Towards mental time travel: a hierarchical memory for reinforcement learning agents

Andrew K. Lampinen
DeepMind
London, UK
lampinen@deepmind.com

Stephanie C. Y. Chan
DeepMind
London, UK
scychan@deepmind.com

Andrea Banino
DeepMind
London, UK
abanino@deepmind.com

Felix Hill
DeepMind
London, UK
felixhill@deepmind.com

Abstract

Reinforcement learning agents often forget details of the past, especially after delays or distractor tasks. Agents with common memory architectures struggle to recall and integrate across multiple timesteps of a past event, or even to recall the details of a single timestep that is followed by distractor tasks. To address these limitations, we propose a Hierarchical Chunk Attention Memory (HCAM), which helps agents to remember the past in detail. HCAM stores memories by dividing the past into chunks, and recalls by first performing high-level attention over coarse summaries of the chunks, and then performing detailed attention within only the most relevant chunks. An agent with HCAM can therefore "mentally time-travel"—remember past events in detail without attending to all intervening events. We show that agents with HCAM substantially outperform agents with other memory architectures at tasks requiring long-term recall, retention, or reasoning over memory. These include recalling where an object is hidden in a 3D environment, rapidly learning to navigate efficiently in a new neighborhood, and rapidly learning and retaining new object names. Agents with HCAM can extrapolate to task sequences an order of magnitude longer than they were trained on, and can even generalize zero-shot from a meta-learning setting to maintaining knowledge across episodes. HCAM improves agent sample efficiency, generalization, and generality (by solving tasks that previously required specialized architectures). Our work is a step towards agents that can learn, interact, and adapt in complex and temporally-extended environments.

1 Introduction

Human learning and generalization relies on our detailed memory of the past [54, 51, 16, 29, 42]. If we watch a show, we can generally recall its scenes in some detail afterward. If we explore a new neighborhood, we can recall our paths through it in order to plan new routes. If we are exposed to a new noun, we can find that object by name later. We experience relatively little interference from delays or intervening events. Indeed, human episodic memory has been compared to "mental time travel" [54, 51, 36]—we are able to *transport* ourselves into a past event and re-live it in sequential detail, without attending to everything that has happened since. That is, our recall is both sparse (we attend to a small chunk of the past, or a few chunks) *and* detailed (we recover a large amount of information

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

from each chunk). This combination gives humans the flexibility necessary to recover information from memory that is specifically relevant to the task at hand, even after long periods of time.

If we want Reinforcement Learning (RL) agents to meaningfully interact with complex environments over time, our agents will need to achieve this type of memory. They will need to remember event details. They will need to retain information they have acquired, despite unrelated intervening tasks. They will need to rapidly learn many pieces of new knowledge without discarding previous ones. They will need to reason over their memories to generalize beyond their training experience.

However, current models struggle to achieve rapid encoding combined with detailed recall, even over relatively short timescales. Meta-learning approaches [56] can slowly learn global task knowledge in order to rapidly learn new information, but they generally discard the details of that new information immediately to solve the next task. While LSTMs [20], Transformers [55], and variants like the TransformerXL [9] can serve as RL agent memories in short tasks [43], we show that they are ineffective at detailed recall even after a few minutes. Transformer attention can be ineffective at long ranges even in supervised tasks [53], and this problem is likely exacerbated by the sparse learning signals in RL. By contrast, prior episodic memory architectures [21, 14] can maintain information over slightly longer timescales, but they struggle with tasks that require recalling a temporally-structured event, rather than simply recalling a single past state. That is, they are particularly poor at the type of event recall that is fundamental to human memory. We suggest that this is because prior methods lack 1) memory chunks longer than a single timestep and 2) sparsity of attention. This means that these models cannot effectively "time-travel" to an event, and relive that specific event in detail, without interference from all other events. This also limits their ability to reason over their memories to adapt to new tasks—for example, planning a new path by combining pieces of previous ones [48].

Here, we propose a new type of memory that begins to address these challenges, by leveraging sparsity, chunking, and hierarchical attention. We refer to our architecture as a Hierarchical Chunk Attention Memory (HCAM). The fundamental insight of HCAM is to divide the past into distinct chunks before storing it in memory, and to recall hierarchically. To recall, HCAM first uses coarse chunk summaries to identify relevant chunks, and then mentally travels back to attend to each relevant chunk in further detail. This approach combines benefits of the sparse and relatively long-term recall ability of prior episodic memories [59, 48] with the short-term sequential and reasoning power of transformers [55, 9, 43] in order to achieve better recall and reasoning over memory than either prior approach. While we do not address the problem of optimally chunking the past here, instead employing arbitrary chunks of fixed length, our results show that even fixed chunking can substantially improve memory.

We show that HCAM allows RL agents to recall events over longer intervals and with greater sample efficiency than prior memories. Agents with HCAM can learn new words and then maintain them over distractor phases. They can extrapolate far beyond the training distribution, to maintain and recall memories after $10\times$ more distractors than during training, or after multiple episodes when trained only on single ones. Agents with HCAM can reason over their memories to plan paths near-optimally as they explore a new neighborhood, comparable to an agent architecture designed specifically for that task. HCAM is robust to hyperparameters, and agents with HCAM consistently outperform agents with Gated TransformerXL memories [43] that are twice as wide or twice as deep. Furthermore, HCAM is more robust to varying hyperparameters than other architectures (App. D.11). Thus HCAM provides a substantial improvement in the memory capabilities of RL agents.

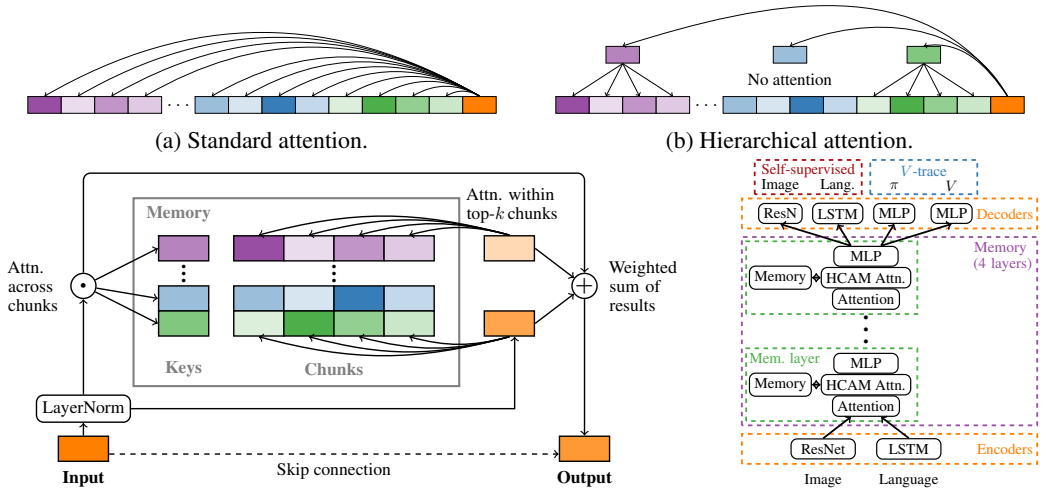
1.1 Background

Memory in RL Learning signals are sparse in RL, which can make it challenging to learn tasks requiring long memory. Thus, a variety of sophisticated memory architectures have been proposed for RL agents. Most memories are based on LSTMs [20], often with additional key-value episodic memories [59, 14]. Often, RL agents require self-supervised auxiliary training because sparse task rewards do not provide enough signal for the agent to learn what to store in memory [59, 14, 19].

Transformers Transformers [55] are sequence models that use attention rather than recurrence. These models—and their variants [9, 27, 57, 45]—have come to dominate natural language processing. Nevertheless, the success of transformers over LSTMs on sequential language tasks suggests that they might be effective agent memories, and a recent work succesfully used transformers as RL agent memories [43]—specifically, a gated version of TransformerXL [9]. However, using TransformerXL memories on challenging memory tasks still requires auxiliary self-supervision [19], and might even

benefit from new unsupervised learning mechanisms [3]. Furthermore, even in supervised tasks Transformers can struggle to recall details from long sequences [53]. In the next section, we propose a new hierarchical attention memory architecture for RL agents that can help overcome these challenges.

2 Hierarchical Chunk Attention Memory



(a) Standard attention. (b) Hierarchical attention.

(c) HCAM attention block. (d) HCAM in an agent.

Figure 1: HCAM motivation and implementation. (a) Standard transformer attention attends to every time step in a fixed window. (b) HCAM divides the past into chunks. It first attends to summaries of each chunk, and only attends in greater detail within relevant chunks. (c) HCAM is implemented using a key-value memory, where the keys are chunk summaries, and the values are chunk sequences. Top-level attention is performed over all summaries (left), and then standard attention is performed within the top-k chunks. The results are relevance-weighted and summed (right), and then added to a residual input to produce the block output. (d) Our agent encodes inputs, processes them through a 4-layer memory with an HCAM block in each layer, and decodes outputs for self-supervised and RL losses.

The HCAM uses a hierarchical strategy to attend over memory (Fig. 1). Specifically, it divides the past into chunks. It stores each chunk in detail—as a sequence—together with a single summary key for that chunk. It can then recall hierarchically, by first attending to chunk summaries to choose the top-k relevant chunks, and then attending within those most relevant chunks in further detail. In particular, let S denote the memory summary keys and Q be a linear projection ("query") layer. Then the input is normalized by a LayerNorm, and the chunk relevance (R) is computed as:

$$R = \text{softmax}(Q(\text{normed input}) \cdot S)$$

Then the model attends in detail within the chunks that have the top-k relevance scores. If the memory chunks are denoted by $\mathcal{C}$ and MHA(query inputs, key/value inputs) denotes multi-head attention:

$$\text{memory query results} = \sum_{i \in \text{top-}k \text{ from } R} R_i \cdot \text{MHA}(\text{normed input}, \mathcal{C}_i)$$

That is, HCAM selects the most relevant memory chunks, then time travels back into each of those chunks to query it in further detail, and finally aggregates the results weighted by relevance. This result is added to the input to produce the output of the block. An HCAM block can be *added* to a transformer layer, to supplement standard attention. (For an open-source HCAM implementation, see App. A.1)

HCAM can attend to any point in its memory, but at each step it only attends to a few chunks, which is much more compute- and memory-efficient than full attention. With N chunks in memory, each of size C, using HCAM over the top-k chunks requires $O(N + kC)$ attention computations, whereas full attention requires $O(NC)$. HCAM succeeds even with $k = 1$ in many cases (App. D.5), reducing computation $10\times$ compared to TrXL. Even with $k \geq 1$, HCAM often runs faster (App. D.10).

Agents Our agents (Fig. 1d) consist of input encoders (a CNN or ResNet for visual input, and an LSTM for language), a 4-layer memory (one of HCAM, a Gated TransformerXL [43], or an LSTM

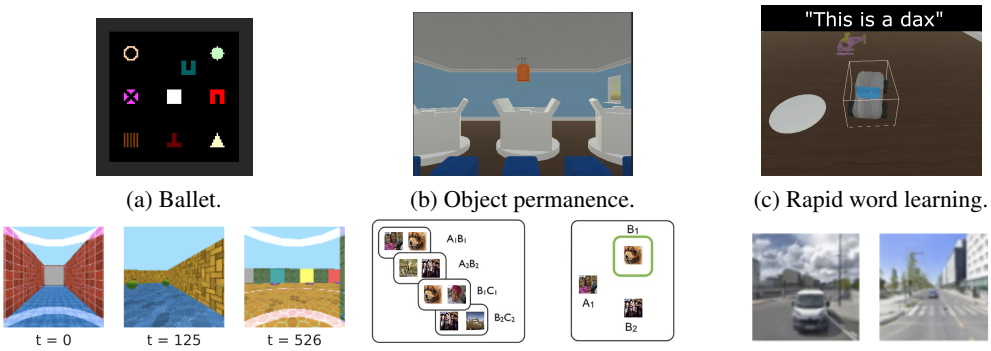
[20]), followed by output MLPs for policy, value, and auxiliary losses. Our agents are trained using IMPALA [11]. To use HCAM in the memory, we adapt the Gated TransformerXL agent memory [43]. We replace the XL memory with an HCAM block between the local attention (which provides short-term memory) and feed-forward blocks of each layer. We find that gating each layer [43] is not beneficial to HCAM on our tasks (App. D.7), so we removed it. See App. A.2 for further agent details.

We store the *initial layer inputs* in memory—that is, the inputs to the local attention—which performs better than storing the output of the local attention, and matches TransformerXL (TrXL) more closely. We accumulate these inputs into fixed-length memory chunks. When enough steps have accumulated to fill the current chunk, we add the current chunk to the memory. We average-pool across the chunk to generate a summary. We reset the memory between episodes, except where cross-episode memory is tested (Section 3.3). We stop gradients from flowing into the memory, just as TrXL [9] stops gradients flowing before the current window. This reduces challenges with the non-differentiability of top-k selection (see App. D.5 for discussion). It also means that we only need to store past activations in memory, and not the preceding computations, so we can efficiently store a relatively long time period.

However, the model must therefore learn to encode over shorter timescales, since it cannot update the encoders through the memory. Thus, as in prior work referenced above, we show that *self-supervised learning* [32], is necessary for successful learning on our tasks (App. D.6). Specifically, we use an auxiliary *reconstruction* loss—at each step we trained the agents (of all memory types) to reconstruct the input image and language as outputs [19]. This forces the agents to propagate these task-relevant features through memory, which ensures that they be encoded [cf. 21, 14].

3 Experiments

We evaluated HCAM in a wide variety of domains (Fig. 2). Our tasks test the ability of HCAM to recall event details (Sec. 3.1); to maintain object permanence (Sec. 3.2); to rapidly learn and recall new words, and extrapolate far beyond the training distribution (Sec. 3.3); and to reason over multiple memories to generalize, even in settings that previously required task-specific architectures (Sec. 3.4).



(a) Ballet. (b) Object permanence. (c) Rapid word learning.

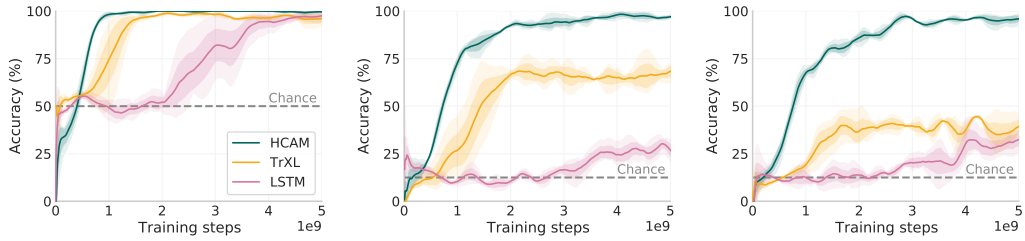
(d) Visual match. (e) Paired associative inference. (f) One-shot StreetLearn navigation.

Figure 2: We evaluated our model across a wide variety of task domains, some of which are modified or used from other recent memory papers [19, 21, 2, 48]. These include a variety of environments (2D, 3D, real images) and tasks (finding hidden objects, language learning, and navigation).

3.1 Remembering the ballet

Our first experiment tests the ability to recall spatiotemporal detail, which is important in real-world events. We place the agent in a 2D world, surrounded by "dancers" with different shapes and colors (Fig. 2a). One at a time, each dancer performs a 16-step solo dance (moves in a distinctive pattern), with a 16- or 48-step delay between dances. After all dances, the agent is cued to e.g. "go to the dancer who danced a square," and is rewarded if it goes to the correct dancer. This task requires recalling dances in detail, because no single step of a dance is sufficient to distinguish it from other dances. The task difficulty depends on the number of dances the agent must watch in an episode (between 2 and 8) and the delay length. We varied both factors across episodes during training—this likely increases the memory challenges, since the agent must adapt in different episodes to recalling different amounts of information for varying periods. See App. B.2 for details.

Fig. 3 shows the results. Agents with an HCAM or a Gated TransformerXL (TrXL) memory perform well at the shortest task, with only 2 dances per episode (Fig. 3a). LSTM agents learn much more slowly, showing the benefits of attention-based memories. With 8 dances per episode (Fig. 3b), HCAM continues to perform well. However, TrXL's performance degrades, **even though the entire task is short enough to fit within a single TrXL attention window**. (The dances are largely in the XL region, where gradients are stopped for TrXL, but HCAM has the same limitation.) LSTM performance also degrades substantially. The advantage of HCAM is even more apparent with longer delays between the dances (Fig. 3c). HCAM can also generalize well ($\geq 90\%$) from training on up to 6 dances to testing on 8 (App. D.2). Furthermore, HCAM is robust to varying memory chunk sizes (App. D.4), even if the chunk segmentation is not task aligned. Finally, HCAM can perform well on this task even if it is only allowed to select a single memory chunk rather than the top-k (App. D.5), thereby attending to $8\times$ fewer time-points than TrXL (while a sparse TrXL is not advantageous, see App. D.9). Thus HCAM can increase computational efficiency. In summary, both attention-based memories recall events better than LSTMs, and HCAM robustly outperforms TrXL at all but the easiest tasks.



(a) 2 dances. (b) 8 dances. (c) 8 dances, long delays.

Figure 3: The ballet task—HCAM (teal) outperforms TrXL (yellow) and LSTM (red) memories at difficult ballets with many dances, especially when there are long delays between. (a) In the shortest ballets, TrXL performs nearly as well as HCAM, while LSTMs require much more training. (b) With 8 dances, HCAM substantially outperforms TrXL, even though the entire task fits within one TrXL attention window. LSTMs do not get far above chance at choosing among the dancers. (c) With longer delays between dances, HCAM performs similarly but TrXL degrades further. (3 runs per condition, lines=means, light regions=range, dark=$\pm$SD. Training steps are agent/environment steps, not the number of parameter updates. Chance denotes random choice among the dancers, not random actions.)

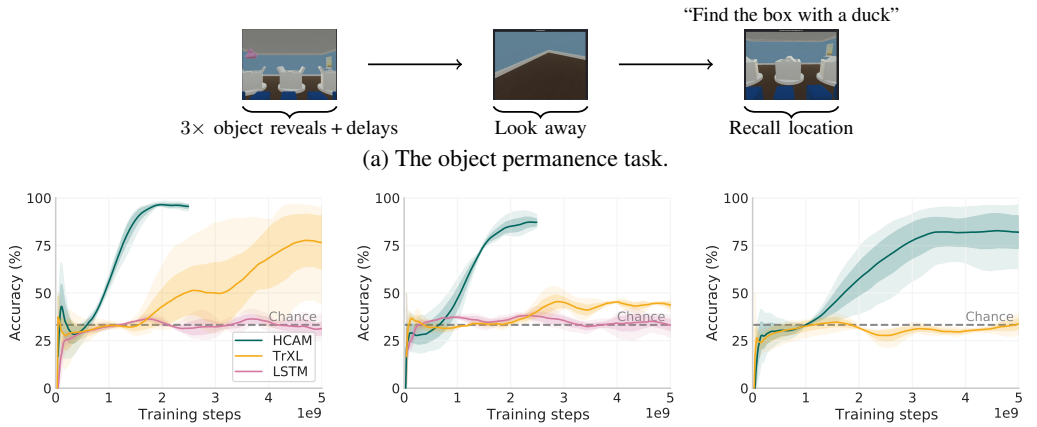
3.2 Object permanence

Our next tasks test the ability to remember the identity and location of objects after they have been hidden within visually-identical boxes, despite delays and looking away. These tasks are inspired by work on object permanence in developmental psychology [e.g. 1]. We implemented these tasks in a 3D Unity environment. The agent is placed in a room and fixed in a position where it can see three boxes (Fig. 4a). One at a time, an object jumps out of each box a few times, and then returns to concealment inside. There may be a delay between one object and the next. After all objects have appeared, the box lids close. Then, the agent is released, and must face backwards. Finally, it is asked to go to the box with a particular object, and rewarded if it succeeds.

HCAM solves these tasks more effectively than the baselines. HCAM learns the shortest tasks (with no delays) substantially faster than TrXL (Fig. 4b), and LSTMs are unable to learn at all. When trained on tasks of varying lengths, HCAM was also able to master tasks with 30 seconds of delay after each object revealed itself, while TrXL performed near chance even with twice as much training (Fig, 4b). Furthermore, HCAM can learn these long tasks even without the shorter tasks to scaffold it (Fig. 4d).

3.3 Rapid word learning with distractor tasks & generalization across episodes

The preceding tasks focused on passively absorbing events, and then recalling them after delays. We next turn to a set of rapid-word-learning tasks (Fig. 2c) that require actively acquiring knowledge, and maintaining it in more challenging settings, with intervening distractor tasks (Fig: 5a). These tasks are based on the work of Hill et al. [19], who showed that transformer-based agents in a 3D environment can meta-learn to bind objects to their names after learning each name only once. That is, immediately after a single exposure to novel object names, their agent can successfully lift objects



(a) The object permanence task.

(b) No delays, varying training. (c) Long delays, varying training. (d) Long delays, long-only training.

Figure 4: The object permanence task—HCAM learns faster than TrXL, and succeeds at longer tasks. (a) The task structure. (b) The shortest tasks, with no delays. When trained with varying delay lengths, HCAM rapidly and consistently learns to track object locations. TrXL learns more slowly and less consistently. (b) Long tasks, with 30-second delays between each object being revealed. HCAM learns quite well, while TrXL barely achieves above-chance performance. (d) HCAM can learn the long tasks even without simultaneous training on the shorter tasks to scaffold its learning. (3 runs per condition. Chance denotes random choice among the boxes, not random actions.)

specified by those names. Their agent then discards its memory before proceeding to the next episode. However, in more complex environments tasks will not be so cleanly structured. There will often be distracting events and delays—or even learning of other information—before the agent is tested.

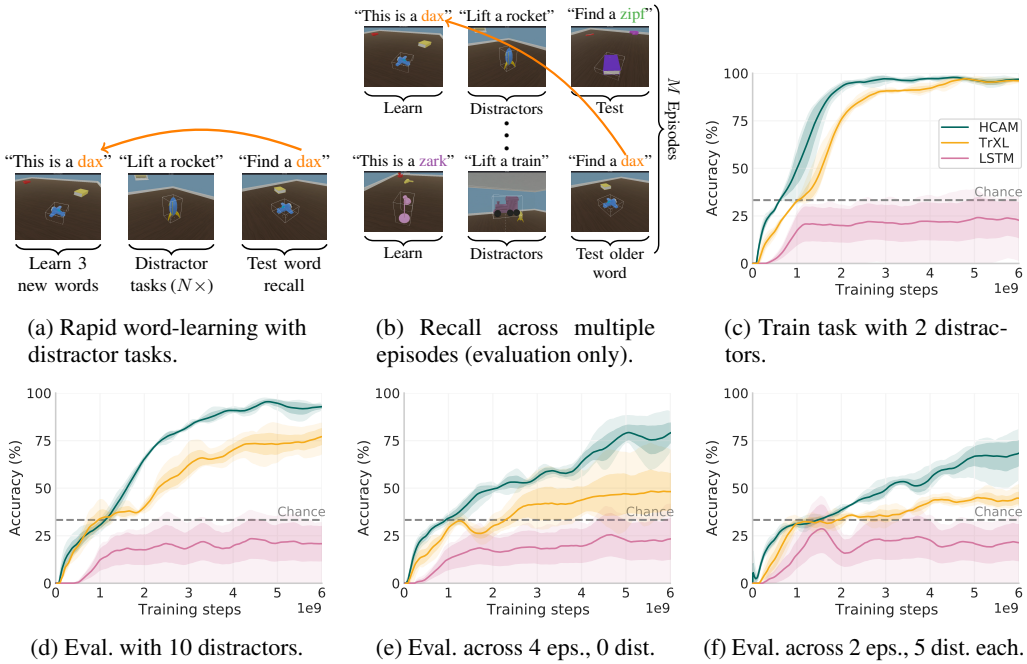
We therefore created more challenging versions of these tasks by inserting distractor tasks between the learning and test phases. In each distractor task we placed the agent in a room with a fixed set of three objects (that were never used for the word-learning portion), and asked it to lift one of those objects (using fixed names). By varying the number of distractor phases, we were able to manipulate the demands on the memory. We trained our agents on tasks with 0, 1, or 2 distractor phases, and evaluated their ability to generalize to tasks with more distractors. HCAM and TrXL could learn the training tasks equally well, and both showed some ability to generalize to longer tasks (App. D.1). However, HCAM was able extrapolate better to tasks with 10 distractor phases (Fig. 5d), and even showed some extrapolation to 20 distractors, an order of magnitude more than trained (Fig. 7a).

Maintaining knowledge across episodes zero-shot It is a fundamental limitation of most meta-learning paradigms that they discard the information they have learned after each episode. The agents used by Hill et al. [19] discarded their memories end of each episode. Children, by contrast, can rapidly learn new words and maintain that knowledge even after learning other new words. This leads to a question: if HCAM can effectively maintain a word-object mapping despite intervening distractor tasks, could it maintain this knowledge across entire episodes of learning new words?

To answer this question, we used the agents trained on single-episode tasks. We evaluated these agents on tasks where they were sometimes tested on a word from several episodes before, despite having learned and been tested on other words in the intervening period (Fig. 5b). In training, the agents were never required to maintain a set of words once they encountered a new learning phase with new words. Nevertheless, our agent with HCAM is able to generalize well to recalling words four episodes later (Fig. 5e). It can even extrapolate along multiple dimensions, recalling words two episodes later with more distractor phases per episode than it ever saw in training (Fig. 5f). In App. D.3 we show how the attention patterns of the hierarchical memory support this generalization across episodes. **Agents with HCAM are able to generalize to memory tasks that are far from the training distribution.**

3.4 Comparisons with alternative memory systems

There has been an increasing interest in architectures for memory in RL, as well as in deep learning more broadly. In order to help situate our approach within this literature, we benchmarked our model



(a) Rapid word-learning with distractor tasks.

(b) Recall across multiple episodes (evaluation only).

(c) Train task with 2 distractors.

(d) Eval. with 10 distractors.

(e) Eval. across 4 eps., 0 dist.

(f) Eval. across 2 eps., 5 dist. each.

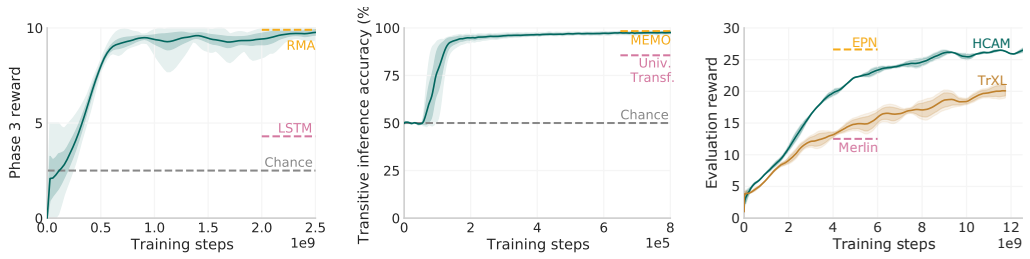
Figure 5: The rapid word-learning tasks. (a) Training tasks consist of learning three words (nouns), completing distractor tasks, and then finding an object named using one of the three learned words. We evaluated generalization to larger numbers of distractors. (b) During training, the agent never had to recall words for more than a single episode. However, we also evaluated the agent on its ability to recall a word learned several episodes earlier. This tests the ability of our memory to bridge from rapid meta-learning to longer-term retention. (c) Agents with HCAM or TrXL memories learn the hardest training task (2 distractors) well, although the agent with HCAM is faster to learn. When trained only on single episodes with 0-2 distractor phases: (d) HCAM outperforms TrXL at extrapolation to 10 distractor phases. (e-f) HCAM can generalize strongly out-of-distribution to recalling words that were learned several episodes earlier, despite having completed other episodes in the intervening time. (3 runs per condition. Chance denotes random performance on the final evaluation choice; this requires complex behaviours: learning all names and completing all distractors and intermediate episodes to reach the final choice. This is why LSTM performance is below chance in one seed.)

against tasks used in several recent papers exploring alternative forms of memory. HCAM is able to perform well across all the tasks—reaching near-optimal performance, comparable with memory architectures that were specifically engineered for each task—and outperforms strong baselines.

Passive Visual Match First, we compare to the Passive Visual Match task from Hung et al. [21]. In this task, the agent must remember a color it sees at the beginning of the episode, in order to choose a matching color at the end (Fig. 2d). HCAM achieves reliably accurate performance (Fig. 6a), comparable to RMA, the best model from Hung et al. [21]. It would be interesting to combine HCAM with the value transport idea Hung et al. proposed, in order to solve their active tasks.

Paired Associative Inference We next considered the Paired Associative Inference task [2]. In each episode, the agent receives a set of pairings of stimuli, and it must chain together transitive inferences over these pairings in order to respond to a probe (Fig. 2e). These tasks therefore require sequential reasoning over multiple memories. HCAM achieves comparable performance to MEMO on the length 3 version of this task, and substantially outperforms baselines like Universal Transformer [10]. This is likely because HCAM, like MEMO, respects the chunk structure of experiencing pairs of images.

One-Shot StreetLearn Finally, we evaluated HCAM on One-Shot StreetLearn (Fig. 2f) proposed by Ritter et al. [48], based on Mirowski et al. [39]. One-shot StreetLearn is a challenging meta-learning setting. The agent is placed in a neighborhood, and must navigate to as many goals as it can within a fixed time. The agent receives its current state and goal as visual inputs (Google StreetView images) and must learn about the neighborhood in early tasks within an episode, in order to navigate efficiently later. Ritter et al. designed a specialized Episodic Planning Network (EPN) to



(a) Passive Visual Match. (b) Paired Associative Inference. (c) One-Shot StreetLearn.

Figure 6: Evaluating HCAM on tasks from other memory papers. Our agent achieves performance competitive with the specialized model proposed to solve each task. Top results (gold) and selected baselines (red) from each paper are indicated. (a) The Passive Visual Match task [21]. (a) The Paired Associative Inference task [2]. HCAM achieves near optimal performance, comparable to MEMO and above Universal Transformers [10]. (c) The One-Shot StreetLearn environment [48]. HCAM achieves comparable performance to EPN on this difficult navigation task, although it is slower to learn. Both substantially outperform strong baselines. (Some prior results were estimated from figures. HCAM results aggregate over: (a) 6 seeds. (b) 3 seeds, selected by validation accuracy. (c) 2 seeds.)

solve these tasks. HCAM—despite its more general purpose architecture—can achieve comparable performance to EPN (although HCAM is somewhat slower to reach this level of performance). Ritter et al. showed that EPN achieves near-optimal planning in later parts of each episode, and therefore **HCAM must be planning close to optimally to match EPN's performance.** Strong baselines perform much worse. Merlin [59]—a strong agent with a learned episodic key-value memory and auxiliary unsupervised training—only achieves around half the performance of HCAM and EPN. TrXL is also less effective. This highlights the value of more sophisticated memory architectures, and in particular emphasizes the value of sequences stored in memory, rather than single vectors: EPN stores transitions (sequences of length 2) and HCAM stores longer sequences, while Merlin has only single vector values for each key (as does TrXL).

4 Discussion

In this paper we have proposed the Hierarchical Chunk Attention Memory. HCAM allows agents to attend in detail to past events without attending to irrelevant intervening information. Thus, HCAM effectively implements a form of "mental time-travel" [54]. Our main insight is that this ability can be achieved using a hierarchically structured memory that sparsely distributes detailed attention. Specifically, recall uses attention over memory chunk summaries to allocate more detailed attention within only the most relevant chunks. We see this as a potentially important insight, because both mental time travel [51] and hierarchical structure [16] are essential to the power of human memory. Furthermore, recent work has suggested that human memory behaves more like a "quantum wave function" distribution over several past events [36], which matches our approach of a distribution over the top-k relevant chunks. HCAM incorporates a number of important features of human memory.

Correspondingly, we showed that agents with HCAM succeed at a wide range of environments and tasks that humans depend on memory to solve. HCAM allows agents to remember sequential events, such as a ballet, in detail. HCAM allows agents to rapidly learn to navigate near-optimally in a new neighborhood by planning over previous paths; to perform transitive inferences; and to maintain object permanence, despite delays, occlusion and looking away. HCAM allows agents to effectively learn new words from a single exposure, and maintain that knowledge across distractor tasks. The agents can even extrapolate far beyond the training distribution to recall words after subsequent learning episodes **without ever being trained to do so**—better memory systems can help with the challenge of bridging from meta-learning to continual learning, which is receiving increasing interest [13, 17, 48, 5, 40].

The tasks we considered are challenging because they rely on flexible use of memory. The agent did not know in advance which paths to remember in One-Shot StreetLearn, or which dancer would be chosen in Ballet. Furthermore, we trained the agent simultaneously on procedurally generated episodes where task features varied—e.g. the agent did not know how many distractor phases would appear in an episode, or the length of delays, so it could not rely on a fixed policy that recalls information after a fixed time. HCAM might be especially useful in these difficult, variable settings.

HCAM is robust to hyperparameters such as chunk size (App. D.4) and the number k of memories selected at each layer and step (App. D.5)—it is even able to solve many tasks with $k = 1$. It also outperforms TrXL models that are either twice as wide or twice as deep, and correspondingly have many more parameters and, in the deeper case, make many more attention computations (App. D.8). Our comparisons to other approaches (Sec. 3.4) show that our approach is competitive with state of the art, problem-specific memory architectures, and outperforms strong baselines like Merlin and Universal Transformers. Finally, in hyperparameter sweeps suggested by our reviewers to improve TrXL's performance, we found that HCAM was consistently more robust to hyperparameter variation (App. D.11). These observations suggest that our results should be relatively generalizable.

Self-supervised learning As in prior memory work [59, 14, 19] we found that self-supervised learning was necessary to train the agent to store all task-relevant information in memory. Training the agent to reconstruct its input observations as outputs was sufficient in the tasks we explored, but more sophisticated forms of auxiliary learning [23, 3] might be useful in other settings.

Memory in RL vs. supervised settings Our work has focused on improving the memory of a situated RL agent. Compared to supervised settings such as language modelling, where a vanilla transformer can achieve very impressive performance [47, 4], RL poses unique challenges to memory architectures. First, sparse rewards present a more impoverished learning signal than settings that provide detailed errors for each output. The ability of HCAM to restore a memory in detail may help ameliorate this problem. Second, the multimodal (language + vision) stimuli experienced by our agent contains much more information per step than the word-pieces alone that a language model experiences, and therefore our setting may place more demands on the memory architecture. Finally, our tasks require access to detailed, structural aspects of past memories, while many existing NLP tasks do not depend significantly on structure — for example, Pham et al. [44] show that BERT ignores word order when solving many language understanding benchmarks, and produces equivalent outputs for shuffled inputs. Detailed memory will be most beneficial in settings in which the sequential structure of the past is important.

Nonetheless, we do not rule out possible applications of HCAM in supervised settings, such as language processing [47, 4], video understanding [52], or multimodal perception [24]. HCAM would likely be most beneficial in tasks that strongly rely on long-term context and structure, e.g. the full-length version of NarrativeQA [28]. Because videos often contain hierarchically-structured events, video models might especially benefit from HCAM-style attention. More broadly, the greater efficiency of sparse, hierarchical attention might allow detailed memory even in settings with limited computational resources, such as embedded systems.

Episodic memories Nematzadeh et al. [41] suggested endowing transformers with external episodic memories. Several recent papers have proposed systems that can be interpreted as episodic memories from this perspective, specifically looking up related contexts from the training corpus using either nearest neighbors [25], or attention [61]. However, these approaches have generally only used this type of external memory to predict outputs, rather than allowing the model to perform further computations over the memories it recalls, as in HCAM. Thus, these memories would be inadequate for most RL tasks, which require planning or reasoning with memories.

Various works have proposed other types of external/episodic memory, both in RL specifically [e.g. 59, 21, 14] and in deep learning more broadly [e.g. 49, 15]. These approaches have generally not stored memories with the hierarchical summary-chunk structure of HCAM. Ritter et al. [48] proposed an episodic memory for navigation tasks that stores state transitions; this architecture can be seen as a step towards our approach of storing sequences as chunks. Indeed, in the challenging city navigation domain that Ritter et al. explored, we showed that HCAM achieves comparable performance to their EPN architecture, despite HCAM being much more general. Furthermore, both HCAM and EPN substantially outperform Merlin [59], a strong baseline that learns to store vector memories, rather than the rich representations of sequences stored by HCAM (or transitions stored by EPN). This highlights the value of rich, sequential memories. We suggest that episodic memories could benefit from moving beyond the idea of keys and values as single vectors. **It can be useful to store more general structures—such as a time-sequence of states—as a single "value" in memory.**

One benefit of episodic memory is that memory replay can support learning [29, 37], in particular by ameliorating catastrophic interference. It would therefore be interesting to explore whether HCAM's memory could be used for training. For example, could memory summaries be used to locate similar or contrasting memories that should be interleaved together [38] to improve continual learning?

Transformer memories Many recent works have attempted to improve the computational efficiency of transformer attention over long sequences [e.g. 27, 57]. However, these approaches often perform poorly at even moderately long tasks [53]. Similarly, we found that on tasks like Ballet or One-Shot StreetLearn, Transformer-XL can perform suboptimally even when the entire task fits within a single, relatively short attention window. However, there could potentially be complementary benefits to combining approaches to obtain even more effective attention with hierarchical memory, which should be investigated in future work. In particular, some recent work has proposed a simple inductive bias which allows Transformers to benefit from evaluation lengths much longer than they were trained on [45]—while this strategy would likely not help with recalling specific instances in detail, it might be complementary to an approach like ours.

Other works have used a hierarchy of transformers with different timescales for supervised tasks [33, 63, 60, 31], for example encoding sentences with one transformer, and then using these embeddings as higher-level tokens. This approach improves performance on tasks like document summarization, thus supporting the value of hierarchy. Luong et al. [35] also showed benefits of both local and global attention computation in LSTM-based models. However, we are not aware of prior transformers in which the coarse computations are used to select chunks for more detailed computation, as in HCAM (although recent work has demonstrated top-k patch selection in CNNs [8]); nor are we aware of prior works that have demonstrated their approaches in RL, or beyond a single task domain.

Memory and adaptation One major benefit of memory is that a model can flexibly use its memories in a goal- or context-dependent way in order to adapt to new tasks [54, 51, 29]. Adult humans use our recall of rich, contextual memories in order to generalize effectively [51, 42]. While our tasks required some forms of goal-dependent memory use—e.g. combining old paths to plan new ones—it would be interesting to evaluate more drastic forms of adaptation. Recent work shows that transforming prior task representations allows zero-shot adaptation to substantially altered tasks [30]. HCAM could potentially learn to transform and combine memories of prior tasks in order to adapt to radically different settings. Exploring these possibilities offers an exciting future direction.

Limitations & future directions While we see our contribution as a step towards more effective mental time-travel for agent memory, many aspects could be further improved. Our implementation requires the ability to keep each previous step in (hardware) memory, even if the chunk containing that step is irrelevant. This approach would be challenging to scale to the entire lifetime of episodic memory that humans retain. Nonetheless, storing the past is feasible up to tens of thousands of steps on current accelerators. This could be extended further by using efficient kNN implementations, which have recently been used to perform kNN lookup over an entire language dataset [25].

The challenge of scaling to longer-term memory would be helped by deciding what to store, as in some episodic memory models [15], soft forgetting of rarely retrieved memories, and similar mechanisms. However, such mechanisms are simultaneously limiting, in that the model may be unable to recall knowledge that was not obviously useful at the time of encoding. Over longer timescales, HCAM might also benefit from including more layers of hierarchy—grouping memory chunks into higher-order chunks recursively to achieve logarithmic complexity for recall. Even over moderate timescales, more intelligent segmentation of memory into chunks would potentially be beneficial—human encoding and recall depends upon complex strategies for event segmentation [62, 50, 6, 34, 7]. HCAM might also benefit from improved chunk summaries; mean-pooling over the chunk is a relatively naïve approach to summarization, so learning a compression mechanism might be beneficial [46]. These possibilities provide exciting directions for future work.

Conclusions We have proposed a Hierarchical Chunk Attention Memory for RL agents. This architecture allows agents to recall their most relevant memories in detail, and to reason over those memories to achieve new goals. This approach outperforms (or matches the optimal performance of) a wide variety of baselines, across a wide variety of task domains. It allows agents to remember where objects were hidden, and to efficiently learn to navigate in a new neighborhood by planning from memory. It allows agents to recall words they have learned despite distracting intervening tasks, and even across episodes. These abilities to learn, adapt, and maintain new knowledge are critical to intelligent behavior, especially as the field progresses towards more complex environments. We hope that our work will motivate further exploration of hierarchically structured memories, in RL and beyond. Hierarchical memory may have many benefits for learning, reasoning, adaptation, and intermediate- or long-term recall.

Acknowledgments and Disclosure of Funding

We would like to acknowledge Adam Santoro, Emilio Parisotto, Jay McClelland, David Raposo, Wilka Carvalho, Tim Scholtes, Tamara von Glehn, Sébastien Racanière, and the anonymous reviewers for helpful comments and suggestions.

References

[1] Renee Baillargeon. Infants' reasoning about hidden objects: evidence for event-general and event-specific expectations. *Developmental science*, 7(4):391–414, 2004.

[2] Andrea Banino, Adrià Puigdomènech Badia, Raphael Köster, Martin J Chadwick, Vinicius Zambaldi, Demis Hassabis, Caswell Barry, Matthew Botvinick, Dharshan Kumaran, and Charles Blundell. Memo: A deep network for flexible combination of episodic memories. In *International Conference on Learning Representations*, 2020.

[3] Andrea Banino, Adrià Puidomenech Badia, Jacob Walker, Tim Scholtes, Jovana Mitrovic, and Charles Blundell. Coberl: Contrastive bert for reinforcement learning. *arXiv preprint arXiv:2107.05431*, 2021.

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[5] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexande Lacoste, David Vazquez, et al. Online fast adaptation and knowledge accumulation: a new approach to continual learning. *arXiv preprint arXiv:2003.05856*, 2020.

[6] Stephanie CY Chan, Marissa C Applegate, Neal W Morton, Sean M Polyn, and Kenneth A Norman. Lingering representations of stimuli influence recall organization. *Neuropsychologia*, 97:72–82, 2017.

[7] Claire H. C. Chang, Christina Lazaridi, Yaara Yeshurun, Kenneth A. Norman, and Uri Hasson. Relating the Past with the Present: Information Integration and Segregation during Ongoing Narrative Processing. *Journal of Cognitive Neuroscience*, pages 1–23, 04 2021. ISSN 0898-929X. doi: 10.1162/jocn_a_01707. URL `https://doi.org/10.1162/jocn_a_01707`.

[8] Jean-Baptiste Cordonnier, Aravindh Mahendran, Alexey Dosovitskiy, Dirk Weissenborn, Jakob Uszkoreit, and Thomas Unterthiner. Differentiable patch selection for image recognition. *arXiv preprint arXiv:2104.03059*, 2021.

[9] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[10] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[11] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[12] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[13] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. In *International Conference on Machine Learning*, pages 1920–1930. PMLR, 2019.

[14] Meire Fortunato, Melissa Tan, Ryan Faulkner, Steven Hansen, Adrià Puigdomènech Badia, Gavin Buttimore, Charlie Deck, Joel Z Leibo, and Charles Blundell. Generalization of reinforcement learners with working and episodic memory. *arXiv preprint arXiv:1910.13406*, 2019.

[15] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538 (7626):471–476, 2016.

[16] Uri Hasson, Janice Chen, and Christopher J Honey. Hierarchical process memory: memory as an integral component of information processing. *Trends in cognitive sciences*, 19(6):304–313, 2015.

[17] Xu He, Jakub Sygnowski, Alexandre Galashov, Andrei A Rusu, Yee Whye Teh, and Razvan Pascanu. Task agnostic continual learning via meta learning. *arXiv preprint arXiv:1906.05201*, 2019.

[18] Felix Hill, Andrew Lampinen, Rosalia Schneider, Stephen Clark, Matthew Botvinick, James L McClelland, and Adam Santoro. Environmental drivers of systematicity and generalization in a situated agent. In *International Conference on Learning Representations*, 2020.

[19] Felix Hill, Olivier Tieleman, Tamara von Glehn, Nathaniel Wong, Hamza Merzic, and Stephen Clark. Grounded language learning fast and slow. In *International Conference on Learning Representations*, 2021.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[21] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value. *Nature communications*, 10(1):1–12, 2019.

[22] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9 (3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[23] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*, 2016.

[24] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. *arXiv preprint arXiv:2103.03206*, 2021.

[25] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2020.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[28] Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

[29] Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

[30] Andrew K Lampinen and James L McClelland. Transforming task representations to perform novel tasks. *Proceedings of the National Academy of Sciences*, 117(52):32970–32981, 2020.

[31] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C Paulson. Isarstep: a benchmark for high-level mathematical reasoning. In *International Conference on Learning Representations*, 2020.

[32] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Zhaoyu Wang, Li Mian, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *arXiv preprint arXiv:2006.08218*, 1(2), 2020.

[33] Yang Liu and Mirella Lapata. Hierarchical transformers for multi-document summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5070–5081, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1500. URL `https://www.aclweb.org/anthology/P19-1500`.

[34] Qihong Lu, Uri Hasson, and Kenneth A Norman. Learning to use episodic memory for event prediction. *bioRxiv*, 2020.

[35] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[36] Jeremy R Manning. Episodic memory: Mental time travel or a quantum "memory wave" function? *Psychological Review*, 2021.

[37] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[38] James L McClelland, Bruce L McNaughton, and Andrew K Lampinen. Integration of new information in memory: new insights from a complementary learning systems perspective. *Philosophical Transactions of the Royal Society B*, 375(1799):20190637, 2020.

[39] Piotr Mirowski, Andras Banki-Horvath, Keith Anderson, Denis Teplyashin, Karl Moritz Hermann, Mateusz Malinowski, Matthew Koichi Grimes, Karen Simonyan, Koray Kavukcuoglu, Andrew Zisserman, et al. The streetlearn environment and dataset. *arXiv preprint arXiv:1903.01292*, 2019.

[40] Arseny Moskvichev and James A Liu. Updater-extractor architecture for inductive world state representations. *arXiv preprint arXiv:2104.05500*, 2021.

[41] Aida Nematzadeh, Sebastian Ruder, and Dani Yogatama. On memory in human and artificial language processing systems. In *Bridging AI and Cognitive Science Workshop at ICLR 2020*, 2020.

[42] Chi T. Ngo, Susan L. Benear, Haroon Popal, Ingrid R. Olson, and Nora S. Newcombe. Contingency of semantic generalization on episodic specificity varies across development. *Current Biology*, 2021. ISSN 0960-9822. doi: https://doi.org/10.1016/j.cub.2021.03.088. URL `https://www.sciencedirect.com/science/article/pii/S0960982221004619`.

[43] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pages 7487–7498. PMLR, 2020.

[44] Thang M Pham, Trung Bui, Long Mai, and Anh Nguyen. Out of order: How important is the sequential order of words in a sentence in natural language understanding tasks? *arXiv preprint arXiv:2012.15180*, 2020.

[45] Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

[46] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

[47] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 2020.

[48] Samuel Ritter, Ryan Faulkner, Laurent Sartran, Adam Santoro, Matt M Botvinick, and David Raposo. Rapid task-solving in novel environments. In *International Conference on Learning Representations*, 2021.

[49] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

[50] Ignasi Sols, Sarah DuBrow, Lila Davachi, and Lluís Fuentemilla. Event boundaries trigger rapid memory reinstatement of the prior events to promote their representation in long-term memory. *Current Biology*, 27(22):3499–3504, 2017.

[51] Thomas Suddendorf, Donna Rose Addis, and Michael C Corballis. Mental time travel and the shaping of the human mind. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1521):1317–1324, 2009.

[52] Chen Sun, Fabien Baradel, Kevin Murphy, and Cordelia Schmid. Learning video representations using contrastive bidirectional transformer. *arXiv preprint arXiv:1906.05743*, 2019.

[53] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qVyeW-grC2k`.

[54] Endel Tulving. Memory and consciousness. *Canadian Psychology/Psychologie canadienne*, 26 (1):1, 1985.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[56] Jane X Wang. Meta-learning in natural and artificial intelligence. *Current Opinion in Behavioral Sciences*, 38:90–95, 2021.

[57] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[58] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi: 10.21105/joss.03021. URL `https://doi.org/10.21105/joss.03021`.

[59] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

[60] Linyi Yang, Tin Lok James Ng, Barry Smyth, and Riuhai Dong. Html: Hierarchical transformer-based multi-task learning for volatility prediction. In *Proceedings of The Web Conference 2020*, pages 441–451, 2020.

[61] Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. Adaptive semiparametric language models. *arXiv preprint arXiv:2102.02557*, 2021.

[62] Jeffrey M Zacks and Khena M Swallow. Event segmentation. *Current directions in psychological science*, 16(2):80–84, 2007.

[63] Xingxing Zhang, Furu Wei, and Ming Zhou. Hibert: Document level pre-training of hierarchical bidirectional transformers for document summarization. *arXiv preprint arXiv:1905.06566*, 2019.

A Methods

A.1 Open source attention module

We have open-sourced a Jax/Haiku implementation of our Hierarchical Attention Module, which can be found at: `https://github.com/deepmind/deepmind-research/tree/master/hierarchical_transformer_memory/hierarchical_attention`

We have also released our Ballet and Rapid Word Learning environments, and include links to those and the released versions of other environments we used below.

A.2 Agent architecture and training

Table 1: Hyperparameters used in main experiments. Where only one value is listed across multiple columns, it applies to all. Where hyperparameters were swept, parameters used for HCAM and TrXL are reported separately.

	Ballet	**Objects**	**Words**	**Image**	**Streets**	**Associative**
All activation fns	ReLU					
State dimension	512					
Memory dimension	512					
Memory layers	4					
Memory num. heads	8					
HCAM chunk size	32	16	16	16	4	2
HCAM chunk overlap	-				1	-
HCAM top-k	8	8	16	16	32	32
HCAM total num. chunks	always greater than max episode length // chunk size					
TrXL extra length	256	512	256	NA	200 (full)	NA
Visual encoder	CNN	ResNet				
Vis. enc. channels	(16, 32, 32)					
Vis. enc. filt. size	(9, 3, 3)	(3, 3, 3)				
Vis. enc. filt. stride	(9, 1, 1)	(2, 2, 2)				
Vis. enc. num. blocks	NA	(2, 2, 2)				
Language encoder	1-layer LSTM			NA		
Lang. enc. dimension	256			NA		
Word embed. dimension	32			NA		
Policy & value nets	MLP with 1 hidden layer with 512 units.					
Reconstruction decoders	Architectural transposes of the encoders, with independent weights.					
Recon. loss weight (HCAM)	1.				0.3	NA
Recon. loss weight TrXL	1.			NA	1.	NA
V-trace loss weight (HCAM)	0.1				1.	NA
V-trace loss weight (TrXL)	0.3	0.1		NA	1.	NA
V-trace baseline weight (HCAM)	1.				0.3	NA
V-trace baseline weight (TrXL)	1.	0.3	1.	NA	1.	NA
Entropy weight	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$				NA
Batch size	32					
Training unroll length	64	128	128	128	64	NA
Optimizer	Adam [26]					
LR	$2 \cdot 10^{-4}$				$4 \cdot 10^{-4}$	$1 \cdot 10^{-4}$

Table 2: Hyperparameter sweeps used in main experiments. The One-Shot StreetLearn and Paired Associative Inference tasks used different chunk size sweeps due to the different task lengths and demands. The Paired Associative Inference tasks do not use RL losses and so did not use the corresponding loss sweeps.

Reconstruct. loss weight	{0.1, 0.3, 1.}
V-trace loss weight	{0.1, 0.3, 1.}
V-trace baseline weight	{0.1, 0.3, 1.}
HCAM chunk size	{16, 32, 64}, except Streets=(2, 4, 8, 16) and Associative=None
LR	None, except Streets and Associative={1, 2, 4}$\cdot 10^{-4}$

In Table 1 we show the hyperparameters used for all experiments, and in Table 2 we show the hyperparameters sweeps used, although we generally used a subset of the full sweep. We swept each hyperparameter with a single seed per condition, and then reran the best parameter settings for each condition with more seeds to get a more robust estimate of the performance of each approach.

In most cases the hyperparameters that were not swept were taken from other sources without tuning for our architecture. In particular, the first tasks we considered were the rapid word learning tasks, and many parameters were taken directly from the hyperparameters of the original paper on which the tasks were based [19]. These hyperparameters were therefore tuned directly by prior researchers for other models, but we found them to work well for our memory as well. The visual encoder, language encoder, unsupervised reconstruction loss etc. were copied from those described in the prior work.

Since we ran all other experiments after the initial word learning experiments, we used many of these same hyperparameters on other experiments, such as the visual and language encoder architecture across all experiments. However, some hyperparameters do differ across tasks due to specific task features. For example, the visual encoder for the ballet tasks is set to have a filter size of 9 because this is the resolution of each square in the grid, and the entropy cost for the ballet tasks was chosen from our prior work [18] which used a similar grid world action space. These decisions were shared across all architectures, so should not favor our model over the baselines.

Self-supervised reconstruction loss We used the same reconstruction loss as Hill et al. [19], namely reconstructing the language with a softmax cross-entropy loss, and reconstructing the image pixels (normalized to range [0, 1] on each color channel) with a sigmoid cross-entropy loss. The image reconstruction loss was averaged across all pixels and channels, while the language reconstruction loss was summed across the sequence.

A.2.1 Bug fixed between original and revised versions of this paper

Shortly before the camera-ready deadline for NeurIPS, we discovered a bug in the configuration of the HCAM in the Ballet, Words, and Passive Visual Match domains: the local attention window was much longer than intended. Fixing this bug did not substantially alter results in the Ballet or Passive Visual Match tasks, but did change our results somewhat in the Rapid Word Learning tasks. The qualitative patterns of extrapolation and generalization to multiple episodes remain the same, but generalization of HCAM is somewhat worse, although still much better than the baseline models. This does not substantially affect the conclusions of the paper. We have revised the Rapid Word Learning plots in the main text to reflect these updated results, and included evaluation on the original levels in Fig. 7. However, note that our supplemental analyses in this domain were carried out with the longer attention window.

A.3 Plotting methods

In all plots, each curve is an average across multiple runs. The x-axis is always the number of agent steps (actions taken/frames seen) during training. The number of learner updates is generally $2000 - 4000\times$ smaller with a batch size of 32 trajectories per update, and unroll lengths of 64-128. The dark regions around the curve show $\pm$SD across runs, the light regions show the total range. The plots are smoothed by interpolation with a triangular window, with width and sampling frequency chosen to present results clearly depending on the speed and variability of learning the different tasks. All figures were made with seaborn [58] and matplotlib [22].

A.4 Compute resources

All experiments were run using Google TPU v2, v3, and v4 devices. Each run lasted between a few hours and a few days depending on the experiment. We ran actors/evaluators on CPUs. We estimate the total time needed to reproduce all experiments (including baselines and experiments in appendices) to be around 1000 TPU-hours + 300000 CPU hours.

B Tasks

We have uploaded selected video recordings of the HCAM-based agent performing our main tasks at `https://www.youtube.com/playlist?list=PLE5lx5-YU_Hr8Q9IgTAfisJ6XCy3Jhh6F`

B.1 Open source or released tasks

We have open-sourced our Ballet environment at: `https://github.com/deepmind/deepmind-research/tree/master/hierarchical_transformer_memory/hierarchical_attention`

We have released our rapid-word-learning tasks in the repository for the paper they were based upon `https://github.com/deepmind/dm_fast_mapping`

The environments from other papers that we used also have corresponding releases:

1. Passive Visual Match: `https://github.com/deepmind/deepmind-research/tree/master/tvt`
2. Paired Associative Inference: `https://github.com/deepmind/deepmind-research/tree/master/memo`
3. One-Shot StreetLearn `https://github.com/deepmind/deepmind-research/tree/master/rapid_task_solving`

B.2 Ballet

The tasks took place in a 9×9 tile room with an extra 1 tile wall surrounding on all sides, for a total of 11×11 tiles. This was upsampled at a resolution of 9 pixels per tile to form a 99×99 image as input to the agent. The agent was placed in the center of the room, and the dancers were placed randomly in 8 possible locations around it. The dancers always had distinct colors and shapes, selected from 15 shapes and 19 colors. These features merely served to distinguish the dancers. The agent always appeared as a white square. The agent received egocentric inputs (that is, its visual input was centered on its location), as this can improve generalization [18].

In Listing 1 we show the dance sequences used for the ballet tasks. All dances are 16 steps long. We trained all agents with levels uniformly sampled to have 16 or 48 steps of delay between dances, and 2, 4, or 8 dances. The number of dancers in the room corresponded to the number of dances, such that if there were only 2 dances, there were only 2 dancers, while if there were 8 dances there were 8 dancers. This is why chance-level performance is 50% with 2 dances, but 12.5% with 8. The agent was given a reward of 1 for a correct choice, and 0 for an incorrect choice.

B.3 Object permanence

The tasks took place within a 3D environment created with Unity. The agent received a visual observation of $96 \times 72 \times 3$ pixel RGB images, and a language observation that was tokenized at the word-level. The agent was initially placed in a fixed position near one wall of the room facing toward the center, and the boxes were randomly placed within the agent's field of view. When each object appeared, it jumped out of its box three times in succession. If there was a delay period, it began after the object returned to its box for the third time. The delay periods we used for the varying length training were 0, 10, 20, and 30 seconds. After the last presentation and delay phase, the lids of the boxes closed.

After the lids of the boxes closed, the agent was allowed to move and look around, and was given the instruction "look backward." The agent was rewarded 0.3 for looking backwards (far enough that the

```
{
"circle_cw": [0, 2, 4, 4, 6, 6, 0, 0, 2, 2, 4, 4, 6, 6, 0, 2],
"circle_ccw": [0, 6, 4, 4, 2, 2, 0, 0, 6, 6, 4, 4, 2, 2, 0, 6],
"up_down": [0, 4, 4, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 4, 4, 0],
"left_right": [2, 6, 6, 2, 2, 6, 6, 2, 2, 6, 6, 2, 2, 6, 6, 2],
"diagonal_uldr": [7, 3, 3, 7, 7, 3, 3, 7, 7, 3, 3, 7, 7, 3, 3, 7],
"diagonal_urdl": [1, 5, 5, 1, 1, 5, 5, 1, 1, 5, 5, 1, 1, 5, 5, 1],
"plus_cw": [0, 4, 2, 6, 4, 0, 6, 2, 0, 4, 2, 6, 4, 0, 6, 2],
"plus_ccw": [0, 4, 6, 2, 4, 0, 2, 6, 0, 4, 6, 2, 4, 0, 2, 6],
"times_cw": [1, 5, 3, 7, 5, 1, 7, 3, 1, 5, 3, 7, 5, 1, 7, 3],
"times_ccw": [7, 3, 5, 1, 3, 7, 1, 5, 7, 3, 5, 1, 3, 7, 1, 5],
"zee": [1, 6, 6, 2, 2, 5, 1, 5, 5, 2, 2, 6, 6, 1, 5, 1],
"chevron_down": [7, 4, 3, 1, 0, 5, 1, 5, 1, 4, 5, 7, 0, 3, 7, 3],
"chevron_up": [3, 0, 7, 5, 4, 1, 5, 1, 5, 0, 1, 3, 4, 7, 3, 7],
}
```

Listing 1: Dances used in the ballet task. 0-7 refer to directions of movement, clockwise from 0 = up.

boxes were out of view), and looking backward allowed it to advance to the choice phase of the task. In the choice phase, the agent was told "go to the box containing the [duck]" and was rewarded 1 for making the correct choice, and 0 for an incorrect choice.

B.4 Rapid word learning with distractors

We used the tasks created by Hill et al. [19] with the following modifications. First, we removed three of the possible objects (trains, robots, and rockets) to be used in the distractor task. We then added 0-20 distractor phases between the word binding and test phases. In each distractor phase, the agent and the three distractor objects were randomly placed in the room, and the agent was asked to lift one of them, e.g. "lift the rocket." The agent received a reward of 0.1 for successfully lifting the right object, and was allowed to progress to the next distractor task. If the agent lifted the wrong object, it was neither rewarded nor allowed to progress until it had lifted the correct object or until 20 seconds had passed. All agents rapidly learned to solve these distractor tasks. A fixed time limit of 450 seconds was used across all episodes, after which the episode terminated with reward 0 regardless of what phase the agent was in.

For the multi-episode evaluation tasks, we simply combined the number of episodes we wished to test across into a single "super-episode." For the final test phase, where we tested earlier words, the distractor objects were always taken from the same learning phase as the target object (to ensure that the agent remembered the exact name-object pairings, rather than simply which name appeared with which group of objects). We set a time limit of 450 seconds to complete *all* the sub-episodes. Agents with HCAM and TrXL memories were able to consistently complete the super-episodes within this time limit—even though TrXL could not choose the correct objects, it was consistently reaching the end and choosing *some* object for a chance at the final reward. However, the LSTM-based agents often timed out on these multi-episode evaluations.

B.5 Comparisons to other papers

The Passive Visual Match and Paired Associative Inference tasks were used unmodified. The StreetLearn [39] images and maps we used for the One-Shot StreetLearn were a more recent version than those used by Ritter et al. [48]. Because the task difficulty is fixed through the sampling of neighborhoods from the larger city graph, this should not substantially alter the difficulty of the tasks. We received permission from an author on each paper to use their tasks.

All three tasks from other papers have been released under Apache licenses, and the open source code can be found at:

- Passive visual match: `https://github.com/deepmind/deepmind-research/tree/master/tvt/dmlab`

- Paired Associative Inference: `https://github.com/deepmind/deepmind-research/tree/master/memo`
- One-Shot StreetLearn: `https://github.com/deepmind/deepmind-research/tree/master/rapid_task_solving`.

The PAI task In order to apply HCAM to the supervised PAI task, we took the following steps. We embedded all the input memories and probes using a single shared embedding layer. The structure of the memories for the PAI task matches the structure of HCAM's contents, where each pair of associated images corresponds to a single chunk in memory (of length 2). We therefore created a HCAM-style memory containing these embedded contents, and keyed by their summaries (averages across each chunk). We then provided the embedded query as input to the multi-layer HCAM model, but used the same set of embedded task memories at every layer. After 4 HCAM layers, we averaged-pooled across the sequence of resulting embeddings, and then performed a linear projection to produce a final output embedding. We then compared this ouptut embedding to the embeddings of the two possible choices using dot products. These dot products were used as logits in a softmax to choose the answer, and the model was trained using a cross-entropy loss. We did not use a self-supervised reconstruction loss for this setting.

C Detailed results

In Table 3 we show the mean performance and standard deviation across runs from our main experiments.

Table 3: Numerical results from main experiments/figures—mean $\pm$ standard deviation across 3 runs per condition. Results are average performance (% correct) across evaluations during the last 1% of training, except for the One-Shot StreetLearn tasks, where they are average reward during the last 1% of training. (Note that on some levels LSTMs were not consistently completing the task before the episode time limit. Incomplete episodes are scored as 0.)

Experiment	Level	Fig.	HCAM	TrXL	LSTM
Ballet	2 dances, delay 16	3a	99.8 ± 0.3	96.9 ± 1.4	97.4 ± 1.8
	8 dances, delay 16	3b	98.1 ± 3.3	65.7 ± 13.7	25.2 ± 2.7
	8 dances, delay 48	3c	97.2 ± 2.5	49.2 ± 13.0	29.7 ± 9.4
Objects	No delay, varying train	4b	96.7 ± 0.9	82.2 ± 28.7	33.2 ± 6.6
	Long delay, varying train	4c	91.7 ± 8.3	46.1 ± 20.0	34.8 ± 5.5
	Long delay, long-only train	4d	82.9 ± 16.4	31.1 ± 0.6	-
Words	10 distractors	5d	93.0 ± 4.0	76.7 ± 12.5	21.0 ± 18.3
	4 episodes, 0 distractor each	5e	82.8 ± 6.4	49.3 ± 16.5	19.8 ± 17.4
	2 episodes, 5 distractors each	5f	71.1 ± 8.0	48.7 ± 13.2	22.9 ± 20.1
Image		6a	97.0 ± 2.8	-	-
Associative		6b	97.5 ± 0.9	-	-
Streets		6c	26.8 ± 0.44	19.9 ± 0.65	-

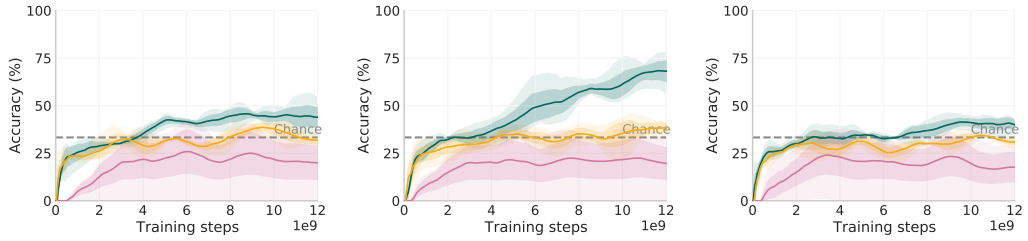
D Supplemental experiments

In this section we present some supplemental experiments and analyses. However, we make several notes here. First, these supplemental analyses were mostly run with a longer local attention window than used in the main text, see App. A.2.1, which could potentially affect results, particularly in the rapid word learning domain. Second, we use the original acronym HTM instead of HCAM in most of these plots, because we revised it only after a reviewer pointed out a name clash.

D.1 Fast-binding performance on harder hold-out tasks

In our original version of this paper, we presented generalization results on a harder set of evaluation tasks. Unfortunately, the high generalization performance on these results seemed to be at least in part due to a bug causing our HCAM memory to have a large local attention window (see above). We therefore changed the main text figures to show performance on slightly easier task variations.

However, in Fig. 7 we show performance on the original evaluation tasks. HCAM still achieves off-chance performance in 2 out of 3 cases, with fairly decent performance in one case, and performance continues to improve as training goes on.

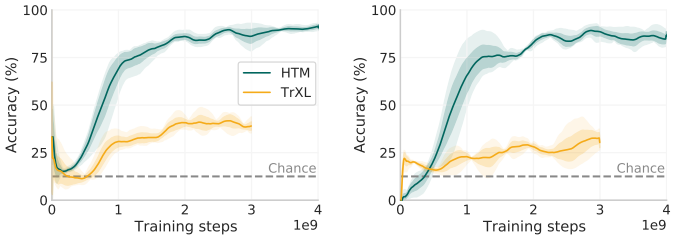


(a) Eval. 20 dist. (b) Eval. 4 eps., 1 dist. (c) Eval. 3 eps., 5 dist.

Figure 7: Evaluating HCAM on the harder generalization tasks we considered in the original version of this paper, after longer training (note horizontal axis). HCAM achieves off-chance performance, and continues to improve as training goes on. (3 seeds per condition.)

D.2 Ballet generalization

In the main text Ballet experiments (Fig. 3), we compared differences only in training performance. In Fig. 8, we show that HCAM is also able to generalize well from training on 2, 4, or 6 dances, to evaluation on 8 dances with either short or long delays.



(a) Evaluate 8 dances, short delays. (b) Evaluate 8 dances, long delays.

Figure 8: HCAM (labeled as HTM) trained on 2, 4, or 6 dance ballets generalizes well to 8 dance ballets, while TrXL does not. Results are from two seeds in each condition.

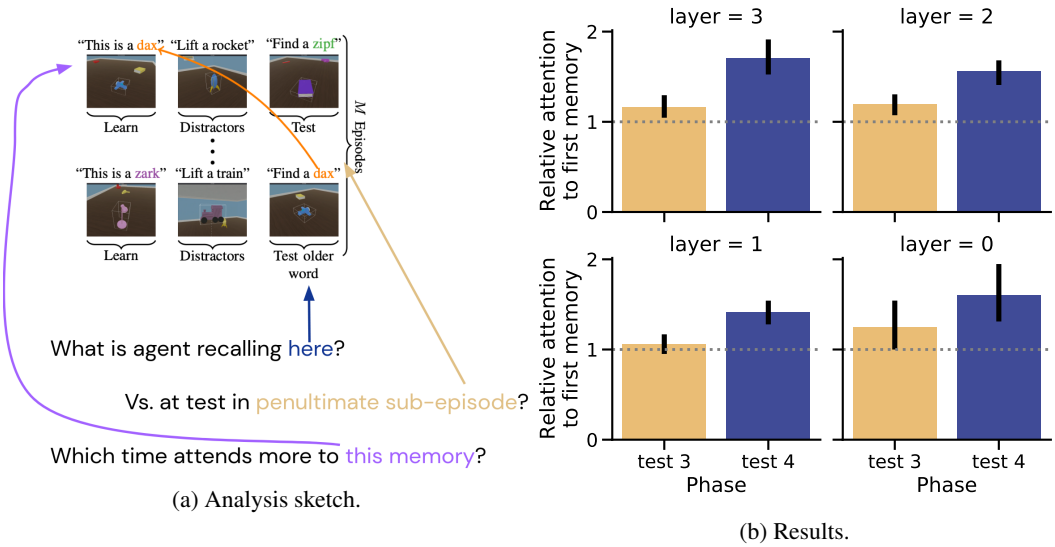
D.3 Analyzing memory attention in the rapid word learning tasks

In this section, we show that the HCAM agent's extrapolation to cross-episode evaluation (Fig. 5b) in the rapid word learning tasks is supported by selective patterns of memory access. In summary, we show that when the agent is asked to recall a word from many episodes before, all layers of its memory exhibit significantly higher attention to phase when it learned that word, compared to its attention patterns when tested on a more recent word.

Specifically, we considered the case of evaluation across 4 episodes, with 1 distractor each. To lay out the "super-episode" structure of this setting very explicitly, it proceeds through 4 episodes, each of which has three distinct phases. In other words, the episodes proceed as: learn 1, distract 1, test 1, learn 2, distract 2, test 2, learn 3, distract 3, test 3, learn 4, distract 4, test 4. Tests 1-3 evaluate memory for a word learned in their respective learn phases 1-3, but test 4 tests surprises the agent by asking about a word learned in learning phase 1 (Fig. 5b). As shown in the main text (Fig. 5e), the HCAM-based agent achieves above 90% performance on test 4, despite never being evaluated on words from earlier episodes during training.

To investigate the attention patterns underlying this result, we analyzed what chunks of memory the agent was attending to in test phase 4 (Fig. 9). In particular, we ran the agent on 100 of these episodes,

and saved its attention weights for each phase of the experiment. In 93 of these episodes, the agent chose correctly in test 4. Within those 93 episodes, we then evaluated the agent's attention to the first memory chunk of learn 1, the learning phase of the first episode (which generally contained most, if not all, of that learn phase). We compare the attention weight on this chunk when the agent is tested on one of these words in test phase 4 to a within-episode control: the relative weight when the agent is tested on a word from learn 3 during test phase 3. Is the agent attending more to its memory of learn phase 1 in test 4, when that memory is relevant, compared to test 3, when it is irrelevant? In fact, we find that across all four layers of the agent's memory, the agent is attending more strongly to the memory when it is relevant than when it is not. That is, the agent is distributing its attention intelligently, in a query-dependent way. It is attending most strongly to memories of the learn 1 phase when it is asked to recall a word from it, compared to a within-super-episode control where it is asked to recall a word from another phase (learn 3).



(a) Analysis sketch.

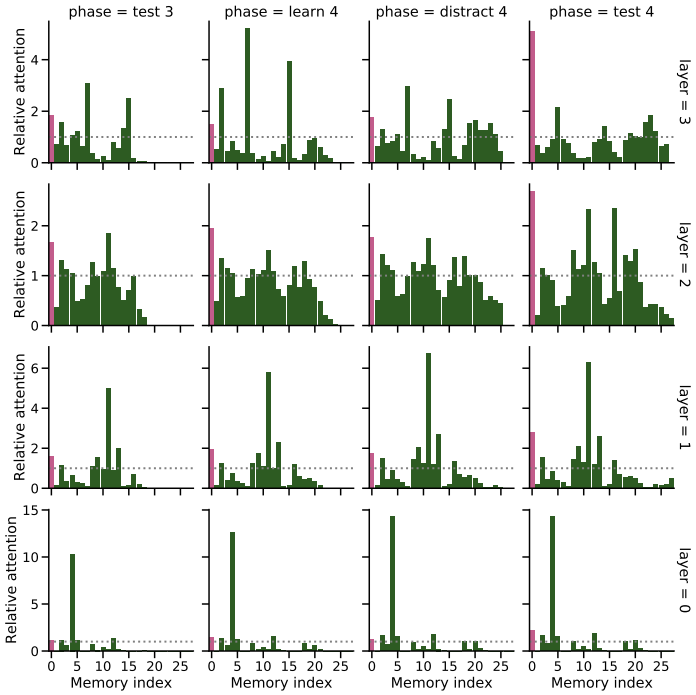
(b) Results.

Figure 9: The HCAM-based agent selectively attends to relevant memories in the rapid-word-learning generalization tasks. (a) We analyze the relative weight of attention to the first memory from the first learning phase, when the agent is asked to recall a word form the first phase in test 4 vs. when it is asked to recall a word from a later phase in test 3. (b) Across all 4 memory layers, the agent attends more strongly to its memory of this first learning phase when that memory is relevant—in test 4—compared to when that memory is irrelevant—in test 3. (This plot shows relative attention weights—that is, attention weights divided by average attention weight, so that if the agent were attending uniformly to all memories, their relative attention weights would be 1, indicated by the dotted line. This plot shows averages and 95%-CIs across the 93 episodes where the agent made a correct choice in test 4, out of 100 total super-episodes run.)

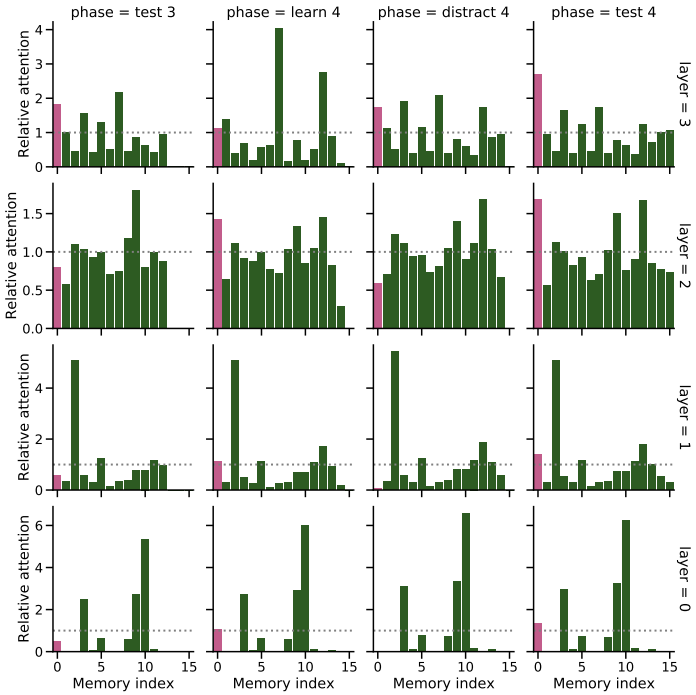
The results above involve several levels of aggregation: averaging within phases, and across many super-episodes. To give a flavor for the complexity of the full patterns of attention, in Fig. 10 we show the average attention weights for every layer across all the stored memories, in the final four phases of two super-episodes.

D.4 Varying chunk sizes

In Fig. 11 we show that the performance of the HCAM model is robust to varying chunk sizes in the ballet task; therefore its advantage in this task is not due to having additional information about the correct segmentation of the episodes. Furthermore, HCAM performs well even when its chunk size is 12, and so the total number of timepoints it can attend to at each layer is smaller than the number that the TrXL can attend to at each layer. Thus its advantage in these tasks is not due to attending to more of the episode, but rather to attending more effectively.

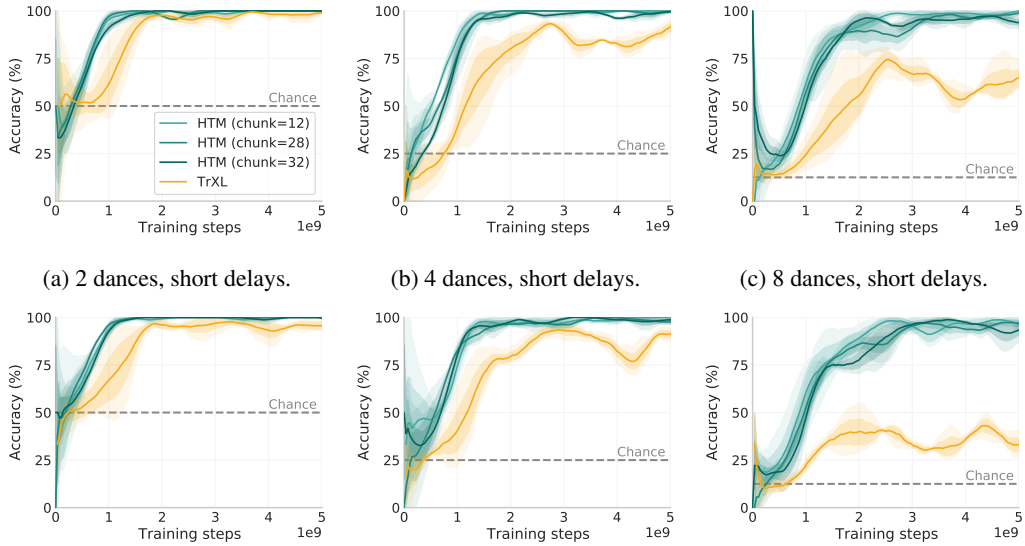


(a) Example super-episode 1



(b) Example super-episode 2

Figure 10: Patterns of attention within four phases of two example (randomly chosen) super-episodes on the rapid-word-learning generalization tasks. The higher layers of the network show clear shifts in attention patterns between the different phases, although with some consistent biases within each super-episode, especailly at the lower layers. The pink bar shows the weight on the first chunk from learn phase 1—the analyis shown in Fig. 9 corresponds to comparing the pink bars in the first and last column, aggregated across many more episodes. (This plot shows relative attention weights—that is, attention weights divided by average attention weight, so that chance level relative attention would be 1, indicated by the dotted line. Note that these were computed *before* the top-k operation on the attention weights, which is why more than 16 weights are active. The two super-episodes had different lengths, which is why more memories were stored in the first.)



(a) 2 dances, short delays.

(b) 4 dances, short delays.

(c) 8 dances, short delays.

(d) 2 dances, long delays.

(e) 4 dances, long delays.

(f) 8 dances, long delays.

Figure 11: Comparison of HCAM (labeled as HTM) with different chunk sizes to TrXL across the different ballet levels. The performance of the HCAM model is robust to varying chunk size, indicating that HCAM does not need a task-relevant segmentation to perform well. The results reported in the main text use chunk size 32; panels a, c, and f correspond to main text Fig. 3. These comparisons were run before a minor bug was fixed in HCAM memory writing. Results are from three seeds in each condition.

D.5 Varying k for memory selection

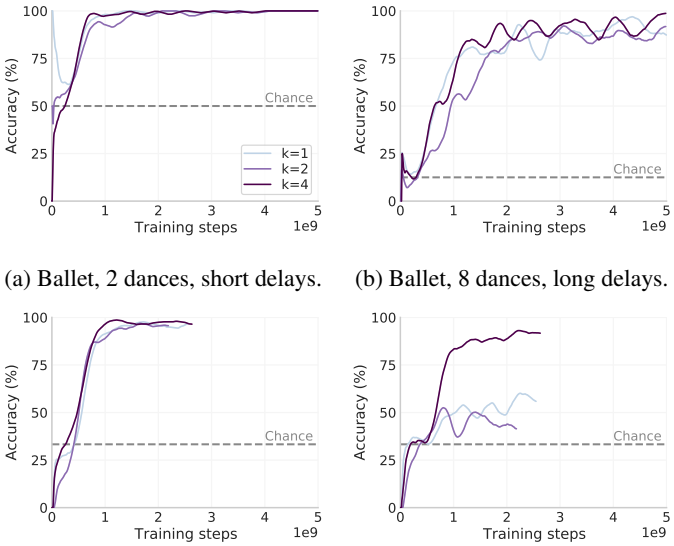
In Fig. 12 we show that HCAM is robust to varying the number k of memory chunks selected in the top-k step of the hierarchical attention at each layer. Specifically, while the main text experiments used $k = 16$, we show that HCAM is able to perform the ballet and object permanence tasks well even with $k = 4$, and can perform the shorter tasks even with $k = 2$ or even $k = 1$. While it initially surprised us that hard memory selection with $k = 1$ did not harm the optimization process, it resonates with recent results from the Switch Transformer [12], which found that hard selection of a single expert was effective in a mixture-of-experts style model.

D.6 The importance of self-supervised learning

In Fig. D.6, we show that the self-supervised loss (image + language reconstruction at the agent output) that we used as an auxiliary loss during training is necessary for our model to achieve good performance on the ballet and fast-binding tasks. This is presumably because this loss forces the model to encode the input in detail, and therefore that information is in-principle retrievable from the state representations stored in the agent memory.

D.7 Memory layer gating

In Fig. 14, we show that HCAM's performance is not enhanced by the gating mechanism proposed by Parisotto et al. [43], and in fact HCAM actually learns slightly more slowly when its layers are gated. This is potentially because HCAM already has some notion of gating in its selection of relevant chunks, and additional gating therefore only interferes with the optimization process. Thus, we did not use gating for HCAM in our main experiments. Furthermore, we found that gating was not necessary to train the TrXL memory on our tasks, although we used it in our main experiments to match Parisotto et al. [43]. However, HCAM with or without gating outperforms TrXL with or without gating.



(a) Ballet, 2 dances, short delays. (b) Ballet, 8 dances, long delays.

(c) Object permanence, no delays. (d) Object permanence, 30s delays.

Figure 12: Varying the number k of memory chunks selected in the top-k step of hierarchical attention. Performance is relatively robust to k smaller than used in the main experiments ($k = 16$), in fact shorter tasks can be learned even with $k = 1$, while longer tasks require $k \geq 4$. (a-b) On the ballet tasks, HCAM can learn fairly well even with $k = 1$, both for the shorter and longer tasks. (c-d) On the object permanence tasks, HCAM can learn the shortest tasks well even with $k = 1$, but struggles to learn the longer tasks unless $k \geq 4$. (One seed per condition.)

D.8 Comparing a TrXL that is 2× wider/deeper than HCAM

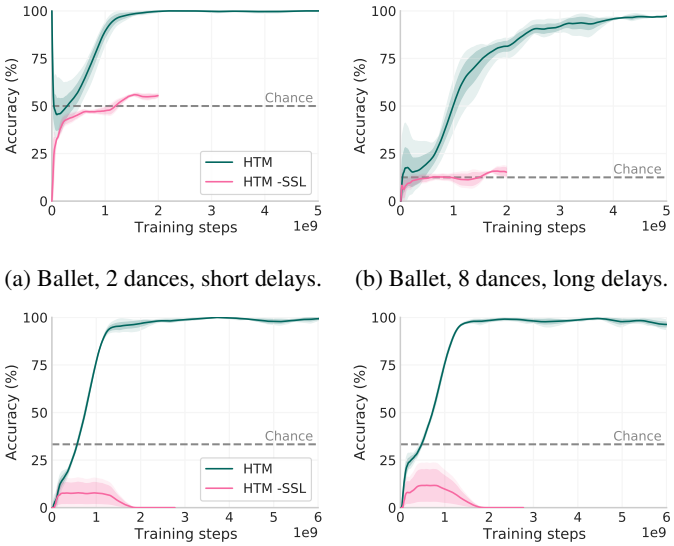
Because our HCAM-based agents have an added HCAM attention block in each memory layer compared to our TrXL-based ones, it might seem that they have somewhat more parameters and greater total depth. However, as noted in Section D.7, HCAM does not use the gating layers used by the TrXL memory [43] and because of this HCAM uses about 20% fewer parameters and is in some sense shallower than our TrXL baselines. However, it does have more layers of attention. To ensure that this or other simple factors were not the primary driver of HCAM's advantage, we ran comparisons where we either made the TrXL twice as wide (i.e. each layer had twice as many hidden units, including the attention projections etc.) or twice as deep (i.e. an 8-layer TrXL memory instead of 4-layers as we used for our main experiments). Both of these have substantially more parameters than our HCAM-based models, and the latter is substantially deeper as well. However, we show in Fig. 15 that these much larger TrXL models were also unable to match the performance of HCAM on the rapid word-learning tasks. Thus the advantage of HCAM is not due to parameters or depth alone.

D.9 Sparsity without hierarchy: a top-k TrXL

One possible explanation of our results would be that sparsity alone is sufficient—perhaps the TrXL is suffering from spreading its attention across too many points in the past, but if it were restricted to only a few points hierarchy would not be necessary. To evaluate this possibility, we created a modified TrXL where we imposed sparsity of attention, by truncating its attention to only the top-k most relevant timepoints. We chose $k = 16$ to match HCAM. We show the results in Fig. D.9. The top-k TrXL performs comparably to a standard TrXL on the ballet tasks (i.e. does not perform as well as HCAM), and fails to learn properly on the rapid word-learning tasks, even if allowed to attend to a larger number of points ($k = 32$). Thus, sparsity without hierarchy does not suffice, and may actually harm learning.

D.10 Compute efficiency assessed by learner FPS

One goal of HCAM is that sparser attention might be more efficient than full attention. This is especially true when comparing HCAM without gating to the more computationally intense Gated



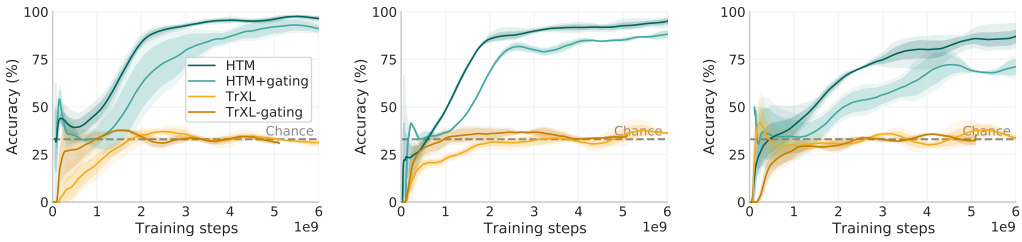
(a) Ballet, 2 dances, short delays.

(b) Ballet, 8 dances, long delays.

(c) Rapid word learning, no distractors.

(d) Rapid word learning, 2 distractors.

Figure 13: The self-supervised loss (SSL) is necessary for the model to learn appropriate representations. When the SSL is disabled, the model either fails to achieve substantially-above-chance performance, for example in the ballet tasks (a-b), or fails to learn the tasks to even chance level, as in the rapid word learning tasks (c-d). (HTM refers to HCAM, see note above. 3 runs per condition for main results, 2 runs per condition for results without SSL.)



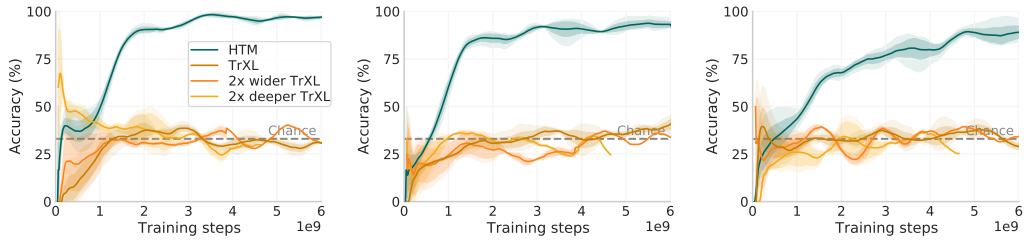
(a) Train 0-2 distractors, evaluate 20.

(b) Evaluate 4 episodes, 1 distractor each.

(c) Evaluate 3 episodes, 5 distractors each.

Figure 14: HCAM (labeled as HTM) performs better without gating [43] than with gating. On the fast-binding tasks HCAM with gating learns slightly more slowly and generalizes slightly worse than without gating. Gating of memory layers does not appear necessary for TrXL in our tasks, unlike the experiments of Parisotto et al. [43]. However, neither gated nor ungated TrXL are able to extrapolate to the tasks that gated or ungated HCAM does. (3 seeds per condition for main runs, 2 per condition for alternatives.)

TrXL [43]. Correspondingly, we show in Fig. 17 that HCAM generally runs ∼30-40% faster than TrXL.

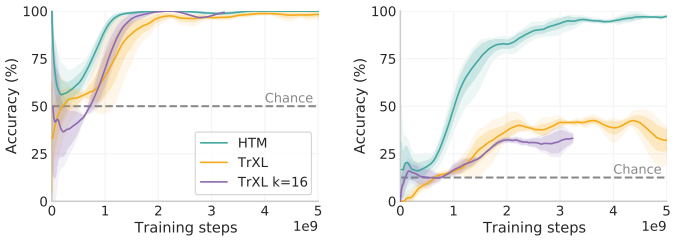


(a) Train 0-2 distractors, evaluate 20.

(b) Evaluate 4 episodes, 1 distractor each.

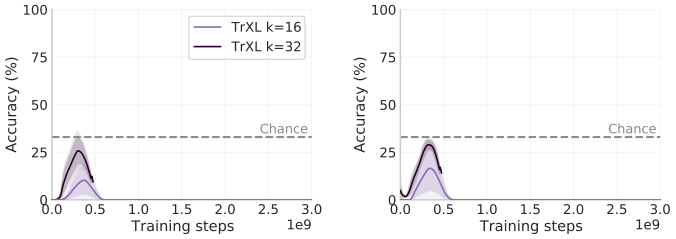
(c) Evaluate 3 episodes, 5 distractors each.

Figure 15: Comparing agents with TrXL memories that have more parameters than HCAM on the rapid word-learning tasks. Neither a TrXL model that is twice as wide, nor one that is twice as deep are able to perform as well as HCAM. Thus, HCAM's advantage is not due to the added blocks or slightly more parameters than our TrXL baseline. (HTM refers to HCAM, see note above. 3 seeds per main condition, 2 seeds per condition for supplemental.)



(a) Ballet, 2 dances, short delays.

(b) Ballet, 8 dances, long delays.



(c) Word learning, train: no distractors.

(d) Word learning, train: 2 distractors.

Figure 16: Sparsity alone is not sufficient—a TrXL restricted to attend to only the top-k timepoints performs comparably to a standard TrXL at the ballet tasks (a-b), but collapses and fails to learn in the rapid word learning tasks (c-d), even if given a larger k. The advantage of HCAM is not due to sparsity alone. (HTM refers to HCAM, see note above. 3 seeds per main condition, 2 seeds per condition for supplemental.)

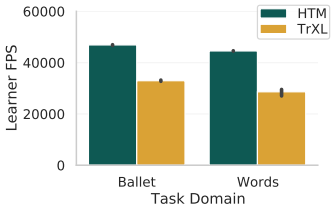
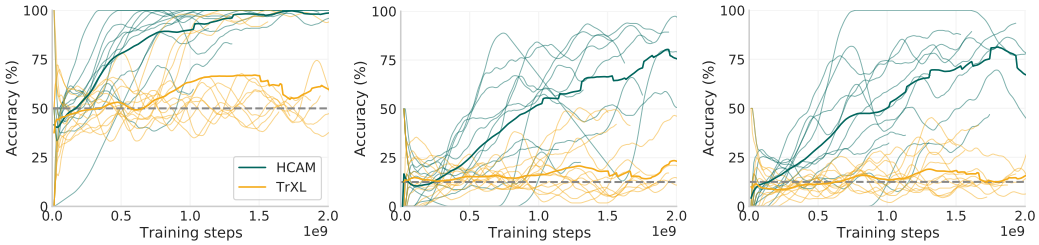


Figure 17: HCAM (labeled as HTM) runs at a higher speed (measured in average learner frames processed per second) than TrXL. Results on TPUv3, error bars show 95%-CI across 3 runs.

D.11 Results are robust to reviewer-suggested hyperparameter sweeps

Our reviewers evaluated the paper carefully, and expressed concerns that there might be bias in our hyperparameter selection. In particular, one reviewer raised a concern that TrXL might learn tasks like Ballet if given a different learning rate. To address these concerns, we ran a set of follow-up hyperparameter sweeps. We swept the learning rate and entropy weight (which we had not varied previously from the values used in prior work) on both the Ballet and Word tasks. We ran a full product of three learning rates above/below our original values (5e-4, 5e-5, 1e-5) and entropy weights $5\times$ more or less than the original value, with 2 seeds per condition (for a total of 24 hyper $\times$ seed $\times$ memory type combinations per task domain). We emphasize that the same hyperparameters were tested for both models, and that these sweeps centered on hyperparameter settings that were previously untuned (sourced from prior papers), and that this sweep focuses on learning rate, which a reviewer suggested might particularly benefit TrXL. Our results show that HCAM is more robust to variation in these hyperparameters than TrXL, and generally sweeping these parameters does not improve TrXL's performance substantially beyond the results reported in the main text.

Ballet results: HCAM substantially outperforms TrXL in this sweep as well. First, HCAM is far more robust to varying the hyperparameters—in every hyperparameter setting, agents with HCAM achieved off chance performance (measured as window-averaged performance >5 percentage points above chance-level) on the hard 8-dance tasks within 1 billion steps. By contrast, 58% of the TrXL jobs did not attain off chance performance on even the easiest task within 1.5 billion steps (when we stopped the training). In addition, 66% of the HCAM jobs achieved above-75% performance on the easiest tasks before *any* of the TrXL jobs achieved above-chance performance on any task. The performance of the best TrXL jobs from this sweep is comparable to the performance at the same point in training from our original experiments: about 40-50% performance on the 8 dance, short delays task, and 25-35% on the 8 dance, long delays task at 1.5 billion steps. HCAM performed much better than TrXL, with 75% of the HCAM jobs outperforming even the best TrXL agents on the hardest task, and the best HCAM jobs comparable to the results in the paper, achieving 80-90% performance on the hardest tasks at 1.5 billion steps. In both 8 dance levels, the advantage of the two HCAM seeds with the best hyperparams over the two TrXL seeds with the best hyperparams is significant by a paired[1] t-test, respectively $t(1) = 21, p = 0.03$ and $t(1) = 101, p = 0.006$. In summary, TrXL's performance at these tasks does not seem to be improved by varying learning rates or entropy weight, and HCAM seems much more robust to variation in these hyperparameters.



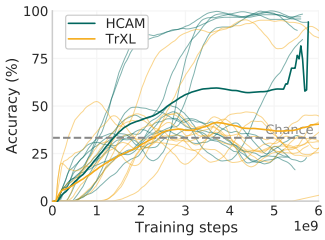
(a) Ballet, 2 dances, short delays. (b) Ballet, 8 dances, short delays. (c) Ballet, 8 dances, long delays.

Figure 18: Sweeping hyperparameters (learning rate and entropy weight) in the Ballet tasks: HCAM is substantially more robust to variation in these hyperparameters, and the conclusions of our main text experiments are unaltered. The thick line plots the mean, while the thin lines plot individual sweep values. (The wide variability in early accuracy values should be disregarded—it is due to smoothing artifacts due to sparse data in this region as evaluation jobs are starting.)

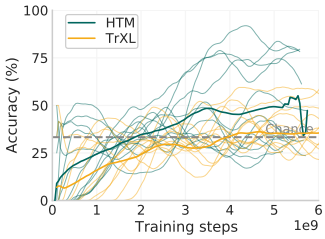
Rapid word learning results: The results are similar to the above. First, HCAM is more robust to varying hyperparameters: In these more challenging tasks, only 17% of the TrXL jobs achieve high training performance within 5 billion steps, while 50% of the HCAM jobs achieve high performance on the training tasks. HCAM also generalizes better than TrXL. However, unlike our original experiments, one set of these TrXL jobs does achieve somewhat above-chance performance at one of

[1] paired reflecting the non-independence of the encoder initialization when the agents are initialized with the same random seed, but results are similar with an unpaired test, respectively $t(2) = 29, p = 0.001$; $t(2) = 9.5$, $p = 0.01$)

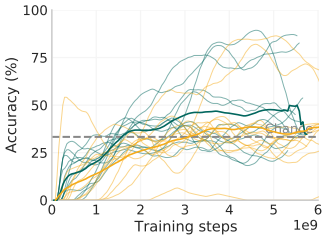
the the evaluation tasks we considered. We performed a replication with three new random seeds in the best hyperparameters from this sweep for each memory (as in the main results), and in this replication the TrXL did not achieve significantly off chance performance. However, HCAM did achieve significantly off-chance performance, (though not as high as the main text results using the hyperpareters tuned in our original sweeps). Thus, HCAM again appears to be both more robust across hyperparameters, and better when comparing best-hyperparameter configurations.



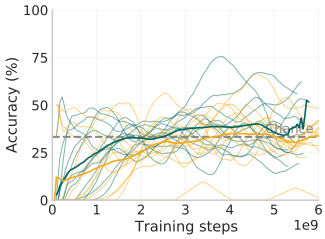
(a) Sweep: Train 0-2 distractors, evaluate 2 distractors.



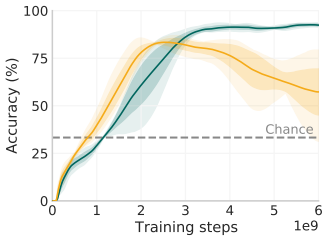
(b) Sweep: Train 0-2 distractors, evaluate 20 distractors.



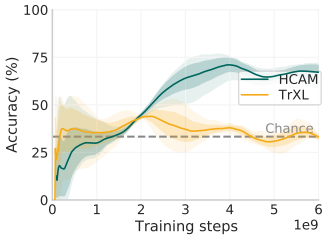
(c) Sweep: Train 0-2 distractors, evaluate 4 eps, 1 dist. each.



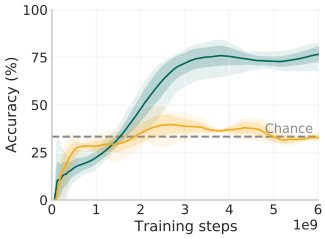
(d) Sweep: Train 0-2 distractors, evaluate 3 eps, 5 dist. each.



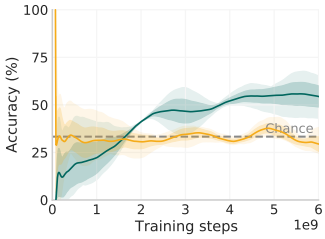
(e) Replication: Evaluate 2 distractors.



(f) Replication: Evaluate 20 distractors.



(g) Replication: Evaluate 4 eps, 1 dist. each.



(h) Replication: Evaluate 3 eps, 5 dist. each.

Figure 19: Sweeping hyperparameters (learning rate and entropy weight) in the Words tasks: HCAM is again more robust to variation in these hyperparameters. (a-d) The sweep results. The thick line plots the mean, while the thin lines plot individual sweep values. While HCAM is much more robust overall, as shown in the number of hyperparameter settings that learn the train tasks (a), a few TrXL jobs show above chance generalization on some tasks in the sweep. (e-h) To follow-up on the above result, we ran a replication of the best hyperparameters from the sweep with three new random seeds (as we did for all our main text results). This replication does not show substantially above-chance generalization performance from TrXL and shows some collapse in performance on the train tasks. HCAM's performance remains substantially above chance in a replication of the best values in the sweep, although note that the best results from the sweep are worse than the results with the tuned hyperparameters used in the original experiments. (Note also that these experiments were run with a longer attention window for HCAM than intended, see App. A.2.1.)