

A Appendix – Glossary

In this section, we provide a comprehensive glossary of key terms and concepts used throughout this paper. The definitions are intended to clarify the terminology proposed in our research and to ensure that readers have a clear understanding of the main elements underpinning our work.

1. \mathcal{M} – MDP environment
2. \mathcal{M}_P – POMDP environment
3. $\tilde{\mathcal{M}}_P$ – memory-intensive environment
4. $h_{0:t-1} = \{(o_i, a_i, r_i)\}_{i=0}^{t-1}$ – agent history of interactions with environment
5. K – agent base model context length
6. \bar{K} – context memory border of the agent, such that $K \in [1, \bar{K}] \Leftrightarrow$ strictly LTM problem
7. $\mu(K)$ – memory mechanism that increases number of steps available to the agent to process
8. $K_{eff} = \mu(K)$ – the agent effective context after applying the memory mechanism
9. $\alpha_{t_e}^{\Delta t} = \{(o_i, a_i, r_i)\}_{i=t_e}^{t_e+\Delta t}$ – an event starting at time t_e and lasting Δt , which the agent should recall when making a decision in the future
10. $\beta_{t_r} = \beta_{t_r}(\alpha_{t_e}^{\Delta t}) = a_t \mid (o_t, \alpha_{t_e}^{\Delta t})$ – the moment of decision making at time t_r according to the event $\alpha_{t_e}^{\Delta t}$
11. $\xi = t_r - t_a - \Delta t + 1$ – an event’s correlation horizon

B Appendix – Additional notes on the motivation for the article

B.1 Why use definitions from neuroscience?

Definitions from neuroscience and cognitive science, such as short-term and long-term memory, as well as declarative and procedural memory, are already well-established in the RL community, but do not have common meanings and are interpreted in different ways. We strictly formalize these definitions to avoid possible confusion that may arise when introducing new concepts and redefine them with clear, quantitative meanings to specify the type of agent memory, since the performance of many algorithms depends on their type of memory.

In focusing exclusively on memory within RL, we do not attempt to exhaustively replicate the full spectrum of human memory. Instead, our goal is to leverage the intuitive understanding of neuroscience concepts already familiar to RL researchers. This approach avoids the unnecessary introduction of new terminology into the already complex Memory RL domain. By refining and aligning existing definitions, we create a robust framework that facilitates clear communication, rigorous evaluation, and practical application in RL research.

B.2 On practical applications of our framework

The primary goal of our framework is to address practical challenges in RL by providing a robust classification of memory types based on temporal dependencies and the nature of memorized information. This classification is essential for standardizing memory testing and ensuring that RL agents are evaluated under conditions that accurately reflect their capabilities.

In RL, memory is interpreted in various ways, such as transformers with large context windows, recurrent networks, or models capable of skill transfer across tasks. However, these approaches often vary fundamentally in design, making comparisons unreliable and leading to inconsistencies in testing. Our framework resolves this by providing a clear structure to evaluate memory mechanisms under uniform and practical conditions.

The proposed definitions of declarative and procedural memory use two straightforward numerical parameters: the number of environments (n_{envs}) and episodes (n_{eps}). These parameters allow researchers to reliably determine the type of memory required for a task. This simplicity and alignment with numerical parameters make the framework practical and widely applicable across diverse RL problems.

Moreover, the division of declarative memory into long-term and short-term memory, as well as the need to use a balance between the agent’s context length K and the correlation horizons of the environment ξ when conducting the experiment, allows us to unambiguously determine which type of memory is present in the agent. This clarity ensures fair comparisons between agents with similar memory mechanisms and highlights specific limitations in an agent’s design. By aligning memory definitions with practical testing requirements, the framework provides actionable insights to guide the development of memory-enhanced RL agents.

C Appendix – Memory Mechanisms

In RL, memory has several meanings, each of which is related to a specific class of different tasks. To solve these tasks, the authors use various memory mechanisms. The most prevalent approach to incorporating memory into an agent is through the use of Recurrent Neural Networks (RNNs) (Rumelhart, Hinton, and Williams 1986), which are capable of handling sequential dependencies by maintaining a hidden state that captures information about previous time steps (Wierstra et al. 2010; Hausknecht and Stone 2015; Sorokin et al. 2015; Duan et al. 2016; Song et al. 2018; Zintgraf et al. 2020). Another popular way to implement memory is to use Transformers (Vaswani et al. 2017), which use self-attention mechanisms to capture dependencies inside the context window (Parisotto et al. 2020; Lampinen et al. 2021; Esslinger, Platt, and Amato 2022; Melo 2022; Team et al. 2023; Pramanik et al. 2023; Robine et al. 2023; Ni et al. 2023; Grigsby, Fan, and Zhu 2024; Shala, Biedenkapp, and Grabocka 2024). State-space models (SSMs) (Gu, Goel, and Ré 2021; Smith, Warrington, and Linderman 2023; Gu and Dao 2023) combine the strengths of RNNs and Transformers and can also serve to implement memory through preservation of system state (Hafner et al. 2019; Lu et al. 2023; Becker, Freymuth, and Neumann 2024; Samsami et al. 2024). Temporal convolutions may be regarded as an effective memory mechanism, whereby information is stored implicitly through the application of learnable filters across the time axis (YuXuan Liu and Hsieh 2016; Mishra et al. 2018). A world model (Ha and Schmidhuber 2018) which builds an internal environment representation can also be considered as a form of memory. One method for organizing this internal representation is through the use of a graph, where nodes represent observations within the environment and edges represent actions (Morad et al. 2021; Zhu, Li, and Elhoseiny 2023; Kang et al. 2024).

A distinct natural realization of memory is the utilization of an external memory buffer, which enables the agent to retrieve pertinent information. This approach can be classified into two categories: read-only (writeless) (Oh et al. 2016; Lampinen et al. 2021; Goyal et al. 2022; Cherepanov et al. 2024) and read/write access (Graves et al. 2016; Zaremba and Sutskever 2016; Parisotto and Salakhutdinov 2017).

Memory can also be implemented without architectural mechanisms, relying instead on agent policy. For instance, in the work of Deverett et al. (2019), the agent learns to encode temporal intervals by generating specific action patterns. This approach allows the agent to implicitly represent timing information within its behavior, showcasing that memory can emerge as a result of policy adaptations rather than being explicitly embedded in the underlying neural architecture.

Using these memory mechanisms, both decision-making tasks based on information from the past within a single episode and tasks of fast adaptation to new tasks are solved. However, even in works using the same underlying base architectures to solve the same class of problems, the concepts of memory may differ.

D Appendix – POMDP

D.1 POMDP

The Partially Observable Markov Decision Process (POMDP) is a generalization of the Markov Decision Process (MDP) that models sequential decision-making problems where the agent has incomplete information about the environment’s state. POMDP can be represented as a tuple $\mathcal{M}_P = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z} \rangle$, where \mathcal{S} denotes the set of states, \mathcal{A} is the set of actions, \mathcal{O} is the set of observations and $\mathcal{Z} = \mathcal{P}(o_{t+1} | s_{t+1}, a_t)$ is an observation function such that $o_{t+1} \sim \mathcal{Z}(s_{t+1}, a_t)$. An agent takes an action $a_t \in \mathcal{A}$ based on the observed history $h_{0:t-1} = \{(o_i, a_i, r_i)\}_{i=0}^{t-1}$ and receives a reward $r_t = \mathcal{R}(s_t, a_t)$. It is important to note that state s_t is not available to the agent at time t . In the case of POMDPs, a policy is a function $\pi(a_t | o_t, h_{0:t-1})$ that uses the agent history $h_{0:t-1}$ to obtain the probability of the action a_t . Thus, in order to operate effectively in a POMDPs, an agent must have memory mechanisms to retrieve a history $h_{0:t-1}$. Partial observability arises in a variety of real-world situations, including robotic navigation and manipulation tasks, autonomous vehicle tasks, and complex decision-making problems.

E Appendix – Meta Reinforcement Learning

In this section, we explore the concept of Meta-Reinforcement Learning (Meta-RL), a specialized domain within POMDPs that focuses on equipping agents with the ability to learn from their past experiences across multiple tasks. This capability is particularly crucial in dynamic environments where agents must adapt quickly to new challenges. By recognizing and memorizing common patterns and structures from previous interactions, agents can enhance their efficiency and effectiveness when facing unseen tasks.

Meta-RL is characterized by the principle of “*learning to learn*”, where agents are trained not only to excel at specific tasks but also to generalize their knowledge and rapidly adjust to new tasks with minimal additional training. This adaptability is achieved through a structured approach that involves mapping data collected from various tasks to policies that guide the agent’s behavior.

Meta-RL algorithm is a function f_θ parameterized with *meta-parameters* that maps the data \mathcal{D} , obtained during the process of training of RL agent in MDPs (tasks) $\mathcal{M}_i \sim p(\mathcal{M})$, to a policy $\pi_\phi : \phi = f_\theta(\mathcal{D})$. The process of learning the function f is typically referred to as the *outer-loop*, while the resulting function f is called the *inner-loop*. In this context, the parameters θ are associated with the outer-loop, while the parameters ϕ are associated with the inner-loop. Meta-training proceeds by sampling a task from the task distribution, running the inner-loop on it, and optimizing the inner-loop to improve the policies it produces. The interaction of the inner-loop with the task, during which the adaptation happens, is called a *lifetime* or a *trial*. In Meta-RL, it is common for \mathcal{S} and \mathcal{A} to be shared between all of the tasks and the tasks to only differ in the reward $\mathcal{R}(s, a)$ function, the dynamics $\mathcal{P}(s' | s, a)$, and initial state distributions $P_0(s_0)$ (Beck et al. 2024).

F Appendix – Experiment Details

F.1 Appendix – Environments description

This section provides an extended description of the environments used in this work.

Passive-T-Maze (Ni et al. 2023). In this T-shaped maze environment, the agent’s goal is to move from the starting point to the junction and make the correct turn based on an initial signal. The agent can select from four possible actions: $a \in \{left, up, right, down\}$. The signal, denoted by the variable $clue$, is provided only at the beginning of the trajectory and indicates whether the agent should turn up ($clue = 1$) or down ($clue = -1$). The episode duration is constrained to $T = L + 1$, where L is the length of the corridor leading to the junction, which adds complexity to the task. To facilitate navigation, a binary variable called $flag$ is included in the observation vector. This variable equals 1 one step before reaching the junction and 0 at all other times, indicating the agent’s proximity to the junction. Additionally, a noise channel introduces random integer values from the set $\{-1, 0, +1\}$ into the observation vector, further complicating the task. The observation vector is defined as $o = [y, clue, flag, noise]$, where y represents the vertical coordinate.

The agent receives a reward only at the end of the episode, which depends on whether it makes a correct turn at the junction. A correct turn yields a reward of 1, while an incorrect turn results in a reward of 0. This configuration differs from the conventional Passive T-Maze environment (Ni et al. 2023) by featuring distinct observations and reward structures, thereby presenting a more intricate set of conditions for the agent to navigate and learn within a defined time constraint. To transition from a sparse reward function to a dense reward function, the environment is parameterized using a penalty defined as $penalty = -\frac{1}{T-1}$, which imposes a penalty on the agent for each step taken within the environment. Thus, this environment has a 1D vector space of observations, a discrete action space, and sparse and dense configurations of the reward function.

Minigrid-Memory (Chevalier-Boisvert et al. 2023). Minigrid-Memory is a two-dimensional grid-based environment specifically crafted to evaluate an agent’s long-term memory and credit assignment capabilities. The layout consists of a T-shaped maze featuring a small room at the corridor’s outset, which contains an object. The agent is instantiated at a random position within the corridor. Its objective is to navigate to the chamber, observe and memorize the object, then proceed to the junction at the maze’s terminus and turn towards the direction where the object, identical to that in the initial chamber, is situated. A reward function defined as $r = 1 - 0.9 \times \frac{t}{T}$ is awarded upon successful completion, while failure results in a reward of zero. The episode concludes when the agent either makes a turn at a junction or exhausts a predefined time limit of 95 steps. To implement partial observability, observational constraints are imposed on the agent, limiting its view to a 3×3 frame size. Thus, this environment has a 2D space of image observations, a discrete action space, and sparse reward function.

F.2 Experimental Protocol

For each experiment, we conducted three runs of the agents with different initializations and performed validation during training using 100 random seeds ranging from 0 to 99. The results are presented as the mean success rate (or reward) \pm the standard error of the mean (SEM).

Table 1: Online RL baselines hyperparameters used in the Minigrid-Memory and Passive T-Maze experiments.

Table 2: SAC-GPT-2

Hyperparameter	Value
Number of layers	2
Number of attention heads	2
Hidden dimension	256
Batch size	64
Optimizer	Adam
Learning rate	3e-4
Dropout	0.1
Replay buffer size	1e6
Discount (γ)	0.99
Entropy temperature	0.1

Table 3: DQN-GPT-2

Hyperparameter	Value
Number of layers	2
Number of attention heads	2
Hidden dimension	256
Batch size	64
Optimizer	Adam
Learning rate	3e-4
Dropout	0.1
Replay buffer size	1e6
Discount (γ)	0.99

Table 4: DTQN

Hyperparameter	Value
Number of layers	4
Number of attention heads	8
Hidden dimension	128
Batch size	32
Optimizer	Adam
Learning rate	3e-4
Dropout	0.1
Replay buffer size	5e5
Discount (γ)	0.99

References

Beck, J.; Vuorio, R.; Liu, E. Z.; Xiong, Z.; Zintgraf, L.; Finn, C.; and Whiteson, S. 2024. A Survey of Meta-Reinforcement Learning. arXiv:2301.08028.

Table 5: Offline RL baselines hyperparameters used for Decision Transformer and BC-LSTM in T-Maze experiments.

Table 6: Decision Transformer (DT)

Hyperparameter	Value
Number of layers	8
Number of attention heads	4
Hidden dimension (d_{model})	128
Feedforward dimension (d_{inner})	128
Head dimension (d_{head})	128
Context length (K)	$3T$
Dropout	0.0
DropAttention	0.0
Optimizer	AdamW
Learning rate	$1e-4$
Weight decay	0.1
Adam betas	(0.9, 0.999)
Batch size	64
Warmup steps	1000
Epochs	200

Table 7: BC-LSTM

Hyperparameter	Value
Number of layers	1
Hidden dimension (d_{model})	64
Bidirectional	False
Effective Context length (K_{eff})	$3T$
Dropout	0.0
Optimizer	AdamW
Learning rate	$3e-4$
Weight decay	0.01
Adam betas	(0.9, 0.999)
Batch size	64
Warmup steps	100
Epochs	100

Becker, P.; Freymuth, N.; and Neumann, G. 2024. KalMamba: Towards Efficient Probabilistic State Space Models for RL under Uncertainty. *arXiv:2406.15131*.

Cherepanov, E.; Staroverov, A.; Yudin, D.; Kovalev, A. K.; and Panov, A. I. 2024. Recurrent Action Transformer with Memory. *arXiv preprint arXiv:2306.09459*.

Chevalier-Boisvert, M.; Dai, B.; Towers, M.; de Lazcano, R.; Willems, L.; Lahlou, S.; Pal, S.; Castro, P. S.; and Terry, J. 2023. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. *CoRR*, abs/2306.13831.

Deverett, B.; Faulkner, R.; Fortunato, M.; Wayne, G.; and Leibo, J. Z. 2019. Interval timing in deep reinforcement learning agents. *Advances in Neural Information Processing Systems*, 32.

Duan, Y.; Schulman, J.; Chen, X.; Bartlett, P. L.; Sutskever, I.; and Abbeel, P. 2016. RL²: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv:1611.02779*.

Esslinger, K.; Platt, R.; and Amato, C. 2022. Deep Transformer Q-Networks for Partially Observable Reinforcement Learning. *arXiv preprint arXiv:2206.01078*.

Goyal, A.; Friesen, A. L.; Banino, A.; Weber, T.; Ke, N. R.; Badia, A. P.; Guez, A.; Mirza, M.; Humphreys, P. C.; Konyushkova, K.; Sifre, L.; Valko, M.; Osindero, S.; Lillicrap, T.; Heess, N.; and Blundell, C. 2022. Retrieval-Augmented Reinforcement Learning. *arXiv:2202.08417*.

Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwińska, A.; Gómez, S.; Grefenstette, E.; Ramalho, T.; Agapiou, J.; Badia, A.; Hermann, K.; Zwols, Y.; Ostrovski, G.; Cain, A.; King, H.; Summerfield, C.; Blunsom, P.; Kavukcuoglu, K.; and Hassabis, D. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538.

Grigsby, J.; Fan, L.; and Zhu, Y. 2024. AMAGO: Scalable In-Context Reinforcement Learning for Adaptive Agents. *arXiv:2310.09971*.

Gu, A.; and Dao, T. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Gu, A.; Goel, K.; and Ré, C. 2021. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*.

Ha, D.; and Schmidhuber, J. 2018. Recurrent World Models Facilitate Policy Evolution. *arXiv:1809.01999*.

Hafner, D.; Lillicrap, T.; Fischer, I.; Villegas, R.; Ha, D.; Lee, H.; and Davidson, J. 2019. Learning Latent Dynamics for Planning from Pixels. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 2555–2565. PMLR.

Hausknecht, M.; and Stone, P. 2015. Deep recurrent q-learning for partially observable mdps.

Kang, Y.; Zhao, E.; Zang, Y.; Li, L.; Li, K.; Tao, P.; and Xing, J. 2024. Sample Efficient Reinforcement Learning Using Graph-Based Memory Reconstruction. *IEEE Transactions on Artificial Intelligence*, 5(2): 751–762.

Lampinen, A.; Chan, S.; Banino, A.; and Hill, F. 2021. Towards mental time travel: a hierarchical memory for reinforcement learning agents. *Advances in Neural Information Processing Systems*, 34: 28182–28195.

Lu, C.; Schroecker, Y.; Gu, A.; Parisotto, E.; Foerster, J.; Singh, S.; and Behbahani, F. 2023. Structured State Space Models for In-Context Reinforcement Learning. *arXiv:2303.03982*.

Melo, L. C. 2022. Transformers are Meta-Reinforcement Learners. *arXiv:2206.06614*.

Mishra, N.; Rohaninejad, M.; Chen, X.; and Abbeel, P. 2018. A Simple Neural Attentive Meta-Learner. *arXiv:1707.03141*.

Morad, S. D.; Liwicki, S.; Kortvelesy, R.; Mecca, R.; and Prorok, A. 2021. Graph Convolutional Memory using Topological Priors. *arXiv:2106.14117*.

Ni, T.; Ma, M.; Eysenbach, B.; and Bacon, P.-L. 2023. When Do Transformers Shine in RL? Decoupling Memory from Credit Assignment. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Oh, J.; Chockalingam, V.; Singh, S.; and Lee, H. 2016. Control of Memory, Active Perception, and Action in Minecraft. *arXiv:1605.09128*.

Parisotto, E.; and Salakhutdinov, R. 2017. Neural Map: Structured Memory for Deep Reinforcement Learning. *arXiv:1702.08360*.

Parisotto, E.; Song, F.; Rae, J.; Pascanu, R.; Gulcehre, C.; Jayakumar, S.; Jaderberg, M.; Kaufman, R. L.; Clark, A.; Noury, S.; et al. 2020. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, 7487–7498. PMLR.

Pramanik, S.; Elelimy, E.; Machado, M. C.; and White, A. 2023. Recurrent Linear Transformers. *arXiv preprint arXiv:2310.15719*.

Robine, J.; Höftmann, M.; Uelwer, T.; and Harmeling, S. 2023. Transformer-based World Models Are Happy With 100k Interactions. In *The Eleventh International Conference on Learning Representations*.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *Nature*, 323: 533–536.

Samsami, M. R.; Zholus, A.; Rajendran, J.; and Chandar, S. 2024. Mastering Memory Tasks with World Models. *arXiv:2403.04253*.

Shala, G.; Biedenkapp, A.; and Grabocka, J. 2024. Hierarchical Transformers are Efficient Meta-Reinforcement Learners. *arXiv:2402.06402*.

Smith, J. T. H.; Warrington, A.; and Linderman, S. W. 2023. Simplified State Space Layers for Sequence Modeling. *arXiv:2208.04933*.

Song, D. R.; Yang, C.; McGreavy, C.; and Li, Z. 2018. Recurrent Deterministic Policy Gradient Method for Bipedal Locomotion on Rough Terrain Challenge.

Sorokin, I.; Seleznev, A.; Pavlov, M.; Fedorov, A.; and Ignateva, A. 2015. Deep Attention Recurrent Q-Network. *arXiv:1512.01693*.

Team, A. A.; Bauer, J.; Baumli, K.; Baveja, S.; Behbahani, F.; Bhoopchand, A.; Bradley-Schmieg, N.; Chang, M.; Clay, N.; Collister, A.; Dasagi, V.; Gonzalez, L.; Gregor, K.; Hughes, E.; Kashem, S.; Loks-Thompson, M.; Openshaw, H.; Parker-Holder, J.; Pathak, S.; Perez-Nieves, N.; Rakicevic, N.; Rocktäschel, T.; Schroecker, Y.; Sygnowski, J.; Tuyls, K.; York, S.; Zacherl, A.; and Zhang, L. 2023. Human-Timescale Adaptation in an Open-Ended Task Space. *arXiv:2301.07608*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wierstra, D.; Förster, A.; Peters, J.; and Schmidhuber, J. 2010. Recurrent policy gradients. *Logic Journal of the IGPL*, 18: 620–634.

YuXuan Liu, T. D.; and Hsieh, W. 2016. Temporal Convolutional Policy Networks.

Zaremba, W.; and Sutskever, I. 2016. Reinforcement Learning Neural Turing Machines - Revised. *arXiv:1505.00521*.

Zhu, D.; Li, L. E.; and Elhoseiny, M. 2023. Value Memory Graph: A Graph-Structured World Model for Offline Reinforcement Learning. *arXiv:2206.04384*.

Zintgraf, L.; Shiarlis, K.; Igl, M.; Schulze, S.; Gal, Y.; Hofmann, K.; and Whiteson, S. 2020. VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning. *arXiv:1910.08348*.