural language agent score on NetHack by 98%, which suggests that `diff` history is responsible for the highly efficient representation learning that we observe (Section 4.2.1). The performance of `diff` history agents scales with data (Section 4.4).

2. Related Work

Reasoning with Language Models: The recent successes of large LMs are difficult to understate. However, even the largest-scale and best-performing models continue to struggle with generalizable, interactive reasoning at inference-time. This is particularly true for more complex, longer-horizon, and multi-step tasks (Gemini Team, 2023; OpenAI, 2023; Chowdhery et al., 2023; Touvron et al., 2023; Rae et al., 2022; Brown et al., 2020; Radford et al., 2019). Though supervised finetuning has been demonstrated to improve LM reasoning in certain natural language settings, the problem persists (Mishra et al., 2022; Zhou et al., 2022; Lester et al., 2021; Chung et al., 2022; Wang et al., 2023b).

Several methods have looked to abstraction to address these issues. Works like "Scratchpad" show that adding hand-engineered annotations to explicitly train language models to reason step-by-step during inference can improve the quality of generations in multi-step tasks, increasing the cost of inference but requiring no extra demonstration data (Nye et al., 2021; Li et al., 2022).

Other methods including "Chain-of-Thought" and "Tree-of-Thought" demonstrate that a well-designed iterative prompting schema can be used *in lieu of finetuning* to systematize reasoning in certain environments (Wei et al., 2022; Yao et al., 2023; Besta et al., 2023), improving the performance of neural language agents without gradient updates in dodmains like the video game Minecraft (Wu et al., 2023).

We draw inspiration from both of these lines of work in formulating our method. However, the approach we propose is task-agnostic and requires no hand-crafting, human annotation, or prompt-tuning to improve the quality of generations. Moreover, we extensively test our method on a task that is partially-observable, stochastic, infinite-horizon, and has no compactly summarizable rule-set (Küttler et al., 2020; Hambro et al., 2022a), unlike Minecraft (Wu et al., 2023).

Natural Language for Supervised Decision-Making: Many recent works interface natural language and LMs with sequential decision-making.

Peng et al. (2023) query humans for descriptions of robot learning tasks and use language models to translate these into task-relevant abstractions for a state-based policy, while Ahn et al. (2022) employ LMs as high-level controllers over low-level skill policies to ground natural language in robot

affordances. Jiang et al. (2023) incorporate text and pixel inputs to train transformer-based vision-language models with supervised learning for table-top manipulation tasks, interleaving tokens across modalities during training. Similarly, Brohan et al. (2023) finetune large vision-language models on robot control datasets, expressing robot actions as text.

Finally, most similar to our work, Li et al. (2022) encode environment goal, history, and visual observations into embeddings, average, and use the results as input to tune a pretrained LM for decision-making in the BabyAI and VirtualHome environments (Chevalier-Boisvert et al., 2018). However, both of these settings are low-dimensional compared to the general decision-making settings for which we look to equip language agents in this paper, like NetHack.

BabyAI-Text: BabyAI-Text is a fully natural language-based and multi-task gridworld environment introduced by Carta et al. (2023). It is based on the BabyAI environment developed by Chevalier-Boisvert et al. (2018).

Each instance of the environment requires an agent to complete a task belonging to one of five classes; to solve each task, agents must interact with procedurally-generated objects and rooms (see Appendix C.1). There has been extensive work studying decision-making in BabyAI (Kirk et al., 2021; Reed et al., 2022; Li et al., 2022).

NetHack: NetHack is an extremely long horizon and open-ended "dungeon-crawler" video game. Games typically span from thousands to hundreds of thousands of keypresses, but may go on indefinitely as long as the player avoids death. It is also extremely complex, with players needing to explore and navigate through a series of interlinked "dungeon" levels with layout, objects, and monsters randomly regenerated in each instance of the game (Küttler et al., 2020; Hambro et al., 2022a).

As a result, NetHack is very difficult for humans to play, let alone for language or vision agents. Hand-crafted symbolic policies continue to outperform state-of-the-art neural agents for NetHack by a large margin, including those trained on hundreds of billions of keypresses of expert gameplay (Hambro et al., 2022a;b; Zhong et al., 2022; Klissarov et al., 2023; Piterbarg et al., 2023; Tuyls et al., 2023).

We provide an extensive description of NetHack and the NetHack Learning Environment in Appendix C.2.

3. diff History

We look to improve the downstream performance of neural LMs that are trained and/or finetuned for sequential decision-making. Training language models on text-based data may constitute a challenging learning problem, especially in low-sample and high-dimensional settings, where full-text environment observations may be verbose, complex, and information-sparse. To address this, we propose diff history. In this section, we provide a formalism for diff history, details on computing it, and a discussion on training LMs with diff history.

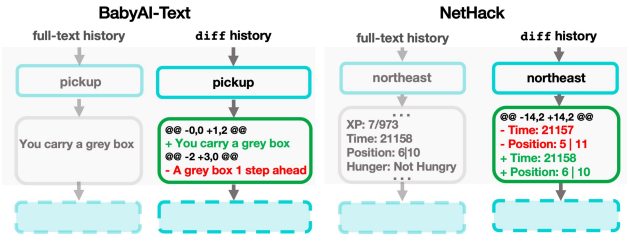

Figure 2. **A closer look at the text deltas produced by diff**: we provide examples of full-text and diff observations in our two environments of interest, BabyAI-Text and NetHack. A more detailed example of Unix-style diff outputs can be found in Appendix A.

3.1. Text-Based Decision-Making

We operate in the text-based imitation learning setting in this work, where we have access to datasets of demonstration data annotated with action labels. Concretely, our training datasets $\mathcal{D}$ consists of sequences of observations and expert actions across contiguous timesteps of interaction, paired with an *instruction* describing the task/goal. A formal expression for $\mathcal{D}$ is provided in Equation 1.

$$\mathcal{D} = \left\{\{g^i, (o_1^i, a_1^i, \ldots o_H^i, a_H^i)\}\right\}_{i=1}^{N} \tag{1}$$

We refer to the interleaved observation and action sequences in $\mathcal{D}$ as *interaction histories*. In our experiments, we train language agents on interaction histories reflecting a fixed, maximal horizon, H.

Because our data are fully text-based, individual instructions, observations, and actions may correspond to multiple tokens. As a result, our finetuning problem is *sequence-to-sequence*: our goal is train neural language agents to effectively model distributions of expert action tokens conditioned on both instruction tokens and interaction history tokens representing some $h : 1 \leq h \leq H$ previous timesteps.

From a practical standpoint, neural language agents will need to effectively use long histories ($h \gg 1$) to model such distributions in sequential decision-making settings that reveal only partial information at individual timesteps, i.e. partially-observable Markov decision processes (Kaelbling et al., 1998).

Let a_t^i be an expert action in $\mathcal{D}$ and let

$(o^i_{t-h+1}, a^i_{t-h+1}, \ldots, o^i_t)$ represent h previous timesteps of interaction reflecting the instruction g^i. We can write the tokenized version of this interaction history as shown in Equation 2.

$$(\boldsymbol{\omega}^i_{t-h+1}, \boldsymbol{\alpha}^i_{t-h+1}, \ldots, \boldsymbol{\omega}^i_t) \tag{2}$$

In the expression above, $\boldsymbol{\omega}^i_j$ and $\boldsymbol{\alpha}^i_j$ denote the token subsequences corresponding to the observation o^i_j and the action a^i_j, respectively. Using this notation, a neural LM agent π_θ with parameters θ that models sequences of action tokens can be represented as shown below, with the additional symbol $\boldsymbol{\gamma}^i$ representing the tokenized instruction g^i.

$$\boldsymbol{\alpha}^i_t \sim \pi_\theta\left(\boldsymbol{\gamma}^i, \boldsymbol{\omega}^i_{t-h+1}, \boldsymbol{\alpha}^i_{t-h+1}, \ldots, \boldsymbol{\omega}^i_t\right) \tag{3}$$

Parameters θ can be trained or tuned with teacher-forcing (Williams & Zipser, 1989) by minimizing the cross-entropy loss between predicted next-token logits and one-hot vectors representing "ground truth" expert action tokens. We provide a formal and complete definition for this loss function in Appendix A.2, Equation 8.

3.2. Unix `diff`s and Text Deltas

Given a pair of text strings, the Unix operation `diff` matches the contents of each string line-by-line by treating each line as a separate substring. It then computes and returns a symbolically annotated summary of any detected differences between the text strings, enumerated line-by-line (Kernighan & Pike, 1984).

The `diff` operator is implemented in control systems like `git` and integrated development environments like Visual Studio to help software engineers efficiently track and visualize changes between versions of text files. A detailed example of the outputs and default notation employed by Unix-style `diff` is provided in Appendix A.1.

3.3. Computing `diff` history for Decision-Making

Naively, without `diff` history, one might train a model to predict text actions a^i_t for an instruction g^i given an interaction history $(o^i_{t-h+1}, a^i_{t-h+1}, \ldots, o^i_t)$ consisting of past full-text observations and actions tokenized as in Equation 2. We refer to such an interaction history as a *full-text history*.

With `diff` history, we substitute the full-text observations that follow the first (or "oldest") timestep in the history with text deltas Δo^i_l by computing Unix `diff`'s between consecutive pairs of full-text observations as in Equation 4.

$$\Delta o^i_l \coloneqq \texttt{diff}(o^i_{l-1}, o^i_l),\ \forall l : t-h < l \leq t \tag{4}$$

This yields an interaction history that is equivalent to the original (Equation 5).

$$(o^i_{t-h+1}, a^i_{t-h+1}, \Delta o^i_{t-h+2}, a^i_{t-h+2}, \ldots, \Delta o^i_t) \tag{5}$$

The single, full-text observation o^i_{t-h+1} acts as an "anchor" against which subsequent deltas are computed. We refer to interaction histories that are processed using the Unix `diff` operation in this manner as **`diff` histories**. To parallel Equation 2, we can write the tokenized version of a `diff` history as in Equation 6.

$$(\boldsymbol{\omega}^i_{t-h+1}, \boldsymbol{\alpha}^i_{t-h+1}, \boldsymbol{\delta}^i_{t-h+2}, \boldsymbol{\alpha}^i_{t-h+2}, \ldots, \boldsymbol{\delta}^i_t) \tag{6}$$

In the expression above, $\boldsymbol{\delta}^i_j$ denotes the token subsequence corresponding to the delta or `diff` text observation Δo^i_j. A `diff` history neural LM agent can thus be represented as below.

$$\boldsymbol{\alpha}^i_t \sim \pi_\theta\left(\boldsymbol{\gamma}^i, \boldsymbol{\omega}^i_{t-h+1}, \boldsymbol{\alpha}^i_{t-h+1}, \boldsymbol{\delta}^i_{t-h+2}, \ldots, \boldsymbol{\delta}^i_t\right) \tag{7}$$

The loss function that we employ for training or finetuning θ on `diff` histories is defined in Appendix A.2, Equation 9. A diagrammatic visualization of `diff` history is provided in Figure 1 (right) as well as in Figure 2, where we show examples of the annotated summaries of differences between consecutive full-text observations output by `diff`.

3.4. Why would `diff` history help?

We conjecture two possible effects of `diff` history: (1) the addition of task-agnostic and relational abstraction to sequence modeling for decision-making; (2) token-level compression of long interaction histories in high dimensional environments.

Task-Agnostic and Relational Abstraction: The Unix `diff` operation computes pairwise text deltas between arbitrary strings. This makes it task-agnostic. Moreover, text deltas computed with `diff` are structured like code, providing a summary of text differences using commonplace symbols, notations, and abstractions via a simple "arithmetic-style" logic, as demonstrated in Figure 2, Appendix A.1, and Appendix E. Thus, finetuning LMs for sequential decision-making with `diff` observations does not require re-tokenization or the introduction of large sets of new tokens. This property makes it attractive for settings with little labeled data. Finally, as shown in Figure 2, the line-by-line matching performed by `diff` can localize salient changes to environments that occur between consecutive timesteps of interaction, tracking changes in objects' states and relations, high level features of the scene, and

more. As a result, diff history may provide a denser and richer learning signal during LM finetuning than the one provided by "raw" full-text interaction histories, improving downstream agent generalization after finetuning on limited data.

Token-Level Compression: Language model context length defines a maximal "computational budget" for token generation. Compressing relevant historical information as densely as possible on the token-level across context-length thus offers a direct means for extending and improving neural agent memory. This may result in improved modeling of expert action distributions during instruction finetuning, yielding LMs that produce higher quality generations at test-time. We explicitly probe this property in our experiments (see Figure 3 and Section 4.1.3).

4. Experiments

To understand the benefits of diff history, we experiment on two different procedurally-generated environments: BabyAI-Text (Chevalier-Boisvert et al., 2018; Carta et al., 2023) and NetHack (Küttler et al., 2020). Concretely, we seek to answer the following questions:

1. How does diff history impact the quality of generations after finetuning LMs on limited labeled data? How does it impact the robustness of neural language agents?
2. Does large-scale task-agnostic pretraining affect the performance of diff history neural language agents?
3. Does the downstream performance of diff history agents scale with data samples?
4. How do diff history agents compare against state-of-the-art neural agents?

We probe (1) in both BabyAI-Text and NetHack. In the remainder of our study, we concentrate our investigations on the high-dimensional environment NetHack, where low-resource neural agent training and finetuning have been demonstrated to be exceptionally difficult in prior work (Hambro et al., 2022a;b; Tuyls et al., 2023; Piterbarg et al., 2023). Summaries of our experimental results are provided in Figures 4, 5, and Table 1.

4.1. Generating Natural Language Interaction Data

4.1.1. BabyAI-Text

We leverage the procedural BabyAI bot (Chevalier-Boisvert et al., 2018) to collect interaction demonstrations in the multi-task BabyAI-Text environment. The configuration of this environment is as in Carta et al. (2023), so that data reflect five classes of tasks. A description of each of these task classes is provided in Appendix C.1.

We first run the bot across distinct random seeds of the environment to collect 5,000 full-length goal-conditioned solutions. Then, we sub-sample these across "chunks" $(H = 4)$ to yield two low-sample datasets, consisting of 1,000 and 5,000 goal-conditioned interaction demonstrations respectively. Further data statistics are reported in Appendix D.1.

4.1.2. NetHack

While the video game NetHack is vision-based by default, the recent NetHack Language Wrapper (Goodger et al., 2023) makes it possible to play the game fully through natural language. Unfortunately, as of the writing of this paper, existing datasets for NetHack are incompatible with this wrapper. These datasets store games in the impoverished tty* format, which makes it difficult to procedurally generate natural language descriptions of visual observations (Küttler et al., 2020; Hambro et al., 2022a; Piterbarg et al., 2023; Tuyls et al., 2023).

Thus, to support our experiments with diff history, we generate a new NetHack dataset consisting of full game states by running the leading symbolic bot AutoAscend in the environment (Hambro et al., 2022a). Note that all current state-of-the-art neural agents for NetHack are trained or pretrained on large datasets of AutoAscend games via supervised learning from pixels (Hambro et al., 2022b; Piterbarg et al., 2023; Tuyls et al., 2023). We call our dataset LangHack.

Our procedure for generating LangHack mirrors the procedure that we use in BabyAI-Text. We first run the AutoAscend bot across 10,000 NetHack seeds, recording full-length games. Then, we sub-sample these games uniformly at random into a dataset appropriate for LM finetuning consisting of 500,000 "chunks" of interaction demonstrations $(H = 64)$. Note that we scale up the length of

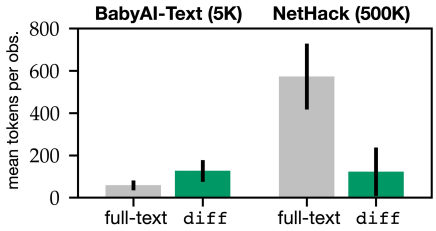

Figure 3. **Mean token counts per observation, before and after diff**: We compute token statistics of full-text and diff observations across natural language interaction datasets generated for a low-dimensional environment (BabyAI-Text) and a high-dimensional environment (NetHack) via the tokenizer we use for all LM experiments (Radford et al., 2019). Error bars indicate one standard deviation across datasets.

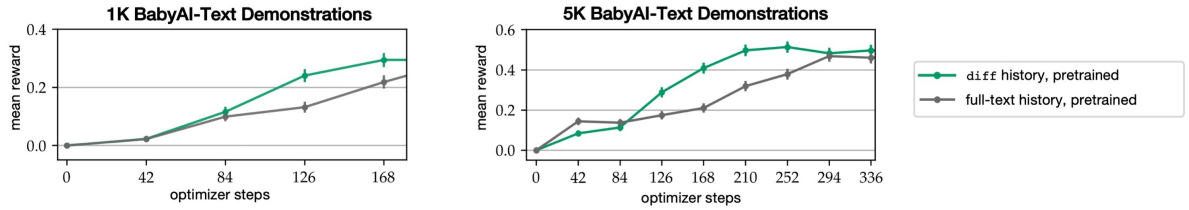

Figure 4. **Low-resource BabyAI-Text**: We compare the performance of language agents tuned with `diff` history against full-text history ablations in two low-sample finetuning settings, with 1,000 demonstrations (left) and 5,000 demonstrations (right) respectively. Though both agents eventually achieve equivalent performance over the course of finetuning in both experiments, the `diff` history language agent requires $\geq$**25% fewer FLOPs** to attain this peak. Error bars indicate one standard error in mean reward on a batch of withheld task instances.

NetHack interaction histories compared to the length of the interaction histories that we employ for BabyAI-Text data experiments ($H = 4$) due to the difference in difficulty between these two environments. A detailed report of `LangHack` statistics is provided in Appendix D.2

4.1.3. Properties of generated diff history data

We probe the effect of `diff` history on the token statistics associated with environment observations in Figure 3. In NetHack, we find that `diff` acts as a "soft compression" mechanism on natural language observations, yielding a 460% reduction in average token count per timestep. This finding supports our conjecture in Section 3.4. Nevertheless, we find that `diff` has *the opposite effect* in BabyAI-Text, where it produces a 210% average-case increase in per observation tokens. We attribute this property to the difference in state dimensionality between the two environments. Indeed, inspection of the generated data reveals that the natural language observations in BabyAI bot demonstrations change entirely between most consecutive timesteps of interaction. However, the relational property of `diff` described in Section 3.4 holds in both environments: `diff` observations consistently identify and localize per timestep changes in the descriptions of objects and scene properties. Examples of BabyAI-Text and NetHack `diff` interaction histories are provided in Appendix E.

4.2. diff vs. Full-Text History Language Agents

We begin our investigations by comparing the performance of neural language agents trained with `diff` history against full-text history ablations in BabyAI-Text and NetHack as a function of optimizer steps.

In all experiments, we tune the smallest parameter-count checkpoint of the decoder-only language model GPT-2 (Radford et al., 2019), employing identical hyperparameters and hardware across individual environments. Finetuning is conducted in mixed-precision with Microsoft DeepSpeed (Wang et al., 2023a) and HuggingFace Accelerate (Gugger et al., 2022) on NVIDIA A100 GPU nodes. We preserve the default 1024-token context length of the model in BabyAI-Text, but extend model context lengths by a factor of four up to 4096 tokens for finetuning on `LangHack` by introducing new positional encodings. All samples are padded or truncated to these context lengths, though we note that we find truncation to be unnecessary on our low-resource BabyAI-Text interaction datasets. A complete description of our training procedures is provided in Appendix F.

To evaluate agents, we run a batch of games on withheld, procedurally-generated instances of both environments. At each timestep of interaction, we provide recent interaction history in-context to LMs (as visualized in Figure 1) and use greedy decoding (Lowerre & Reddy, 1976) to generate natural language actions. We use no token sampling restrictions or other heuristics during conditional generation. Following previous work, we employ the total reward and/or game score achieved by agents across withheld task instances as a metric for language agent performance in our evaluations. More details on evaluation are in Appendix G.

We find the introduction of `diff` history to result in significant improvements in neural language agent performance in all of our instruction finetuning experiments. Learning curves reflecting the results of `diff` ablations on neural agent performance are displayed in Figure 4 and Figure 5, respectively. We employ the loss functions defined in Appendix A.2 and Appendix A.3 to tune models.

4.2.1. Action Prediction

In BabyAI-Text, we find that ablating `diff` observations when finetuning language agents for action prediction does not impact the eventual performance achieved by models. However, `diff` history agents reach this performance level with more than 25% fewer optimizer steps or FLOPs in both the ultra low-sample and low-sample labeled data settings that we test (Figure 4). We note that this improvement in efficiency occurs despite the average-case token count increase effect identified in Section 4.1.3 and Figure 3.

In NetHack, the effect of ablating `diff` observations from

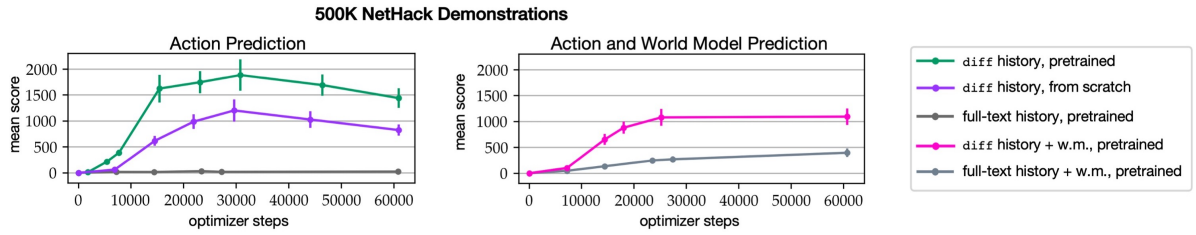

Figure 5. **Low-resource NetHack**: We compare the performance of language agents with `diff` history against ablations, testing efficiency and generalization when LMs are tuned on demonstrations of interaction history for natural language action prediction (left) and joint action-world model prediction (right). Error bars indicated one standard error in mean game score achieved by agents across a batch of withheld NetHack instances.

low-sample finetuning on action prediction is more dramatic, resulting in a 98% drop in the final max-performance achieved by neural language agents on new game seeds (see Figure 5 (left) and Table 1).

4.2.2. Action and World Model Prediction

In an effort to probe the performance gap between `diff` and full-text history agents tuned on `LangHack`, we experiment with removing all masking from observation tokens in the demonstration data used for finetuning. This results in language agents tuned not only on action prediction, but also on an *auxiliary natural language world modeling objective*. The conjecture behind this experiment is that in the information-sparse, full-text NetHack setting, introducing an auxiliary objective may improve the efficiency of representation learning. Our findings are displayed in Figure 5 (right).

Indeed, we find that the addition of the causal world model objective narrows the gap between LMs tuned with `diff` and full-text history, reducing the 98% difference in agent mean NetHack score on test games to a 69% difference (see Table 1). The auxiliary objective produces a large improvement in the quality of full-text history agent generations (10-fold improvement in mean NetHack score), but decreases the quality of `diff` history generations. Finetuning LMs with `diff` history data on action prediction alone yields stronger representations for decision-making in NetHack.

4.3. Large-Scale Pretraining

Next, we test the effect of ablating large-scale pretraining from `diff` history language agents in NetHack. Learning curves reflecting this ablation are also displayed in Figure 5 (left). We find that pretraining *does* have a significant positive effect on the performance achieved by `diff` history NetHack agents – re-initializing model weights to train on `LangHack` from scratch results in a 36% reduction in mean NetHack game score (see Table 1). Interestingly, we also find that the performance gap between pretrained and from-scratch agents remains fairly constant with optimizer steps over the course of training/tuning (see Appendix F).

4.4. Doubling NetHack Interaction Demonstrations

We confirm that finetuning LMs on labeled interaction demonstrations with `diff` history scales with data by running an experiment in which we double the number of samples in `LangHack`. We refer to this upscaled dataset as `LangHack-2x`. All other properties of `LangHack-2x` are precisely the same as those of `LangHack` described in Section 4.1.2, e.g. interaction demonstration "chunks" ($H = 64$) are sampled from the same set of 10,000 full `AutoAscend` games. The high-level statistics of `LangHack-2x` match those of `LangHack` (see Table 3).

The results from this experiment are summarized in Table 1 and in Figure 9. Agents trained with `diff` history for action prediction on `LangHack-2x` achieve 47% higher mean NetHack score than those trained on `LangHack`.

4.5. `diff` history Language Agents vs. NetHack SOTA

Finally, we compare the generalization of NetHack language agents against that of: (1) state-of-the-art vision agents trained for action prediction on `LangHack`; and (2) overall state-of-the-art neural agents for NetHack. Our results are summarized in Figure 1 (left), Table 1, and Figure G.2.

To support (1), we render demonstrator observations in `LangHack` in pixels via the procedures used in previous works (Hambro et al., 2022a). We train two types of vision agents for NetHack: the popular LSTM-based `CDGPT5` architecture developed during the NeurIPS 2021 NetHack Challenge Competition as well as a Transformer-based variant of `CDGPT5`, where the core LSTM module of the network is swapped for a Transformer as in Piterbarg et al. (2023) (see Appendix F.2). All of our language agents outperform state-of-the-art vision-based counterparts (Hambro et al., 2022b; Piterbarg et al., 2023; Tuyls et al., 2023), with the exception of the full-text history action prediction agent.

As for (2), we find that `diff` history agents tuned on `LangHack` outperform neural vision-language agents

Table 1. **Summary of experimental results on NetHack**: We compare the best in-game performance achieved by neural vision and language agents trained on `LangHack` and `LangHack-2x` against previous state-of-the-art neural agents for NetHack. These agents are trained on orders of magnitude more game data from the leading NetHack symbolic bot `AutoAscend` (Hambro et al., 2022a). We annotate agent scores statistics from Piterbarg et al. (2023) with (†) and those from Tuyls et al. (2023) with (‡). Details on evaluation procedures are provided in Appendix G. Models with architecture denoted as "LSTM (CDGPT)" and "XL-LSTM (CDGPT5)" reflect the vision-language model architecture for NetHack introduced as "Chaotic Dwarven GPT-5" (CDGPT5) in the NeurIPS 2021 NetHack Challenge Competition, which outperformed all other purely data-driven polices (Hambro et al., 2022a).

						NetHack Performance		
Agent Type	Model Architecture	# of Expert Actions	Data Format	World Model	Weight Init	Best History Horizon	Mean Score	Median Score
Vision	LSTM (CDGPT5)	32M	Pixel-Vector	-	Scratch	INF	192 ± 7	111
Vision	Trnsfrmr	32M	Pixel-Vector	-	Scratch	2	213 ± 6	114
Language	GPT-2-127M	32M	Full-Text	✗	Pretrain	8	31 ± 5	20
Language	GPT-2-127M	32M	Full-Text	✓	Pretrain	8	344 ± 52	171
Language	GPT-2-127M	32M	diff History	✗	Pretrain	15	$\mathbf{1885 \pm 213}$	1081
Language	GPT-2-127M	32M	diff History	✗	Scratch	15	1203 ± 214	645
Language	GPT-2-127M	32M	diff History	✓	Pretrain	15	1094 ± 160	453
Language	GPT-2-127M	64M	diff History	✗	Pretrain	15	$\mathbf{2766 \pm 481}$	1802
Vision	LSTM (CDGPT5)	3.2B	Pixel-Vector	-	Scratch	INF	$658^{(\dagger)}$	$403^{(\dagger)}$
Vision	Trnsfrmr-LSTM	3.2B	Pixel-Vector	-	Scratch	INF	$\mathbf{1318}^{(\dagger)}$	$917^{(\dagger)}$
Vision	XL-LSTM (CDGPT5)	115B	Pixel-Vector	-	Scratch	INF	$\mathbf{2740}^{(\ddagger)}$	-

trained on $100\times$ more data from the same demonstration source, the symbolic bot `AutoAscend`. Furthermore, our `diff` history language agent tuned for action prediction on `LangHack-2x` matches the performance of the overall current state-of-the-art neural agent for NetHack from Tuyls et al. (2023), which is trained on $1800\times$ as many labeled `AutoAscend` samples. The NetHack score distribution of our best agent has a long tail; indeed, the LM achieves a score exceeding 15000 (see Appendix H, Figure 10) and descends beyond dungeon level six in several test games.

5. Discussion, Limitations, and Conclusion

We propose a simple technique that uses the Unix `diff` operator to dramatically improve the quality of LM generations for sequential decision-making. `diff` history provides a procedural, deterministic, task-agnostic, and human-interpretable way to reformulate the interaction histories used to prompt models, removing redundancies between consecutive observation pairs in favor of a summary of salient differences in objects, scenes, and other abstract entities. It can also extend the modeling horizon of language agents at fixed context lengths. Through our experiments with `diff` history, we demonstrate by example that introducing abstraction into the prompts used to finetune language models into agents can make a powerful impact on downstream performance.

In BabyAI-Text, our experiments show that `diff` history yields a 25% improvement in the computational efficiency of low-data instruction finetuning over full-text interaction history ablations (Figure 4).

In the infinite-horizon video game NetHack, `diff` history makes it possible to tune tiny LMs (127 million parameters) into exceptionally strong agents using relatively little labeled data, but many tuning epochs (Appendix F). Under our paradigm, neural language agents achieve state-of-the-art performance for neural agents on NetHack with $1800\times$ fewer labels than prior work (Figure 1). The results of our experiments are reported in Figure 1, Figure 5, and Table 1. They reveal three key effects:

1. Language agents with `diff` history outperform vision-language baselines trained on equivalent data (89% higher mean NetHack score).

2. Ablating `diff` observations from the language model contexts used for finetuning and prompting destroys language agent performance (98% lower mean NetHack score).

3. The performance of `diff` history language agents scales with data. Tuning on tokens corresponding to

twice as many expert actions results in a 46% improvement in mean score, suggesting that the performance of NetHack diff history agents remains data-limited.

There are several limitations to our work, which if addressed, can bring us closer to general-purpose neural language agents.

First, we assume that observations and actions in decision-making can be expressed completely in textual form. Scaling to arbitrary problems will require work in procedurally converting observations to text or efficiently combining LMs with other modalities. Recent methods exploring this line of research in robot learning settings suggest that it holds promise (Peng et al., 2023; Brohan et al., 2023; Liu et al., 2024).

Second, the abstractions introduced by diff history exploit structure that is present apriori in textual observations. This makes diff history particularly compelling for textual tasks with inherent observational structure, such as web navigation, code editing, and the procedural game settings that we test here. However, we do not expect diff history to provide improvement to LM agents in settings with entirely unstructured and "narrative-like" text observations. Nevertheless, we conjecture that it may be possible to make such environments more amenable to diff history through the addition of a procedural pre-processing step; for example, by using a parser to convert observations into an object- or agent-centric "stateful" form.

Finally, our best language agents do not utilize explicit planning, a la System 2 inference (Kahneman, 2011). Combining the ability of diff history to understand historical observations with explicit modeling of future ones will be an exciting direction for future work.

Acknowledgements

This work is supported by ONR award numbers N00014-21-1-2404 and N00014-21-1-2758. UP is supported by the NSF GRFP Fellowship. LP is supported by the Packard Fellowship. We are especially grateful to Shenglong Wang and NYU High Performance Computing for their support.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., Jesmonth, S., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Sievers, N., Tan, C., Toshev, A., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M., and Zeng, A. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

Allen, K. R., Smith, K. A., and Tenenbaum, J. B. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 117(47):29302–29310, 2020.

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Gianinazzi, L., Gajda, J., Lehmann, T., Podstawski, M., Niewiadomski, H., Nyczyk, P., et al. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*, 2023.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., Florence, P., Fu, C., Arenas, M. G., Gopalakrishnan, K., Han, K., Hausman, K., Herzog, A., Hsu, J., Ichter, B., Irpan, A., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Leal, I., Lee, L., Lee, T.-W. E., Levine, S., Lu, Y., Michalewski, H., Mordatch, I., Pertsch, K., Rao, K., Reymann, K., Ryoo, M., Salazar, G., Sanketi, P., Sermanet, P., Singh, J., Singh, A., Soricut, R., Tran, H., Vanhoucke, V., Vuong, Q., Wahid, A., Welker, S., Wohlhart, P., Wu, J., Xia, F., Xiao, T., Xu, P., Xu, S., Yu, T., and Zitkovich, B. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *arXiv preprint arXiv:2307.15818*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Carta, T., Romac, C., Wolf, T., Lamprier, S., Sigaud, O., and Oudeyer, P.-Y. Grounding large language models in interactive environments with online reinforcement learning, 2023.

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., and Bengio, Y. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling

with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. Scaling instruction-finetuned language models, 2022.

Craddock, D. L. *Dungeon Hacks: How NetHack, Angband, and Other Rougelikes Changed the Course of Video Games*. CRC Press, 2021.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Gemini Team, G. Gemini: A family of highly capable multimodal models, 2023.

Goodger, N., Vamplew, P., Foale, C., and Dazeley, R. A nethack learning environment language wrapper for autonomous agents. *Journal of Open Research Software*, 11:8, 2023.

Gopnik, A., Glymour, C., Sobel, D. M., Schulz, L. E., Kushnir, T., and Danks, D. A theory of causal learning in children: causal maps and bayes nets. *Psychological review*, 111(1):3, 2004.

Gugger, S., Debut, L., Wolf, T., Schmid, P., Mueller, Z., Mangrulkar, S., Sun, M., and Bossan, B. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate, 2022.

Hambro, E., Mohanty, S., Babaev, D., Byeon, M., Chakraborty, D., Grefenstette, E., Jiang, M., Daejin, J., Kanervisto, A., Kim, J., et al. Insights from the neurips 2021 nethack challenge. In *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 41–52. PMLR, 2022a.

Hambro, E., Raileanu, R., Rothermel, D., Mella, V., Rocktäschel, T., Küttler, H., and Murray, N. Dungeons and data: A large-scale nethack dataset. *Advances in Neural Information Processing Systems*, 35:24864–24878, 2022b.

Jiang, Y., Gupta, A., Zhang, Z., Wang, G., Dou, Y., Chen, Y., Fei-Fei, L., Anandkumar, A., Zhu, Y., and Fan, L. Vima: General robot manipulation with multimodal prompts. In *Fortieth International Conference on Machine Learning*, 2023.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Kahneman, D. *Thinking, fast and slow*. Macmillan, 2011.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kernighan, B. W. and Pike, R. *The UNIX programming environment*, volume 270. Prentice-Hall Englewood Cliffs, NJ, 1984.

Kirk, R., Zhang, A., Grefenstette, E., and Rocktäschel, T. A survey of generalisation in deep reinforcement learning. *arXiv e-prints*, pp. arXiv–2111, 2021.

Klissarov, M., D'Oro, P., Sodhani, S., Raileanu, R., Bacon, P.-L., Vincent, P., Zhang, A., and Henaff, M. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166*, 2023.

Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning, 2021.

Li, B. Z., Nye, M., and Andreas, J. Language modeling with latent situations, 2022.

Liu, P., Orru, Y., Paxton, C., Shafiullah, N. M. M., and Pinto, L. Ok-robot: What really matters in integrating open-knowledge models for robotics. *arXiv preprint arXiv:2401.12202*, 2024.

Lowerre, B. and Reddy, R. The harpy speech recognition system: performance with large vocabularies. *The Journal of the Acoustical Society of America*, 60(S1):S10–S11, 1976.

Mishra, S., Khashabi, D., Baral, C., and Hajishirzi, H. Cross-task generalization via natural language crowdsourcing instructions, 2022.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

OpenAI. Gpt-4 technical report, 2023.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch, 2017.

Peng, A., Sucholutsky, I., Li, B., Sumers, T., Griffiths, T., Andreas, J., and Shah, J. Learning with language-guided state abstractions. In *RSS Workshop on Social Intelligence in Humans and Robots*, 2023.

Piterbarg, U., Pinto, L., and Fergus, R. Nethack is hard to hack. *arXiv preprint arXiv:2305.19240*, 2023.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., Rutherford, E., Hennigan, T., Menick, J., Cassirer, A., Powell, R., van den Driessche, G., Hendricks, L. A., Rauh, M., Huang, P.-S., Glaese, A., Welbl, J., Dathathri, S., Huang, S., Uesato, J., Mellor, J., Higgins, I., Creswell, A., McAleese, N., Wu, A., Elsen, E., Jayakumar, S., Buchatskaya, E., Budden, D., Sutherland, E., Simonyan, K., Paganini, M., Sifre, L., Martens, L., Li, X. L., Kuncoro, A., Nematzadeh, A., Gribovskaya, E., Donato, D., Lazaridou, A., Mensch, A., Lespiau, J.-B., Tsimpoukelli, M., Grigorev, N., Fritz, D., Sottiaux, T., Pajarskas, M., Pohlen, T., Gong, Z., Toyama, D., de Masson d'Autume, C., Li, Y., Terzi, T., Mikulik, V., Babuschkin, I., Clark, A., de Las Casas, D., Guy, A., Jones, C., Bradbury, J., Johnson, M., Hechtman, B., Weidinger, L., Gabriel, I., Isaac, W., Lockhart, E., Osindero, S., Rimell, L., Dyer, C., Vinyals, O., Ayoub, K., Stanway, J., Bennett, L., Hassabis, D., Kavukcuoglu, K., and Irving, G. Scaling language models: Methods, analysis insights from training gopher, 2022.

Raymond, E. S. A guide to the mazes of menace: Guidebook for nethack. *NetHack. org*, 2003.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. {ZeRO-Offload}: Democratizing {Billion-Scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564, 2021.

Sipser, M. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

Steinkraus, K. and Kaelbling, L. P. Combining dynamic abstractions in very large mdps, 2004.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., and Goodman, N. D. How to grow a mind: Statistics, structure, and abstraction. *science*, 331(6022):1279–1285, 2011.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Tuyls, J., Madeka, D., Torkkola, K., Foster, D., Narasimhan, K., and Kakade, S. Scaling laws for imitation learning in nethack. *arXiv preprint arXiv:2307.09423*, 2023.

Wang, G., Qin, H., Jacobs, S. A., Holmes, C., Rajbhandari, S., Ruwase, O., Yan, F., Yang, L., and He, Y. Zero++: Extremely efficient collective communication for giant model training. *arXiv preprint arXiv:2306.10209*, 2023a.

Wang, Y., Ivison, H., Dasigi, P., Hessel, J., Khot, T., Chandu, K. R., Wadden, D., MacMillan, K., Smith, N. A., Beltagy, I., et al. How far can camels go? exploring the state of instruction tuning on open resources. *arXiv preprint arXiv:2306.04751*, 2023b.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Wu, Y., Min, S. Y., Prabhumoye, S., Bisk, Y., Salakhutdinov, R., Azaria, A., Mitchell, T., and Li, Y. Spring: Gpt-4 outperforms rl algorithms by studying papers and reasoning. *arXiv preprint arXiv:2305.15486*, 2023.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Zhong, V., Mu, J., Zettlemoyer, L., Grefenstette, E., and Rocktäschel, T. Improving policy learning via language dynamics distillation. *Advances in Neural Information Processing Systems*, 35:12504–12515, 2022.

Zhou, C., He, J., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. Prompt consistency for zero-shot task generalization, 2022.

A. More on diff History

A.1. A Simple Example

Let (file_i.txt, file_j.txt) be a pair of text files. The Unix operation diff computes the *delta* between these files. For example, suppose that file_i.txt contains the text shown below as its contents.

```
Orange
Banana
Mango
```

Now, suppose that file_j.txt has the contents below.

```
Orange
Apple
Mango
```

Then, executing the command "diff file_i.txt file_j.txt" in Unix will return an output that looks similar to the following.

```
--- file_i
+++ file_j
@@  -1,2  +1,2 @@
-Banana
+Apple
```

As demonstrated in this example, the text deltas computed by the Unix diff operation omit information[1] about lines that remain unchanged between files, i.e. the lines in file_i with an exact match in file_j, summarizing additions and deletions only. The diff operation is not commutative.

A.2. Action Prediction Losses

As in Section 3.2, let $\{g, (o_1, a_1, \dots, o_H, a_H)\}$ be a full-length interaction history sample reflecting a task instruction g from a demonstration dataset $\mathcal{D}$ for sequential decision-making. Following the notation introduced in the main paper, we write the tokenized version of the instruction as the sequence $\boldsymbol{\gamma}$. Similarly, we write the tokenized *full-text* version of the interaction history in terms of token subsequences as

$$(\boldsymbol{\omega}_1, \boldsymbol{\alpha}_1, \dots, \boldsymbol{\omega}_H, \boldsymbol{\alpha}_H),$$

and the diff *history* version as,

$$(\boldsymbol{\omega}_1, \boldsymbol{\alpha}_1, \boldsymbol{\delta}_2, \boldsymbol{\alpha}_2, \dots, \boldsymbol{\delta}_H, \boldsymbol{\alpha}_H).$$

In the expressions above, the symbols $\boldsymbol{\omega}_i$ denote token subsequences corresponding to full-text observations, $\boldsymbol{\alpha}_i$ denote token subsequences corresponding to actions, and $\boldsymbol{\delta}_i$ denote token subsequences corresponding to diff observations.

Next, we borrow some notation from computability theory (Sipser, 1996). Let $\boldsymbol{\Sigma}$ denote the set of all defined tokens, or the *language* of our modeling problem. In the settings that we consider, the tokens used to represent observation and action data for sequential decision-making are intentionally chosen to belong to the same language, i.e. all $\boldsymbol{\omega}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\delta}$ are sequences of elements in $\boldsymbol{\Sigma}$, and individual tokens may very well be reused between these different parts of interaction history.

We can further write out token subsequences in terms of their constituent tokens as $\boldsymbol{\omega}_i = (\omega_i^1, \dots)$, $\boldsymbol{\alpha}_i = (\alpha_i^1, \dots)$, $\boldsymbol{\delta}_i = (\delta_i^1, \dots)$, where all $\omega_i^j, \alpha_i^j, \delta_i^j \in \boldsymbol{\Sigma}$.

[1] In some implementations of diff, a few lines that remain unchanged between files but that are *adjacent* to additions or deletions may also returned by diff as "context." However, in many implementations of Unix-style diff such as the one provided by the Python library difflib, such "context" can be turned off.

The loss functions that we minimize when tuning LMs with parameters θ on **causal action prediction** from interaction history sequences can thus be written as,

$$\mathcal{L}_{\text{action-full-text}} = -\sum_i \sum_j \log p_\theta \left(\alpha_i^j \mid \boldsymbol{\gamma}, (\boldsymbol{\omega}_1, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\omega}_i), (\alpha_i^1, \ldots \alpha_i^{j-1}) \right) \tag{8}$$

$$\mathcal{L}_{\text{action-}\texttt{diff}} = -\sum_i \sum_j \log p_\theta \left(\alpha_i^j \mid \boldsymbol{\gamma}, (\boldsymbol{\omega}_1, \boldsymbol{\alpha}_1, \boldsymbol{\delta}_2, \ldots, \boldsymbol{\omega}_i), (\alpha_i^1, \ldots \alpha_i^{j-1}) \right). \tag{9}$$

In other words, we look to maximize the likelihood of individual expert action tokens α_i^j conditioned on: (1) the task instruction; (2) all interaction history up to the timestep j; and (3) the $(j-1)$-length prefix of $\boldsymbol{\alpha}_i$. Note that these objectives reflect the use of teacher-forcing (Williams & Zipser, 1989).

A.3. World Model Prediction Losses

The definitions of the **auxiliary causal world model prediction** losses for full-text and `diff` history neural language agents (as explored in Section 4.2.2 of our experiments) mirror the losses above.

$$\mathcal{L}_{\text{world-full-text}} = -\sum_{i\,:\,i>1} \sum_j \log p_\theta \left(\omega_i^j \mid \boldsymbol{\gamma}, (\boldsymbol{\omega}_1, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\omega}_{i-1}), (\omega_i^1, \ldots \omega_i^{j-1}) \right) \tag{10}$$

$$\mathcal{L}_{\text{world-}\texttt{diff}} = -\sum_{i\,:\,i>1} \sum_j \log p_\theta \left(\delta_i^j \mid \boldsymbol{\gamma}, (\boldsymbol{\omega}_1, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\delta}_{i-1}, \boldsymbol{\alpha}_{i-1}), (\delta_i^1, \ldots \delta_i^{j-1}) \right). \tag{11}$$

Minimizing these objectives is equivalent to maximizing the likelihood of observation tokens in generated full-text and `diff` history demonstration data conditioned on: (1) the task instruction; (2) all previous interaction history; and (3) the $(j-1)$-length prefix of $\boldsymbol{\omega}_i$ or $\boldsymbol{\delta}_i$. As above, our world model prediction losses use teacher-forcing.

B. Formatting Interaction Histories

B.1. Beginning and End-of-Subsequence Tokens

As noted in Section 3.1, the lengths of the token subsequences corresponding to individual observations and actions in interleaved interaction histories might vary from timestep to timestep. Thus, we need a mechanism to distinguish between observation and action token subsequences. We draw inspiration from the formatting methods employed by instruction tuning approaches to tune LMs for chat (Wang et al., 2023b) to resolve this problem, introducing two special tokens to LM tokenizers: `<|action|>` and `<|observation|>`.

These special tokens are added to the start and end of each action token subsequence, respectively. This choice allows us to condition LM policies on interaction histories with variable-length horizons, maneuvering around the constraint of fixed LM context-lengths to "look back" as many timesteps t into the past as we can fit in-context up to a horizon h, where $t : 1 \leq t \leq h \leq H$ and H represents the maximal horizon of interaction histories represented in the demonstration dataset $\mathcal{D}$ (see Section 3.3).

We add these tokens to tokenizers at the start of training or tuning, resizing pre-configured language model token embedding dimensions accordingly.

B.2. Considerations During Generation

During inference, we can also leverage the special tokens `<|action|>` and `<|observation|>` as "beginning"- and "end-of-subsequence" markers to simplify the process of textual action decoding and generation. More precisely, LM agents trained on interaction history sequences with our method can be prompted to start generating an action prediction token-by-token by appending the special token `<|action|>` to the end of a text prompt. Similarly, generation of the special token `<|observation|>` can function as a termination condition for open-ended text action generation. Sampling actions from LM policies trained under our paradigm with these special tokens is thus easy to automate.

C. Environments

C.1. BabyAI-Text

As introduced in the main body of this paper, BabyAI-Text is a simple, procedurally generated grid-world environment with five constituent classes of task, introduced and open-sourced by Carta et al. (2023). It is a fully text-based environment, reflecting the addition of a procedural, text wrapper over the well-known BabyAI environment developed by Chevalier-Boisvert et al. (2018).

We provide a description of each of the task classes in BabyAI-Text below.

1. **Go to `<object>`**: The agent must navigate to the item `<object>`.
2. **Pick up `<object>`**: The agent must navigate to `<object>` and apply the "pick up" action.
3. **Put `<object A>` next to `<object B>`**: The agent must navigate to `<object A>` and pick it up. Then, it must navigate to `<object B>` and put down `<object A>`.
4. **Open door**: The agent must navigate to the key with the same color as the target door and pick this key up. Then, it must navigate to the target door and applying the "toggle" unlocking action.
5. **Pick up `<object A>` and then go to `<object B>`**: The agent must navigate to `<object A>` and apply the "pick up" action before navigating to `<object B>`.

A pixel-rendered visual observation of a task instance belonging to each of these classes is displayed in Figure 6. Across all tasks, there are five natural language actions available to the agent at each timestep: "turn left," "turn right," "go forward," "pick up," "drop," "toggle." We preserve the default tokenization of each of these actions in our neural LM agent experiments.

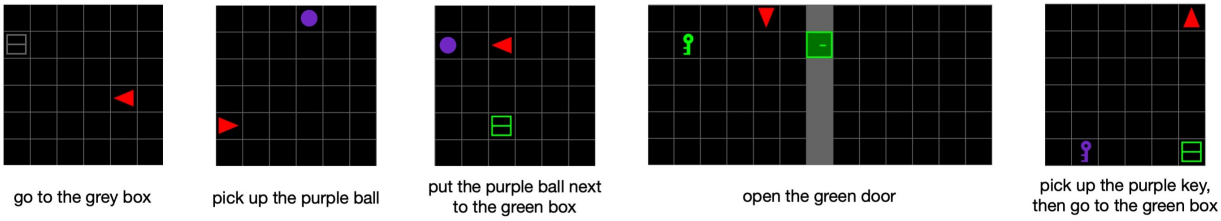

Figure 6. **BabyAI-Text**: Examples of visual observations for each of the five types of tasks in the BabyAI-Text environment, rendered in pixels. The agent is represented by the red triangle all images. The text-based description of the agent's task (as provided to neural LM agents in the "instruction" prefix of all tuning data and inference-time prompts) is provided below each image.

The reward assigned to a rollout across t steps of interaction in BabyAI-Text is computed as

$$r(t) = \mathbb{1}(\text{task successfully completed at timestep } t) \cdot \left(1 - \frac{0.9 \cdot t}{t_{\max}}\right), \tag{12}$$

where $t_{\max} \leq 200$ is a task-specific parameter and $\mathbb{1}(\cdot)$ represents an indicator function. If an agent fails to complete a task in time $t = t_{\max}$, the rollout is assigned a reward of zero. We employ mean rollout reward over a batch of withheld, procedurally generated task instances as central our metric of neural LM agent generalization within this environment.

Examples of formatted full-text and `diff` history interaction demonstrations collected in BabyAI-Text from the BabyAI bot (Chevalier-Boisvert et al., 2018) via the procedures detailed in Section 4.1 are provided in Appendix E.1.

C.2. NetHack

NetHack is an exceptionally challenging and procedurally generated roguelike video game that was released in 1987. The game is rendered with ASCII-characters. Each instance or *random seed* of NetHack features randomly re-generated player roles and attributes as well as seeded "dungeon levels," special items, monsters, and much more. NetHack was also one

of the very first games on the Web, uploaded by its creators to USENET newsgroups. As a result, human player data for NetHack is *plentiful* (Craddock, 2021). The symbolic NetHack bot `AutoAscend` continues to outperform all other decision-making agents for NetHack. The game is very far from solved (Hambro et al., 2022a;b; Piterbarg et al., 2023; Klissarov et al., 2023; Zhong et al., 2022; Tuyls et al., 2023).

C.2.1. Why is NetHack so hard?

To win the game, a player must navigate through dozens of interlinked dungeon levels to the very "bottom" of the game world in order to retrieve a special item: the Amulet of Yendor. The player must then navigate all the way back up to the starting level, from which they must "escape" to conclude the game. This goal is dubbed as *ascension* (Raymond, 2003). It is exceedingly difficult for human expert players to achieve. Indeed, as of 01/30/2024, public play statistics on the NetHack altorg server[2] indicate that only only $\approx 0.96\%$ of all games in January 2024 have concluded in ascension.

Simply surviving in NetHack is also difficult. Throughout the game, monsters will randomly appear on the player's dungeon level. These monsters must be escaped from or vanquished for the game to go on. As the player descends deeper into the dungeon, monsters become increasingly harder to defeat, requiring mastery of increasingly complex weapons, fighting skills, and more. But monsters are not the only threats to a player's existence: the game may also end through "death-by-starvation." To avoid this fate, players must continually search dungeon levels for food that is safe to eat (Raymond, 2003).

Formally, NetHack a high-dimensional, infinite-horizon, stochastic, and partially-observable Markov decision process (POMDP) (Kaelbling et al., 1998; Steinkraus & Kaelbling, 2004). A "fog-of-war" obscures unvisited areas of dungeon levels, and the exact consequences of player actions may be randomized by the game engine. Furthermore, unlike many other games commonly used to evaluate neural agents such as MineCraft, the "rule-set" of NetHack is exceptionally difficult to accurately write down, let alone summarize and fit into a neural LM context length (Raymond, 2003; Wu et al., 2023).

C.2.2. NetHack Learning Environment

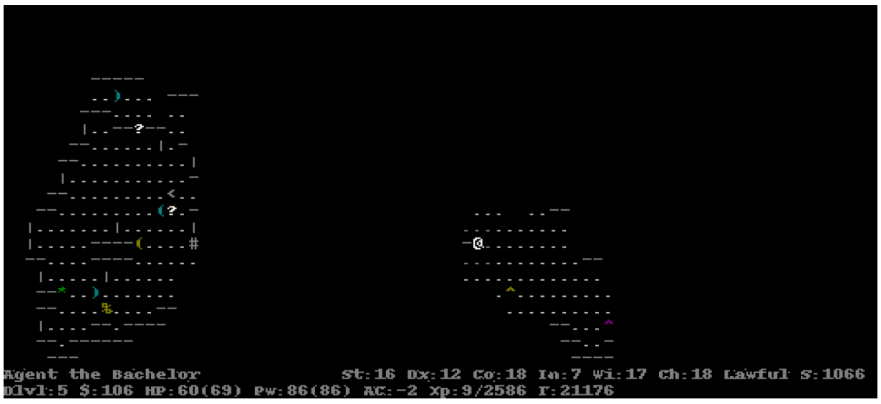

Figure 7. **NetHack Learning Environment**: An example of a visual observation in the NetHack Learning Environment, rendered in pixels. The agent is represented by the white "`@`" character in the middle of the image. The current state of the game is summarized at the bottom of the image, via "bottom-line statistics." The agent is located on level 5 of the dungeon, as indicated by the text reading "`Dlvl 5`" in the bottom, left-hand corner.

The NetHack Learning Environment (NLE) from Küttler et al. (2020) wraps the NetHack video game "as is" into an environment for sequential decision-making research. NetHack *game score* provides a measure of meaningful progress in the game, and is exposed by NLE to agents. This is the metric we use to evaluate neural agent performance and in-game generalization on NetHack in this paper.

The full, default observation space of the environment has ten components, enumerated below.

1. **`glyphs`**: A symbolic representation of all entities in-view to the player on the current dungeon-level.

[2] `https://alt.org/nethack/`

2. **message**: A text string representing the message from the game to the player (if any) at the current timestep of interaction.
3. **blstats**: The "bottom-line statistics" of the game at the current timestep, which include an identifier for the current dungeon level, the total in-game score received, the player's hunger level, etc.
4. **tty_chars**: The ASCII-character "rendered" version of the player's current observation.
5. **tty_colors**: The color codes corresponding to each ASCII character in "`tty_chars`."
6. **tty_cursor**: The current location of the player's cursor over the ASCII-character view of the game.
7. **inv_glyphs**: A symbolic representation of all items currently in the player's inventory.
8. **inv_strs**: A byte array describing current inventory items.
9. **inv_letters**: The text character codes corresponding to each inventory item.
10. **inv_oclasses**: A symbolic representation of the *class* of each inventory item.

Most existing neural agents for NetHack directly leverage this observation space during learning. Indeed, prior to the writing of this paper, all state-of-the-art neural architectures for NetHack render the three `tty*` components[3] of the observation space in pixels (see Figure 7). There is a data writer native to NetHack that writes the `tty*` components of observations across timesteps of game-play to a custom `.ttyrec` format (Küttler et al., 2020). Existing large scale datasets for NetHack store data in this format (Hambro et al., 2022b; Piterbarg et al., 2023).

There are several task variants of NLE. All of our experiments and investigations with NetHack focus on the hardest of these variants: the NetHack Challenge task, which exposes the full space of 121 possible per-timestep game actions to agents. This task was introduced during the NeurIPS 2021 NetHack Challenge Competition, which pitted symbolic decision-making agents against learned ones (Hambro et al., 2022a)

As of 2023, there exists a natural language wrapper for NLE (Goodger et al., 2023), which translates the non-natural language components of NLE observations into equivalent descriptions across all tasks. This language wrapper also translates NLE actions into natural language equivalents and vice-versa. In the case of the "NetHack Challenge" NLE task that we study here, this corresponds to a per-timestep action space of 121 possible *action strings*. A list of all possible commands (some of which correspond to *sequences* of per-timestep actions) is given in the public NetHack wiki[4]. We provide several annotated examples of NetHack actions below to give readers a feel for the game.

- **northeast**: Move diagonally one tile in the northeast direction.
- **eat**: Eat the currently selected item (if any).
- **pickup**: Pickup the item under you (if any).
- **search**: Search the tiles immediately around you. This action may reveal passages or doors on the current dungeon level previously obscured from the player.
- **quaff**: Drink from the item under you or currently selected (if any).
- **loot**: Loot a box on the floor.
- **enhance**: Open an auxilary menu to check your current weapon skills.

Unfortunately, as stated in the main body of the paper, the NetHack language wrapper requires access to full observations, i.e. symbolic components of the NLE observation space (Goodger et al., 2023). We generate a new dataset with these observation components to support the study of text-based decision-making on NetHack in this work.

[3]Note that, together, the three `tty*` components of the NLE observation space are equivalent to a human-player view of NetHack.
[4]`https://nethackwiki.com/wiki/Commands`

D. Dataset Statistics

We provide data statistics reflecting all of the low-sample interaction demonstration datasets that we generate in this paper. A detailed description of the data generation procedures is provided in Section 4.1 of the main body. All token statistics are computed with the GPT-2 tokenizer, which is used for all LM experiments (Radford et al., 2019). Reported errors reflect one standard deviation across the full datasets.

D.1. BabyAI-Text

Table 2. **Processing BabyAI-Text 1K and 5K interaction history demonstration datasets**: Aggregate statistics reflecting the `diff` history and full-text interaction history data formats that are used to tune LMs in our low-sample BabyAI-Text experiments.

	BabyAI-Text-1K		BabyAI-Text-5K	
	diff	Full-Text	diff	Full-Text
History Horizon per Demo	4	4	4	4
Min Tokens Per Demo	93	93	75	82
Max Tokens Per Demo	667	392	667	392
Median Tokens Per Demo	358	232	354	230
Mean Tokens Per Demo	359 ± 103	231 ± 53	$\mathbf{358 \pm 99}$	$\mathbf{230 \pm 50}$
Min Tokens Per Obs	21	12	21	12
Max Tokens Per Obs	266	122	**266**	**123**
Median Tokens Per Obs	130	57	128	57
Mean Tokens Per Obs	126 ± 51	59 ± 23	$\mathbf{126 \pm 49}$	$\mathbf{59 \pm 23}$
Min Tokens Per Action	2	2	2	2
Max Tokens Per Action	3	3	3	3
Total Tokens	**359K**	**231K**	**1.78M**	**1.15M**

D.2. NetHack

Table 3. **Processing `LangHack` and `LangHack-2x` interaction history demonstration data**: Aggregate statistics reflecting all interaction history data formats that are used to train and tune models in our NetHack experiments.

	LangHack (500K)			LangHack-2x (1M)
	diff	Full-Text	Pixel-Vector	diff
History Horizon Per Demo	64	64	64	64
Min Tokens Per Demo	544	13,785	-	544
Max Tokens Per Demo	54,887	105,301	-	73,469
Median Tokens Per Demo	7,915	35,508	-	7,911
Mean Tokens Per Demo	$\mathbf{8{,}465 \pm 2{,}769}$	$\mathbf{36{,}819 \pm 9{,}557}$	-	$\mathbf{8{,}464 \pm 2{,}764}$
Min Tokens Per Obs	1	196	-	1
Max Tokens Per Obs	**2269**	**1833**	-	**2269**
Median Tokens Per Obs	93	550	-	93
Mean Tokens Per Obs	$\mathbf{123 \pm 115}$	$\mathbf{573 \pm 156}$	-	$\mathbf{123 \pm 115}$
Min Tokens Per Action	2	2	-	2
Max Tokens Per Action	5	5	-	5
Total Tokens	**4.23B**	**18.4B**	-	**8.46B**

E. Examples of Formatted Interaction Histories

We provide a side-by-side comparison of example formatted full-text and diff interaction histories in our two environments. These examples are not cherry-picked. For NetHack, we show only two consecutive timesteps of interaction history here in the interest of brevity.

E.1. BabyAI-Text

E.1.1. Instruction

Your task is to go to the yellow key.
You can take 6 different actions: turn left, turn right, go forward, pick up, drop, and toggle.
a wall 4 steps forward
a wall 3 steps left

E.1.2. Full-Text History Completion

<|action|>**turn right**
<|observation|>
a wall 4 steps forward
a wall 3 steps right
<|action|>**turn right**
<|observation|>
a wall 3 steps forward
a wall 3 steps right
a yellow key 2 steps right and 2 steps forward
<|action|>**go forward**
<|observation|>
a wall 2 steps forward
a wall 3 steps right
a yellow key 2 steps right and 1 step forward
<|action|>**go forward**
<|observation|>
a wall 1 step forward
a yellow key 2 steps right
<|action|>**turn right**
<|observation|>
a wall 1 step left
a yellow key 2 steps forward
<|action|>**go forward**

E.1.3. Diff History Completion

<|action|>**turn right**
<|observation|>
@@ −2 +2 @@
−a wall 3 steps left
+a wall 3 steps right
<|action|>**turn right**
<|observation|>
@@ −1 +1 @@
−a wall 4 steps forward
+a wall 3 steps forward
@@ −2,0 +3 @@
+a yellow key 2 steps right and 2 steps forward
<|action|>**go forward**
<|observation|>
@@ −1 +1 @@
−a wall 3 steps forward
+a wall 2 steps forward
@@ −3 +3 @@
−a yellow key 2 steps right and 2 steps forward
+a yellow key 2 steps right and 1 step forward
<|action|>**go forward**
<|observation|>
@@ −1,2 +1,1 @@
−a wall 2 steps forward
−a wall 3 steps right
+a wall 1 step forward
@@ −1,3 +1,2 @@
−a yellow key 2 steps right and 1 step forward
+a yellow key 2 steps right
<|action|>**turn right**
<|observation|>
@@ −1 +1 @@
−a wall 1 step forward
+a wall 1 step left
@@ −1 +1 @@
−a yellow key 2 steps right
+a yellow key 2 steps forward
<|action|>**go forward**

E.2. NetHack

E.2.1. INSTRUCTION

You are an agent playing NetHack. Predict the next keypresses.

statistics[
Strength: 19/19
Dexterity: 14
Constitution: 18
Intelligence: 9
Wisdom: 7
Charisma: 8
Depth: 1
Gold: 25
HP: 57/61
Energy: 25/25
AC: 0
XP: 7/973
Time: 18818
Position: 43|13
Hunger: Not Hungry
Monster Level: 0
Encumbrance: Unencumbered
Dungeon Number: 0
Level Number: 1
Score: 3917
Alignment: Neutral
Condition: None
]
glyphs[
grave very far westsouthwest
boulder very far westnorthwest
orcish spear very far westnorthwest
stairs up very far westnorthwest
rocks very far westnorthwest
stairs down very far westnorthwest
iron skull caps very far westnorthwest
boulder far north, northnortheast, and northnorthwest
rock far northnortheast
rocks far north, westnorthwest, and northnorthwest
vertical wall far west
iron skull cap far westnorthwest and northnorthwest
hooded cloaks far northnorthwest
horizontal wall near north, south, southwest, and northwest
rock near northnortheast
boulder near westnorthwest
leather armor near northnorthwest
dark area very near southeast
tame large dog very near south
vertical wall adjacent northeast and east
rock adjacent southeast
hill orc corpse adjacent northwest
]
...

E.2.2. Full-Text History Completion

<|action|>**northeast**
<|observation|>
statistics
Strength: 16/16
Dexterity: 12
Constitution: 18
Intelligence: 7
Wisdom: 17
Charisma: 18
Depth: 5
Gold: 106
HP: 55/69
Energy: 86/86
AC: -2
XP: 9/2586
Time: 21156
Position: 5|11
Hunger: Not Hungry
Monster Level: 0
Encumbrance: Unencumbered
Dungeon Number: 2
Level Number: 2
Score: 10660
Alignment: Lawful
Condition: None
]
glyphs
arrow far northnortheast
scroll labeled DAIYEN FOOELS far eastnortheast
stairs up far eastnortheast
key far east
scroll labeled KO BATE far east
flute far eastsoutheast
ape corpse far southsoutheast
horizontal wall far south
horizontal wall near northeast
southwest corner near southeast and southwest
arrow near southsoutheast
worthless piece of green glass near south
southeast corner adjacent north and west
northwest room corner adjacent northwest
]
...
<|action|>**search**
<|observation|>
statistics
Strength: 16/16
Dexterity: 12
Constitution: 18
Intelligence: 7
Wisdom: 17
Charisma: 18
Depth: 5
Gold: 106
HP: 55/69
Energy: 86/86
AC: -2
XP: 9/2586
Time: 21157
Position: 5|11
Hunger: Not Hungry
Monster Level: 0
Encumbrance: Unencumbered
Dungeon Number: 2
Level Number: 2
Score: 10660
Alignment: Lawful
Condition: None
]
glyphs
arrow far northnortheast
scroll labeled DAIYEN FOOELS far eastnortheast
stairs up far eastnortheast
key far east
scroll labeled KO BATE far east
flute far eastsoutheast
ape corpse far southsoutheast
horizontal wall far south
horizontal wall near northeast
southwest corner near southeast and southwest
arrow near southsoutheast
worthless piece of green glass near south
southeast corner adjacent north and west
northwest room corner adjacent northwest
]
...
<|action|>**northeast**
...

E.2.3. Diff History Completion

<|action|>**northeast**
<|observation|>
<|@@|> −15 +15 <|@@|>
−Position: 4|12
+Position: 5|11
<|@@|> −27 +26,0 <|@@|>
−horizontal wall far northeast
<|@@|> −30,3 +29,2 <|@@|>
−key far eastnortheast
−scroll labeled KO BATE far eastnortheast
−vertical wall far east
+key far east
+scroll labeled KO BATE far east
<|@@|> −34,9 +32,8 <|@@|>
−southeast corner far southeast
−arrow near southeast
−ape corpse near southeast
−worthless piece of green glass near southsoutheast
−northeast room corner near south
−southwest corner very near southwest
−vertical wall very near west
−southeast corner adjacent north
−horizontal wall adjacent northwest
+ape corpse far southsoutheast
+horizontal wall far south
+horizontal wall near northeast
+southwest corner near southeast and southwest
+arrow near southsoutheast
+worthless piece of green glass near south
+southeast corner adjacent north and west
+northwest room corner adjacent northwest
<|action|>**search**
<|observation|>
<|@@|> −14 +14 <|@@|>
−Time: 21156
+Time: 21157
<|action|>**northeast**
...

F. Tuning and Training Details

F.1. Neural Language Models

All neural language agent experiments employ an off-the-shelf PyTorch (Paszke et al., 2017) implementation of the smallest GPT-2 model (Radford et al., 2019) distributed by the open-source HuggingFace Transformers library (Wolf et al., 2019). This model has approximately 120 million parameters. Experiments with pretrained models employ the pretrained weights shipped with this distribution. Tokenization is similarly performed using the default GPT-2 tokenizer from HuggingFace. Our instruction finetuning procedures heavily draw from those open-sourced by Wang et al. (2023b).

F.1.1. Extending GPT-2's Context Length

As mentioned in the main body of the paper, we extend the context length of GPT-2 in our NetHack finetuning experiments by a factor of four from 1024 to 4096 tokens. This is done by adding new positional encodings to the model. The weights of new encodings are initialized from $\mathcal{N}(0, 1)$, as is default in PyTorch.

F.1.2. Hyperparameters

We employ a batch size of 250 and a 32-epoch linear learning rate schedule with a warm-up ratio of 0.03 in all training and finetuning experiments with the exception of the ultra-low data BabyAI-Text experiment (1K demonstrations only). In this experiment, we tune models with the same batch size and learning rate schedule but for 64 epochs. Note that when we first started conducting experiments, we did not expect such a larger number of consecutive finetuning epochs to be beneficial to LM agent performance. However, preliminary runs evaluating agent generalization with *full rollouts* on held-out tasks confirmed this to be the case, with the effect particularly stark in our ultra-low sample BabyAI-Text finetuning settings.

Learning rate selection was performed once for full-text and `diff` history experiments by sweeping over the set $\gamma \in \{1 \cdot 10^{-3}, 3 \cdot 10^{-4}, 5 \cdot 10^{-5}, 1 \cdot 10^{-5}\}$. Models were tuned with action prediction losses for a single epoch on a $80/20$ split of each tuning dataset and were validated for perplexity on the held-out portion. We found $\gamma = 3 \cdot 10^{-4}$ to consistently be the best performing learning rate.

F.1.3. Hardware

All neural LM experiments were run using NVIDIA A100 GPU compute nodes with InfiniBand on an academic high performance computing cluster. BabyAI-Text finetuning experiments were carried out on single A100 GPUs, whereas NetHack finetuning experiments were conducted on two A100 GPUs at a time with a 48-hour budget per run. Model, optimizer, and data loader state checkpoints were saved frequently throughout the course of finetuning, and were used to restart and continue timed-out NetHack runs.

F.1.4. ZeRO++

Models were trained in mixed-precision with the `BFLOAT16` data format (Dean et al., 2012) and in a distributed fashion using the Microsoft DeepSpeed and the HuggingFace Accelerate libraries (Gugger et al., 2022). We employ the highly performant ZeRO++ protocol for stage three sharding (i.e. sharding of optimizer states, gradients, and model parameters) together with model parameter CPU offloading to dramatically accelerate experiments (Ren et al., 2021; Wang et al., 2023a). We also use gradient accumulation to support training with the desired batch size ($B = 250$).

F.2. NetHack Neural Vision Models

Vision-based state-of-the-art model architectures for NetHack (results reported as "LSTM (CDPGT5)" and "Transformer" in Table 1) are run exactly as originally implemented and open-soured during the NetHack Challenge Competition (Hambro et al., 2022a) and by Piterbarg et al. (2023), respectively. The Transformer-based model variant is referred to as a "flat transformer" in Piterbarg et al. (2023) and directly copies the architecture of CDPGT5, with the exception that the LSTM core module of the latter is replaced with a Transformer.

The history length of all vision agents during training is initialized to the maximum history length of the `LangHack` dataset, $H = 64$. All other training hyperparameters of these models are set as in Hambro et al. (2022b) and Piterbarg et al. (2023). Given our finding that very high epoch counts benefit the performance of tuned language agents, we similarly train vision agents for a very large number of epochs ($E = 32$) on `LangHack`. Model checkpoints are saved frequently during training.

We employ three random seeds to test both model variants, randomizing weight initialization and data loading order in each experiment.

F.2.1. On CDGPT5

The LSTM-based CDPGT5 model architecture won third place in the neural track of the NeurIPS 2021 NetHack Challenge Competition, outperforming all other data-driven neural agents (Hambro et al., 2022a). It is also used "off-the-shelf" by Tuyls et al. (2023) in their imitation learning scaling experiments, where the authors set state-of-the-art performance for neural agents on NetHack by dramatically scaling up training time and sample count to train a large version of this model on 115B expert action labels from the leading NetHack bot `AutoAscend`. Despite these scaling efforts, their neural vision agent still significantly under-performs the expert (Tuyls et al., 2023).

G. Evaluation Details

G.1. Neural Language Models

G.1.1. PHILOSOPHY

Our goal in this paper is to improve the quality of LM generations in low-resource sequential decision-making. As a result, we minimize the use of heuristics in our evaluations of neural language agents. Furthermore, rather than evaluating models by computing more standard language model metrics like perplexity over a validation dataset, we opt for a full "rollout"-style evaluation. Though this choice makes our experiments much more costly, it enables us to directly compare the rates of error accumulation associated with learned policies and adds rigour to our analysis.

At all timesteps of environment interaction, we generate token sequences from LMs with greedy decoding as implemented in HuggingFace Transformers (Wolf et al., 2019). We impose no restrictions on generated tokens.

As alluded to in Appendix B.2, we prompt models to begin generating an action token subsequence by appending the special token `<|action|>` to the end of recent interaction history. We also implement a custom stopping criterion, halting generation as soon as the special token `<|observation|>` is decoded from an LM.

G.1.2. HARDWARE

All neural LMs are evaluated on single NVIDIA RTX8000, A100, or A4000 GPUs on an academic high performance computing cluster.

G.1.3. BABYAI-TEXT

We evaluate all neural language models on 256 withheld instances of the BabyAI-Text task variants used in tuning with interaction history horizon $h = H = 4$. The generation of incorrectly formatted or invalid action token sequences three times in a row results in rollout termination.

G.1.4. NETHACK

As in our BabyAI-Text evaluation, all neural language agents tuned on NetHack demonstrations are evaluated in a standardized fashion. For each agent, we run a sweep over the maximal interaction history horizons $h \in \{4, 8, 15, 24, 64\}$ used to prompt LM action generation. In the event that h timesteps of previous interaction history exceed maximal LM context lengths (4096) at any given timestep, the history horizon included in the prompt is iteratively "back-tracked" across $h' : 1 \leq h' < h$ until this is no longer the case. The performance of models under each maximal history horizon is evaluated on a batch of 128 held-out instances of the NetHack video game via the NetHack Learning Environment (NLE) (Küttler et al., 2020).

We use precisely the same set of 128 game seeds in all evaluation sweeps. Furthermore, we employ the standard, "NetHack Challenge" variant of NLE in our evaluations, with inaction timeout set to $t = 100$ and no restrictions made over NetHack roles.

The mean NetHack neural language agent game scores reported throughout the paper reflect the most performant interaction history horizon h. Standard errors are computed across the corresponding evaluation batch.

G.2. NetHack Neural Vision Models

NetHack neural vision agents are evaluated exactly like NetHack neural language agents, with one exception: faster action generation with these baselines enables us to use a larger evaluation batch size of 1024 withheld game instances and to run more extensive sweeps over pixel-vector interaction history horizon values provided to Transformer-based agents, $h \in \{1, 2, 4, 8, 15, 24, 32, 64\}$. Note that the hidden state of LSTM-based agents (i.e. models with the CDGPT5 architecture (Hambro et al., 2022a)) are updated at each timestep throughout all games with no resetting, reflecting an effective agent interaction history horizon of $h = \infty$. The same evaluation procedure is conducted for checkpoints reflecting all $E = 32$ epochs of training.

As is the case for our neural language agents score statistics, for all vision models, we report mean NetHack game score reflecting the most performant checkpoints and interaction history horizon (for Transformer-based models) in Figure 1 and

Table 1.

Full learning curves of neural vision agent evaluation performance over the course of training (plotted to-scale against neural language model performance) are shown below in Figure G.2.

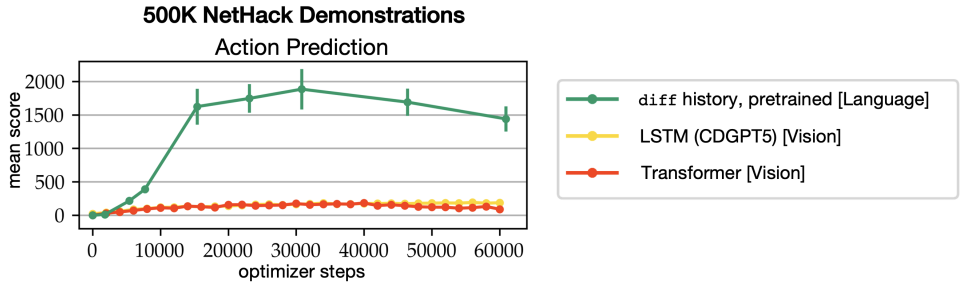

Figure 8. **Training state-of-the-art vision baseline architectures on `LangHack`**: We provide learning curves reflecting the performance of neural vision agent baselines trained on `LangHack` vs `diff` history neural language agents, to parallel those of Figure 5. Error bars indicate one standard error over an evaluation batch of withheld game instances.

H. Scaling diff History in NetHack

Below, we provide the learning curve of the diff history scaling experiment described in Section 4.4 as well as a visualization of the score distribution achieved by the scaled agent.

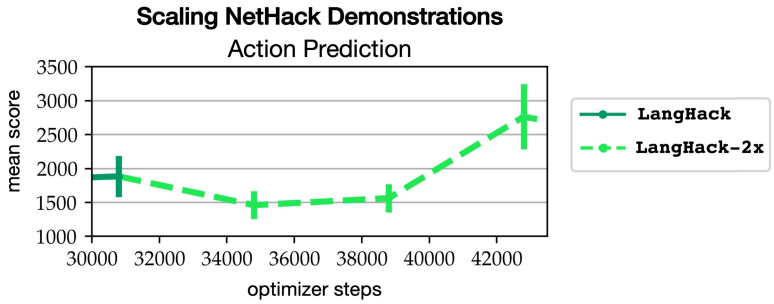

Figure 9. **Doubling NetHack interaction data**: We confirm that the performance of diff history agents scales with data samples by tuning the best-performing diff history agent checkpoint from our LangHack action prediction experiment on twice as many interaction demonstrations via LangHack-2x. This results in a 47% gain in mean NetHack score (see Table 1). Error bars indicate one standard error in score over a batch of withheld games.

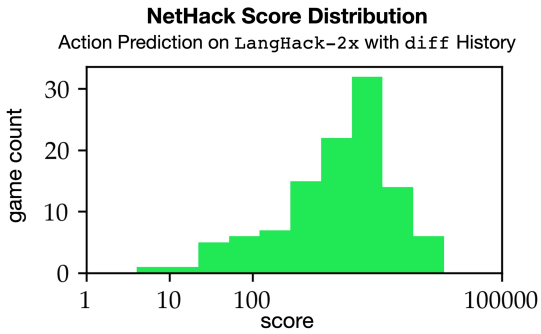

Figure 10. **NetHack game score distribution for best LangHack-2x checkpoint**: We demonstrate the distribution of game scores on withheld seeds of NetHack achieved by the most performant diff language agent trained on action prediction with LangHack-2x ($N = 128$). We observe a long tail in the performance of the agent, with the LM achieving a score of $\approx$ 20000 and successfully descending down to level eight of the dungeon in one of the games. Similar phenomena have been reported in previous work (Hambro et al., 2022a).