Large-Scale Retrieval for Reinforcement Learning

Peter C. Humphreys[*], **Arthur Guez**[*], **Olivier Tieleman,**
Laurent Sifre, Théophane Weber, Timothy Lillicrap
Deepmind, London
{peterhumphreys, aguez, ...}@google.com

Abstract

Effective decision making involves flexibly relating past experiences and relevant contextual information to a novel situation. In deep reinforcement learning (RL), the dominant paradigm is for an agent to amortise information that helps decision-making into its network weights via gradient descent on training losses. Here, we pursue an alternative approach in which agents can utilise large-scale context-sensitive database lookups to support their parametric computations. This allows agents to directly learn in an end-to-end manner to utilise relevant information to inform their outputs. In addition, new information can be attended to by the agent, without retraining, by simply augmenting the retrieval dataset. We study this approach for offline RL in 9x9 Go, a challenging game for which the vast combinatorial state space privileges generalisation over direct matching to past experiences. We leverage fast, approximate nearest neighbor techniques in order to retrieve relevant data from a set of tens of millions of expert demonstration states. Attending to this information provides a significant boost to prediction accuracy and game-play performance over simply using these demonstrations as training trajectories, providing a compelling demonstration of the value of large-scale retrieval in offline RL agents.

1 Introduction

How can reinforcement learning (RL) agents leverage relevant information to inform their decisions? Deep RL agents have typically been represented as a monolithic parametric function, trained to gradually amortise useful information from experience by gradient descent. This has been effective [25, 38], but is a slow means of integrating experience, with no straightforward way for an agent to incorporate new information without many additional gradient updates. In addition, this requires increasingly massive models (see e.g. [31]) as environments become more complex – scaling is driven by the dual role of the parametric function, which must support both computation and memorisation. Finally, this approach has a further drawback of particular relevance in RL – the only way in which previously encountered information (that is not contained in working memory) can aid decision making in a novel situation is indirectly through weight changes mediated by network losses. There is no end-to-end means for an agent to attend to information outside of working memory to directly inform its actions. While there has been a significant amount of work focused on increasing the information available from previous experiences *within an episode* (e.g., recurrent networks, slot-based memory [19, 28]), more extensive direct use of more general forms of experience or data has been limited, although some recent works have begun to explore utilising inter-episodic information from the same agent [6, 10, 29, 34, 41]. We seek to drastically expand the scale of information that is accessible to an agent, allowing it to attend to tens of millions of pieces of information, while learning in an end-to-end manner how to use this information for decision making. We view this as a first step towards a vision in which an agent can flexibly draw on diverse and large-scale information sources,

[*]These authors contributed equally to this work.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

including its own (inter-episodic) experiences along with experiences from humans and other agents. In addition, given that retrieved information need not match the agent's observation format, retrieval could enable agents to integrate information sources that have not typically been utilised such as videos [2], text, and third-person demonstrations.

We investigate a semi-parametric agent architecture for large-scale retrieval, in which fast and efficient approximate nearest neighbor matching is used to dynamically retrieve relevant information from a dataset of experience. We evaluate this approach in an offline RL setting for an environment with a combinatorial state space – the game of 9x9 Go with $\approx 10^{38}$ possible games, where generalisation from past data to novel situations is challenging. We equip our agent with a large-scale dataset of $\sim$50M Go board-state observations, finding that a retrieval-based approach utilising this data is able to consistently and significantly outperform a strong non-retrieval baseline.

Several key advantages of this retrieval approach are worth highlighting: Instead of having to amortise all relevant information into its network weights, a retrieval-augmented network can utilise more of its capacity for computation. In addition, this semi-parametric form allows us to update the information available to the agent at evaluation time without having to retrain it. Strikingly, we find that this enables improvements to agent performance without further training when games played against the evaluation opponent are added to its knowledge base.

2 Methods

Before introducing the details of our method, let us first present its high-level ingredients. We train a model-based agent [36] that predicts future policies and values conditioned on future actions in a given state. This semi-parametric model incorporates a retrieval mechanism, which allows it to utilise information from a large-scale dataset to inform its predictions. We train the agent in a supervised offline RL setting, which allows us to directly evaluate the degree to which retrieval improves the quality of the model predictions. We subsequently evaluate the resulting model, augmented by Monte-Carlo tree search, against a reference opponent. This allows us to determine how these improvements translate to an effective acting policy for out-of-training-distribution situations.

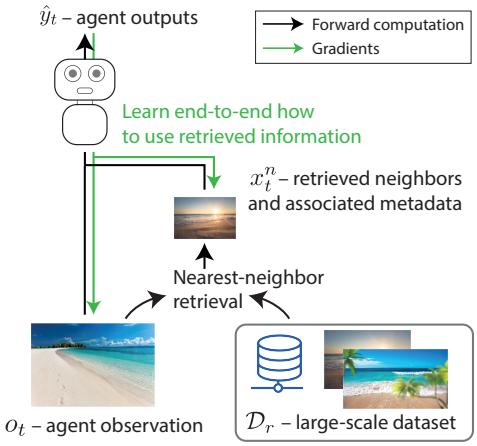

Figure 1: Retrieving information to support decision making. The agent observation o_t is used to generate a query to retrieve relevant information from a large-scale dataset $\mathcal{D}_r$. This information is used to inform network outputs $\hat{y}_t$. The agent is trained end-to-end to use this information to inform its decisions.

To effectively improve model predictions using auxiliary information, we need 1) a scalable way to select and retrieve *relevant* data and 2) a robust way of leveraging that data in our model. As with many successful attention mechanisms, we establish relevance for 1) through an inner product in an appropriate key-query space and select the top N relevant results. Choosing the relevant key-query space is critical, since performing nearest-neighbor on the raw observations will only deliver desirable results for the simplest of domains. For example, in the game of Go, a single stone difference on the board can dramatically change the outcome and reading of the board, while seemingly different board positions might still share high-level characteristics (e.g. influence, partial local sequences, and life-and-death of groups).

At the scale we want to operate, learning the key & query embeddings end-to-end in order to optimize the final model predictions is challenging. Instead, as in the language modelling work by Borgeaud et al. [7], we learn an embedding function through a surrogate procedure and use the resulting frozen function to describe domain-relevant similarity. Moreover, scale also constrains the nearest-neighbors lookup to be approximate, since getting the true nearest-neighbors is too time-consuming at inference time and good approximations can be obtained.

To address 2) and make the best use of the relevant data, we provide the data associated with the nearest-neighbor as additional input features to the parametric part of our model, so that the network can learn to interpret it. We provide some light inductive biases in the architecture to ensure permutation invariance in the neighbors and robustness to outliers and distribution shifts (see Secs. 2.1.3 & 2.3). Letting the network interpret the nearest-neighbor data is essential in large and complex environments, such as Go, because the typically imperfect match between the current situation and retrieved neighbors will require different aspects of this retrieved data to be ignored or emphasized in a highly context specific manner. This is in contrast to episodic RL approaches like [6, 29] which prescribe how to use the retrieved data. Subsequent sections describe the model, retrieval process, and the full algorithm in more detail.

2.1 Retrieval-augmented agents

We consider a setting in which we wish to train a neural network model m_θ on a set of environment trajectories $\tau \in \mathcal{D}$ (for example, the dataset $\mathcal{D}^\pi$ of trajectories produced by a policy π). For each timestep t of τ, the model takes an observation o_t[2] and must produce predictions $\hat{y}$ of targets y_t.

In our retrieval paradigm, in addition to having access to o_t, the model also has access to an auxiliary dataset of knowledge or experience $\mathcal{D}_r$. The auxiliary set $\mathcal{D}_r$ could be the same as $\mathcal{D}$, but it can also contain more information, or in the most general case, information from a completely different distribution and in a different format. If there is overlap between $\mathcal{D}_r$ and $\mathcal{D}$, it is important for robustness to ensure that the model cannot retrieve the same trajectory as it is being trained on (otherwise the network will tend to overly trust information retrieved from $\mathcal{D}_r$). Common information sources in RL are trajectories from other agents or experts (offline RL) or the agent's own previous experiences (episodic memory, replay buffer). Note that, in contrast to offline RL, we do not assume that we should directly use this auxiliary data as trajectories to train on.[3]

We wish to use $\mathcal{D}_r$ to inform the model predictions $\hat{y}_t$, such that $\hat{y}_t = m_\theta(o_t, \mathcal{D}_r)$. This is challenging, as $\mathcal{D}_r$ is typically far too large to directly be consumed as a model input. One solution, shown in Fig. 1, is to adopt a retrieval-based model, wherein the parametric and differentiable portion of the model m_θ is provided with an informative subset of data $\{x_t^1, x_t^2, \ldots, x_t^N\}$ retrieved from $\mathcal{D}_r$, conditioned on o_t:

$$\hat{y}_t = m_\theta(o_t, x_t^1, x_t^2, \ldots, x_t^N). \tag{1}$$

This approach requires a number of design choices relating to the retrieval mechanism, which we explore further below.

2.1.1 Scalable nearest-neighbor retrieval using SCaNN

Inspired by previous work in language modelling, we chose to leverage the SCaNN architecture [11] for fast approximate nearest-neighbor retrieval. This requires each entry o_i in the dataset $\mathcal{D}_r$ to be associated with a key vector $k_i \in \mathbb{R}^d$, and a given observation o_t to be mapped to a query vector $q_t \in \mathbb{R}^d$. During retrieval, the squared Euclidean distances between q_t and the dataset keys k_i are used to determine which neighbors to retrieve – reminiscent of neural attention mechanisms. This retrieval process is very efficient and can be scaled to datasets with billions of items.

We will typically want to retrieve further associated meta-data, or context, for each neighbor. For example, if a neighbor is part of a trajectory, we would like to retrieve information about action choices and their consequences. The neighbor observation o_i, together with its meta-data, forms the auxiliary input x_t^i to the model.

The nearest-neighbor retrieval process is non-differentiable, which means that the query and key mappings cannot be trained end-to-end directly. Instead, in this first study, we pre-train a non-retrieval prediction network m_ϕ^e on our experience dataset $\mathcal{D}$ (details in Sec. A.4). We then use this network to generate an embedding corresponding to a given observation o_t by retrieving the network activations from a specified layer of m_ϕ^e. We use principal component analysis to compress these activations to a d dimensional vector representing this observation state. The embedding and projection step together form our key (& query[4]) network $k_i = g_\phi(o_i)$.

[2] For simplicity of notation, we omit the dependence on past observations for partially-observable domains.

[3] For example, $\mathcal{D}_r$ needs not contain actions, rewards, or the full context used to select actions [9].

[4] In this study, retrieval dataset entries and o_t have the same format, but this need not be the case in general.

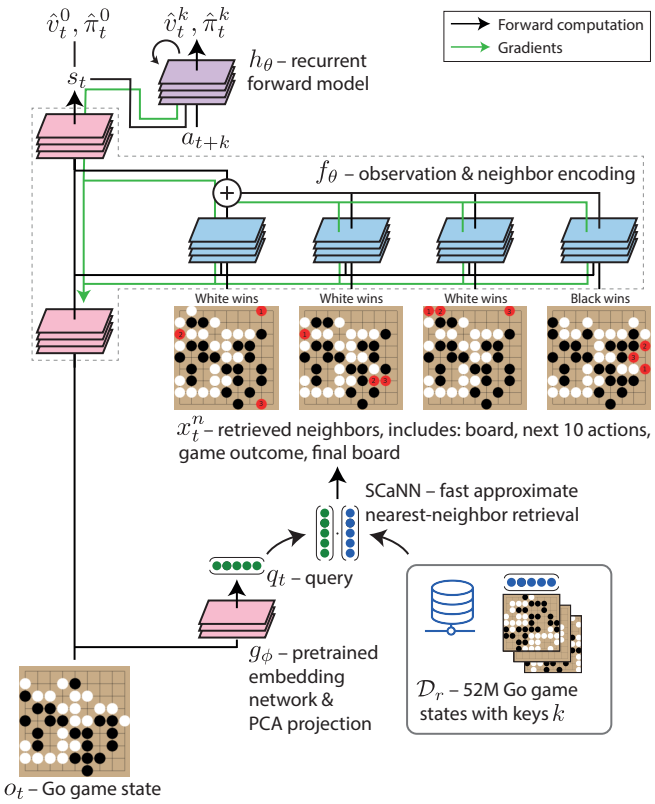

Figure 2: Details of the architecture used for a retrieval-augmented Go playing agent. A pre-trained network is used to generate a query q_t corresponding to the current Go game state o_t. This query is used for fast approximate nearest-neighbor retrieval using SCaNN. Retrieved neighbors x_t^n are processed using an invariant architecture, and used to inform an action-conditional recurrent forward model that outputs game outcome predictions $\hat{v}^k$ and distributions over next actions $\hat{\pi}^k$.

We preprocess all of the observations in $\mathcal{D}_r$ using g_ϕ to produce corresponding keys. The resulting dataset of keys and observations is then used by ScaNN to retrieve neighbors given a query vector. For offline RL training with a fixed training dataset, we can also preprocess the nearest neighbor lookups for all training dataset observations, i.e., $q_t = g_\phi(o)\ \forall o_t \in \mathcal{D}$. However, to *act* during evaluation, we must do this nearest-neighbor lookup online for each new observation. In future experiments, we intend to incorporate end-to-end learning of the query vector [12] to improve agent performance. A future online RL pipeline will also require us to dynamically retrieve neighbors during training.

2.1.2 Policy optimization with retrieval

The objective in our offline RL setting is to leverage the available data in order to optimize a policy π for acting. Our proposed semi-parametric model (Eqn. 1) is compatible with several methods for policy optimization in this offline setting. Indeed, it can represent the actor $\pi_\theta(o_t, \mathcal{D}_r)$ and/or the critic $Q_\theta(o_t, a, \mathcal{D}_r)$ in an offline actor-critic method [21], or the Q-value in a value iteration approach [32]. Here we focus on a *model-based* approach, inspired by MuZero [36], where the learned model is employed in a search procedure based on Monte-Carlo Tree Search (MCTS) [8]. This is motivated by the efficacy of MuZero in the offline RL setting [37, 24].

Model-based search with retrieval Following MuZero, our prediction model m_θ is conditioned on the current observation o_t but also on a sequence of future actions $\vec{a}_t = a_{t+1}, a_{t+2}, \ldots, a_{t+K}$. The model m_θ is therefore redefined as $\hat{y}_t = m_\theta(o_t, \vec{a}_t, x_t^1, x_t^2, \ldots, x_t^N)$. The architecture for this retrieval prediction model is illustrated in Fig. 2. The model m_θ can be decomposed into an encoding step $s_t = f_\theta(o_t, x_t^1, x_t^2, \ldots, x_t^n)$, which incorporates the observation and neighbor information, followed by iterative action-conditioned inference $s_t^{k+1} = h_\theta(s_t^k, a_{t+k})$ to produce embeddings

s_t^k corresponding to subsequent time steps (where $s_t^0 = s_t$). The model output $\hat{y}_t$ is composed of $K+1$ value and policy distributions for the current and next K time-steps: $\{\hat{v}_t^k, \hat{\pi}_t^k\}_{k=0\ldots K}$. [5] These outputs are obtained from their respective embeddings s_t^k. The target for value predictions is the game outcome and the targets for the policy predictions are the actions taken by the expert $a_t, a_{t+1}, \ldots, a_{t+K}$. The sample loss $\mathcal{L}(\theta, \hat{y}_t, y_t)$ is obtained by summing the $K+1$ individual loss terms for value and policy predictions (see detail in App. A.2). The training procedure for this retrieval prediction model is summarized in Alg 1.

Algorithm 1 Training semi-parametric action-conditional model

Input: Training dataset $\mathcal{D}$ and retrieval dataset $\mathcal{D}_r$
Initialize parameter vectors θ, ϕ.
Train key/query network g_ϕ using $\mathcal{D}$.
Pre-compute $k_i = g_\phi(o_i)$ for all $o_i \in \mathcal{D}_r$.
for each gradient step **do**
    Sample mini-batches from $\mathcal{D}$ of (o_t, y_t).
    For each element o_t of the batch, compute $q_t = g_\phi(o_t)$.
    Fetch N neighbor keys with (approx.) smallest distance $||q_t - k_i||_2^2$ for all $k_i \in \mathcal{D}_r$.
    Gather meta-data associated with these keys as $x_t^1, \ldots x_t^n$.
    Compute model output $\hat{y}_t = m_\theta(o_t, \vec{a}, x_t^1, \ldots, x_t^N)$.
    Compute and sum losses $\mathcal{L}(\theta, \hat{y}_t, y_t)$.
    Update θ based on $\nabla_\theta \mathcal{L}$.
end for
output θ, ϕ

In order to act using the trained model, online search is used to generate an improved policy $\pi_s = \text{Search}(m_\theta, \mathcal{D}_r)$. We use MCTS with a pUCT rule for internal action selection [35, 40], carrying out n_{sims} simulations per time step. Model-based search is implemented as follows: after retrieving the observation's neighbors, the encoded state s_t is computed. Search is carried out from s_t by varying the input action sequence $\vec{a}$ for each simulation and collecting the model outputs to update search statistics. Details of this process can be found in [36]. At the end of the search, the resulting policy $\pi_s(a|o_t)$ can be sampled to select the next action.

Due to the semi-parametric nature of the model supporting the search, introducing changes to $\mathcal{D}_r$ will have an immediate effect on the acting policy π_s, even if the model parameters θ remain fixed. As we explore in Sec. 3.4, this provide a mechanism for fast adaption to new information or experiences.

2.1.3 Using neighbor information

Several choices are possible for the network f_θ that processes the observation and neighbors. Since this is not the main focus of this work, we chose a straightforward approach. We first compute an embedding o_t^e of the observation o_t. We then compute an embedding for each neighbor $e_i^t = p_\theta(o_t^e, x_i^t)$, using the same network p_θ for all neighbors. These streams are combined in a permutation invariant way through a sum, and then concatenated with o_t^e to produce the final embedding s_t, :

$$s_t = f_\theta(o_t, x_1, \ldots, x_N) = [o_t^e, \frac{\sum_{i=1}^N e_i^t}{\sqrt{N}}]. \tag{2}$$

2.2 Evaluating retrieval in the domain of Go

In order to test the utility of retrieval in RL, we wish to evaluate whether agents can effectively generalise between related but distinct experiences, as opposed to simply retrieving the outcome of a previous instance of an identical situation. This motivates our choice of Go as a domain. We focus on 9x9 Go, instead of full-scale 19x19 Go, as the less demanding computational costs of 9x9 experiments enable a more thorough analysis. Even for 9x9 Go, the state space of 10^{38} possible games is vastly larger than the number of positions that we could hope to query.

We collected a dataset of ∼3.5M expert 9x9 Go self-play games from an AlphaZero-style agent [40]. We randomly subsampled 15% of the positions from these games, leaving us with ∼50M board

[5] In cases where non-terminal rewards exist, we also output reward estimates for each model transition.

state observations. These, along with metadata on the game outcome, final game board, and future actions (all encoded as input planes), form our retrieval dataset $\mathcal{D}_r$. We chose to subsample as we hypothesised that this would reduce the chance of retrieving multiple neighbors from the same trajectory, and therefore boost the retrieved neighbor diversity. However, we did not perform ablations on this choice. A future solution would be to use a filtering mechanism to reject neighbors from the same trajectory. In this initial work, training and retrieval datasets are the same – at least during training. During training, we split $\mathcal{D}_r$ in two halves such that each game's observations are only in one of the datasets. We retrieve neighbors for an observation o_t from the half it is not contained in. This is simply to avoid retrieving the same position as the query. Other ways to obtain the same effect could be devised.

2.3 Regularisation

We wish to make our network robust to poor quality neighbors, in order to ensure that the network can perform well in settings for which there is lower overlap with the retrieval dataset than encountered in training. We therefore explore several techniques to improve network robustness to irrelevant neighbors. We randomly zero-out a subset of retrieved neighbors during training ("neighbor dropout"), and/or more adversarially, randomly replace a subset of retrieved neighbors with the neighbors of a different observation ("neighbor randomisation"). Inspired by [10], we also explore using a loss to regularise the embedding produced by the neighbor retrieval towards the embedding produced with the observation alone ("neighbor regularisation"). Further details are given in Sec. A.5.

We carried out ablations of these techniques (Appendix Fig. 7), which show that neighbor randomisation is important in some contexts for maintaining performance, but that the others do not seem to have a significant effect in our final configuration. The results reported in this study utilise all of these augmentations, as during the development process they had been found to slightly benefit performance. Interestingly, as we explore in later sections, MCTS with enough simulations compensates for the harmful effect of low-quality neighbors, and can perform effectively without these training augmentations.

3 Results

3.1 Qualitative examination of retrieved neighbors

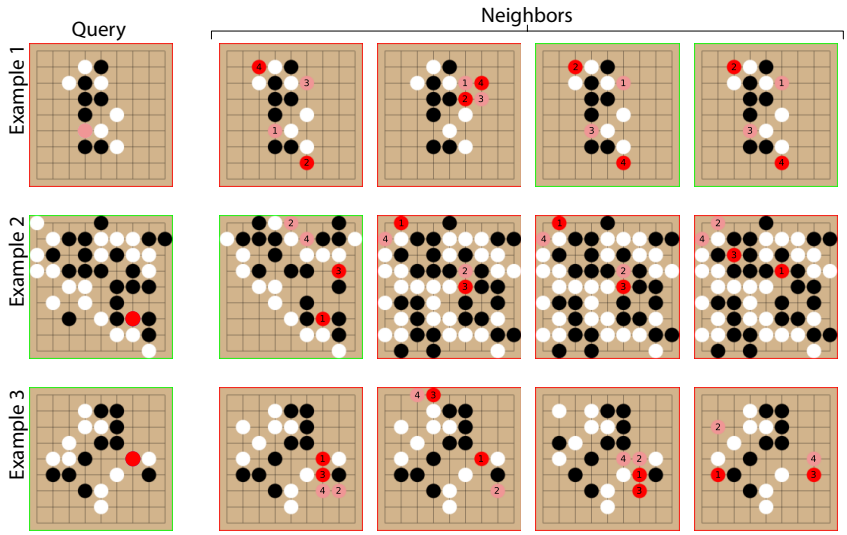

Figure 3: Visualisation of $N = 4$ approximate nearest-neighbors retrieved using a learned Go-specific distance function for 3 query positions (one per row). Red stone indicates the next action(s) – light red stones indicate white moves, dark red stones indicate black moves. The color of the board border indicates whether the current player to play (for each board) won the game.

Changing a single stone in a Go board can dramatically affect the game, and the number of Go positions is enormous. We first wanted to assess whether, despite the combinatorial aspect of the domain, meaningful nearest-neighbors could be retrieved from the dataset using our learned keys/queries. Examples of retrieved positions in Go are shown in Fig. 3, where we observed relevant matches in terms of both local and global structure. In some cases, especially in the early game, the retrieved position is an exact match even though the rest of the game (and therefore the associated nearest-neighbor meta-data) differs. Row 1 in Fig. 3 is an example of this – in this case, the retrieved data effectively provides sample rollouts from the query position akin to those derived from MCTS.

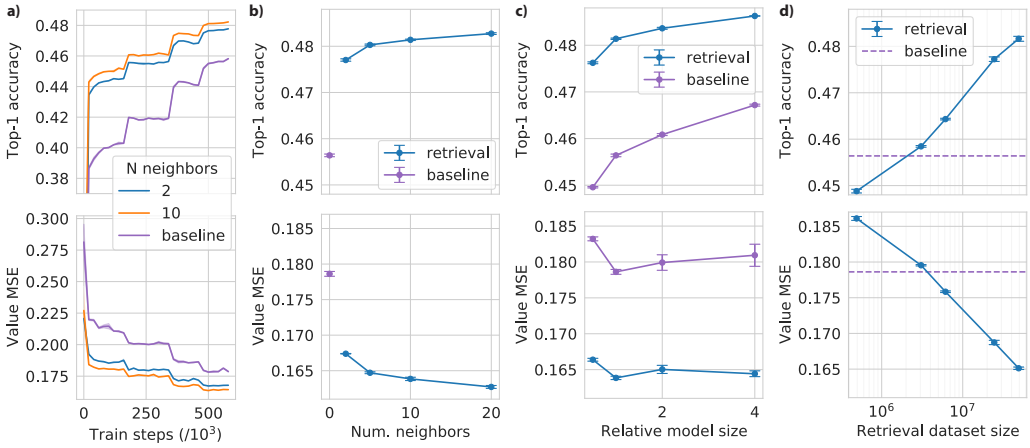

Figure 4: a) Test-set top-1 policy prior $\hat{\pi}^0$ accuracy (top) and value $\hat{v}^0$ mean-squared error (MSE) (bottom) over the course of training, for the (non-retrieval) baseline and retrieval models with $N = 2, 10$. Results are averaged over 3 seeds (error typically too small to see). Note that the sharp transitions are due to the learning rate schedule. b) Final performance after training plotted as a function of number of neighbors N retrieved — a different network is trained for each number of neighbors. c) Final performance as a function of model size relative to the size used elsewhere in this study, for networks with $N = 10$ retrieved neighbors and the non-retrieval baseline. d) Final performance as a function of the evaluation retrieval dataset size, for networks with $N = 10$ retrieved neighbors. The equivalent baseline performance is shown as a dashed line for reference. Note that the networks are trained with dataset fraction $= 0.5$ and simply evaluated at other dataset fractions.

3.2 Impact of retrieval on supervised training

We next evaluated the extent to which the model can learn to exploit retrieved information for better model predictions and better decision-making. First, we evaluated the impact of retrieval on supervised learning losses. As a baseline comparison, we trained the same network architectures but setting $x_i = 0$ for all i — this has the effect of maintaining the number of parameters, flops, and useful capacity in the network, while removing access to the retrieval data. We evaluated losses for retrieval networks trained with different N (Fig. 4b) and model sizes (Fig. 4c, see Sec. A.1 for details). Across conditions, we consistently observed a significant boost in test-set accuracy for all metrics and over the course of training. This improvement to predictions is further observed across game trajectories, and is not limited, for example, to opening play positions.

One advantage of the semi-parametric approach we outlined is that we can modify the retrieval dataset and potentially see immediate effects on the prediction, without changing the parameters θ. We first verified this by evaluating our model when allowed access to varying fractions of the full dataset $\mathcal{D}_r$ (Fig. 4d). We observed clear gains in prediction accuracy from increasing the size of the retrieval set (only half is used at train time). A further important observation is that large-scale datasets are clearly important - the evaluation metrics drop below the baseline level for a dataset $\mathcal{D}_r$ that is 1% the size of the full dataset.

3.3 Evaluation against a reference opponent

While the results observed on the offline dataset suggest a strong positive effect from learning to leverage the retrieval data, this is in the context of a fixed test data distribution that matches the retrieval data distribution. When deployed, games played against different opponents will likely diverge from that distribution. We evaluated the performance of our search policy π_s ($n_{\text{sims}} = 200$) by playing against a fixed reference opponent — the Pachi program [5], which can perform beyond strong amateur level of play in 9x9 Go given sufficient simulation budget (we evaluate against 400k simulations). We observed a significant boost to performance for retrieval-based networks over the equivalent baseline network of the same capacity (Fig. 5). Interestingly, as shown in Appendix Fig. 8, playing using only the base policy prior $\hat{\pi}^0$ shows a much smaller boost over the baseline network. As we explore further below, the quality of retrieved neighbors available during play is much lower than for training - we hypothesise that the search policy is more robust to this distributional shift than $\hat{\pi}^0$.

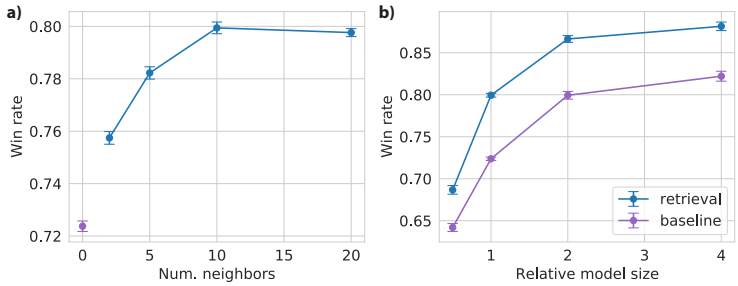

Figure 5: Win rate against a fixed reference opponent (Pachi) when playing using the MCTS policy π_s for (a) retrieval networks using varying numbers of retrieved neighbors and for a baseline non-retrieval network. (b) Win rate as a function of model size relative to the size used elsewhere in this study. Retrieval leads to a clear performance boost compared to non-retrieval baselines of the same capacity.

3.4 Augmenting the retrieval dataset

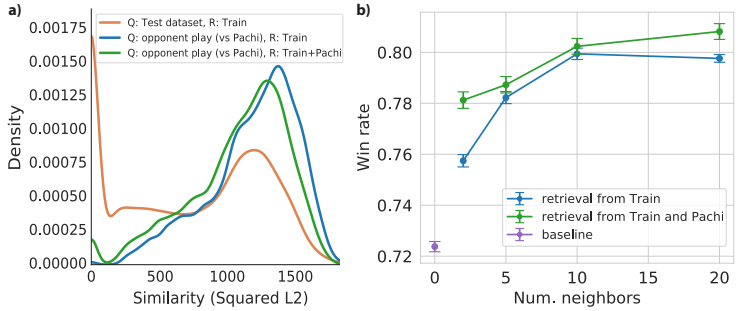

Figure 6: (a) Distribution of similarity distances from an encoded observation to its retrieved nearest neighbors. Retrieving from the original retrieval dataset using positions from the test dataset (orange) gives neighbors with a markedly different distance distribution to positions queried during play against a Pachi opponent (blue). Augmenting the retrieval dataset with ∼600k recorded agent-Pachi game states improves the similarity distribution (green). (b) For play with the MCTS policy π_s, this augmentation also leads to a consistent win-rate boost (green) over using the train retrieval dataset alone (blue).

We observed a significant distribution shift between game positions observed in play against the Pachi opponent versus positions in the retrieval dataset (Fig. 6a), as measured by the empirical distance distribution of game observations to their approximate nearest neighbors. Changing or augmenting the retrieval dataset $\mathcal{D}_r$ modifies this distributional shift. A particularly interesting modification in this setting is to augment the dataset with play recorded between our agents and the Pachi opponent. As can be seen in Fig. 6a, augmenting the dataset with a set of ∼600k agent-Pachi game states increases

the similarity between positions observed in play and their retrieved neighbors. Strikingly, we found that integrating these games into our retrieval dataset leads to improvements in performance without further training for subsequent play against the Pachi opponent (play with the MCTS policy π_s is shown in Fig. 6b, play with the policy prior $\hat{\pi}^0$ is shown in Appendix Fig. 9). This highlights the potential of a semi-parametric model to rapidly integrate recent experience without further training.

As a related intervention, we instead inputted randomly retrieved neighbors into the model at evaluation time (see Appendix Fig. 6). This significantly impaired performance for play using the policy prior $\hat{\pi}^0$, but somewhat unexpectedly, for play with the MCTS policy π_s, performance did not fully regress to the level of the baseline network. This suggests that our retrieval network is able to amortise some information from neighbors during training, leading to better performance even when no relevant neighbors are available.

4 Related work

The idea of supporting decision making by directly attending to a cache of related events has been visited many times in different contexts, under the names of case-based reasoning [30], non-parametric RL [16, 4, 1, 27], or episodic memory [20, 23]. The aim of some methods is to better support the working memory of an agent during a given episode [26, 42] or a series of successive related episodes [33]. Other methods, including ours, aim to leverage a broader class of relevant experience or persistent knowledge (including across episodes, and from other agents) to better support reasoning and planning. One differentiating factor in our work to these past approaches is that we do not prescribe how to process the information from the available data (e.g. through specifying the agent's action-value directly in terms of previously generated value estimates [6, 13, 15, 29], or a model from observed transitions [39]) but rather *learn* end-to-end how the data can support better predictions within the parametric model. A recent approach by Goyal et al. [10] has considered an attention mechanism to select where and what to use from available trajectories, but over a small retrieval batch of data rather than the full available experience data. Another class of method to leverage a transition dataset is to replay the data at *training time* in order to perform more gradient steps per experience, this is a widespread technique in modern RL algorithms [21, 22, 25, 36] but it does not benefit the agent at test time, requires additional learning steps to adapt to new data, and does not allow end-to-end learning of how to relate past experience to new situations.

5 Discussion

Our approach and empirical results highlight how reinforcement learning agents can benefit from direct access to a large collection of raw interaction data at inference time, through a retrieval mechanism, in addition to their already effective parametric representation. We showed this was the case even 1) when the domain is large enough to require generalisation in how to interpret past data, 2) when there is significant distribution shift when acting, and 3) at a scale where only approximate nearest-neighbors can be retrieved. We believe this already demonstrates the potential of this approach for many possible scenarios and applications.

We show that retrieval can be effectively combined with model-based search. We find that the benefits from retrieval and search are synergistic, with increasing numbers of retrieved neighbors and increasing simulations both leading to performance increases in almost all contexts we investigated. Furthermore, empirical evidence suggests that search significantly improves agent robustness to distributional shift as compared to playing with the policy prior. In Appendix Fig. 10, we compare the boost in performance as a function of parametric-model compute cost for increasing MCTS simulations versus increasing the number of retrieved neighbors processed by the model. This provides a tentative indication that retrieval is also a compute efficient means of improving performance.

A key future direction is to investigate the online learning scenario in which recent experience of the agent is rapidly made available for retrieval, hence progressively growing the retrieval dataset over the course of training. While there are additional challenges associated with the online paradigm (e.g., it may be desirable to update the queries and/or keys during training [12]), the fast adaptation effect we highlighted in this work may have even more impact there.

There are many potentially relevant sources of information beyond an agent's own experience, or that of other humans or agents. For example, it has been shown that YouTube videos are useful

for learning to play the Atari game Montezuma's revenge [2]. Training an embedding network on sufficiently diverse data may enable retrieval of information from a wide range of contexts [43], including third-person demonstrations, videos and perhaps even books.

Disclosure of Funding

The authors received no specific funding for this work.

References

[1] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11:75–113, 2004.

[2] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando De Freitas. Playing hard exploration games by watching youtube. *Advances in neural information processing systems*, 31, 2018.

[3] Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Claudio Fantacci, Jonathan Godwin, Chris Jones, Tom Hennigan, Matteo Hessel, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Lena Martens, Vladimir Mikulik, Tamara Norman, John Quan, George Papamakarios, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Wojciech Stokowiec, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL `http://github.com/deepmind`.

[4] André MS Barreto, Doina Precup, and Joelle Pineau. Practical kernel-based reinforcement learning. *The Journal of Machine Learning Research*, 17(1):2372–2441, 2016.

[5] Petr Baudiš and Jean-loup Gailly. Pachi: State of the art open source Go program. In *Advances in computer games*, pages 24–38. Springer, 2011.

[6] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

[7] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. *arXiv preprint arXiv:2112.04426*, 2021.

[8] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[9] Pim De Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[10] Anirudh Goyal, Abram L Friesen, Andrea Banino, Theophane Weber, Nan Rosemary Ke, Adria Puigdomenech Badia, Arthur Guez, Mehdi Mirza, Ksenia Konyushkova, Michal Valko, et al. Retrieval-augmented reinforcement learning. *arXiv preprint arXiv:2202.08417*, 2022.

[11] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896. PMLR, 2020.

[12] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020.

[13] Steven Hansen, Alexander Pritzel, Pablo Sprechmann, André Barreto, and Charles Blundell. Fast deep reinforcement learning using online adjustments from the past. *Advances in Neural Information Processing Systems*, 31, 2018.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[15] Mika Sarkin Jain and Jack W Lindsey. Semiparametric reinforcement learning. In *ICLR*, 2018.

[16] H. JoséAntonioMartín, Javier de Lope Asiaín, and Darío Maravall Gómez-Allende. The knn-td reinforcement learning algorithm. In *IWINAC*, 2009.

[17] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

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

[19] Andrew Lampinen, Stephanie Chan, Andrea Banino, and Felix Hill. Towards mental time travel: a hierarchical memory for reinforcement learning agents. *Advances in Neural Information Processing Systems*, 34, 2021.

[20] Máté Lengyel and Peter Dayan. Hippocampal contributions to control: the third way. *Advances in neural information processing systems*, 20, 2007.

[21] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[22] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.

[23] Zichuan Lin, Tianqi Zhao, Guangwen Yang, and Lintao Zhang. Episodic memory deep q-networks. *arXiv preprint arXiv:1805.07603*, 2018.

[24] Michael Mathieu, Sherjil Ozair, Srivatsan Srinivasan, Caglar Gulcehre, Shangtong Zhang, Ray Jiang, Tom Le Paine, Konrad Zolna, Richard Powell, Julian Schrittwieser, et al. Starcraft ii unplugged: Large scale offline reinforcement learning. In *Deep RL Workshop NeurIPS 2021*, 2021.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[26] Junhyuk Oh, Valliappa Chockalingam, Honglak Lee, et al. Control of memory, active perception, and action in minecraft. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2016.

[27] Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine learning*, 49(2): 161–178, 2002.

[28] 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.

[29] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International Conference on Machine Learning*, pages 2827–2836. PMLR, 2017.

[30] Ashwin Ram and Juan Carlos Santamaria. Continuous case-based reasoning. *Artificial Intelligence*, 90(1-2):25–77, 1997.

[31] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[32] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, pages 317–328. Springer, 2005.

[33] Sam Ritter, Ryan Faulkner, Laurent Sartran, Adam Santoro, Matt Botvinick, and David Raposo. Rapid task-solving in novel environments. *arXiv preprint arXiv:2006.03662*, 2020.

[34] Samuel Ritter, Jane Wang, Zeb Kurth-Nelson, Siddhant Jayakumar, Charles Blundell, Razvan Pascanu, and Matthew Botvinick. Been there, done that: Meta-learning with episodic recall. In *International conference on machine learning*, pages 4354–4363. PMLR, 2018.

[35] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

[36] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[37] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. *Advances in Neural Information Processing Systems*, 34, 2021.

[38] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[39] Aayam Shrestha, Stefan Lee, Prasad Tadepalli, and Alan Fern. Deepaveragers: offline reinforcement learning by solving derived non-parametric mdps. *arXiv preprint arXiv:2010.08891*, 2020.

[40] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[41] Pablo Sprechmann, Siddhant M Jayakumar, Jack W Rae, Alexander Pritzel, Adria Puigdomenech Badia, Benigno Uria, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell. Memory-based parameter adaptation. *arXiv preprint arXiv:1802.10542*, 2018.

[42] 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.

[43] Hu Xu, Gargi Ghosh, Po-Yao Huang, Prahal Arora, Masoumeh Aminzadeh, Christoph Feichtenhofer, Florian Metze, and Luke Zettlemoyer. Vlm: Task-agnostic video-language model pre-training for video understanding. *arXiv preprint arXiv:2105.09996*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] No theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] Instructions and links to relevant libraries were provided.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [N/A]

   (b) Did you mention the license of the assets? [N/A]

   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] No people were involved in the data gathering process.

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

The appendix details the architecture and training mechanisms for our results, it also lists additional ablations and results.

A.1 Architecture

Observations consist of 9x9 feature planes representing the current state of the board (indicating stones for the last two moves, and the player color). The networks we train are convolutional, composed of residual blocks, and maintain the 9x9 topology throughout — except for the final output layers.

As illustrated in Figure 1, the main inference network $s_t = f_\theta(o_t, x_t^1, x_t^2, \ldots, x_t^n)$ is composed of an encoding of the observation o_t using $m_{\text{enc}} = 2$ residual 3×3 convolutional blocks with 256 channels (each block consists of two convolutional layers with a skip connection) [14], preceded by a first convolution layer with 256 filters and followed by a convolution layer that goes back to 128 channels.

Each neighbor information x^i is concatenated separately with the encoded observation o^e to be processed by another residual network p_θ (blue layers in Fig. 1), but with $m_{\text{nn}} = 3$ blocks and 256 channels. The neighbor information, in addition to the feature planes corresponding to the retrieved position, also include 1-hot 9x9 input planes for the 10 next actions, a tiled input plane for the game outcome, and the feature planes corresponding to the final game board. The parameters for each neighbor processing stream are shared. The output from each of the neighbors e_i are then summed (and normalized) and concatenated back with the encoded observation along the channel dimension, see Eq. 2. The concatenated tensor is processed by $m_{\text{root}} = 6$ additional residual blocks with the same hyperparameters to produce the main output embedding $s_t = s_t^0$ (top pink layers in Fig. 1).

Each internal transition in the recurrent action-conditional model (in purple Fig. 1) takes the previous embedding s_t^k, concatenates a one-hot representation of the action a_{t+k}, and outputs the next embedding s_t^{k+1}. These internal models are composed of $m_{\text{tran}} = 3$ residual blocks preceded by a convolution layer to bring the activations back to 256 layers. All internal transitions, and output networks for $k > 1$, share parameters. Non-linear transformations after each layer are rectified linear units (ReLU), preceded by a layer norm layer applied spatially within the residual blocks. The model outputs are obtained at each step k from the embedding s_t^k with a value and policy head. The value head first applies a single 1×1 convolution filter, followed by a ReLU, a flattening operation, and an MLP with a single hidden layer of 256 units to a final scalar output passed through a $\tanh$ non-linearity to obtain $\hat{v}_t^k$. The policy head follows the same pattern, but applies two initial convolution filters, and the final output has dimension $82 = 9 \times 9 + 1$ (all possible actions including pass in 9x9 Go) and corresponds to the logits for $\hat{\pi}_t^k$.

Together, this describes the parametric network m_θ. The search policy $\pi_s = \text{Search}(m_\theta, \mathcal{D}_r)$ employs the trained network to plan and act. The main embedding $s_t = f_\theta(o_t, x_t^1, x_t^2, \ldots, x_t^n)$, and the retrieval process to obtain the neighbors, is computed once per search. Multiple rollouts can be computed from s_t with different input action sequences for a given root observation. The details of the MCTS search mechanism using the trained model follows [36] closely, we refer the reader to that paper for details.

For experiments that vary the model size, model size of 1 is as described above. For model sizes $n = 2$ & 4, we multiply m_{root}, m_{tran}, m_{enc}, and m_{nn} by a factor of 2, with the number of channels at all layers doubled for $n = 4$. The model size labeled 0.5 corresponds to $m_{\text{root}} = 3, m_{\text{tran}} = 2, m_{\text{enc}} = 1$, and $m_{\text{nn}} = 2$.

A.2 Losses and optimization

The total loss, for a single datapoint, is:

$$\mathcal{L}(\theta, y, y^*) = \sum_{k=0}^{K} w_k (l^p(\hat{\pi}_t^k, a_{t+1+k}) + l^v(\hat{v}_t^k, g)) + \alpha ||\theta||^2, \tag{3}$$

with $w_k = \frac{1}{K}$ for $k > 0$ and 1 otherwise ($K = 5$ in our experiments), g the empirical return (the $-1, 1$ relative game outcome for Go), and $\alpha = 1\text{e-}4$ controls the weight decay. The individual loss

terms are $l^v(v, g) = \frac{1}{2}(v - g)^2$ for the value outputs (MSE when averaged over the batch elements), and $l^p(\pi, a) = -\log(\pi(a))$ for the policy outputs. In addition, an optional regularisation loss is described below in Sec. A.5.

We use the Adam optimizer [18] with a learning rate schedule, training for 600k steps with a batch size of 1024. The learning rate is the initial learning rate divided by $\{2, 8, 64, 256\}$ after respectively {180k, 360k, 480k, 570k} steps, with the initial learning rate of 1e-3. Training was performed on TPU [17] using JAXline and Mctx within the DeepMind JAX Ecosystem [3].

A.3 Go game configuration

The komi throughout the experiments is 5.5. The Pachi program is configured with 16 threads, a `max_tree_size` of 2000, a `resign_threshold` of 0.1, no pondering, and the Chinese ruleset. The search policy π_s when testing is defined as the action with the maximum visit counts. To ensure diversity in the evaluation games against Pachi, we seed each game by letting Pachi play the first 6 moves in the opening. After an evaluation game goes past the opening moves and 50 subsequent moves, we stop the game if the value function is above 0.995, or below 1-0.995, to avoid playing long end-games when the game is already clearly settled.

A.4 Embedding network

To train the embedding g_ϕ to map observations o_t to keys and queries, we first train a MuZero model $m^e_\phi(o_t, \vec{a}_t)$ following the same method as in A.1-A.2. We train this on the same data $\mathcal{D}$ that we train the retrieval network on, but without any retrieval input or neighbor processing network. We also use 8 residual blocks for the main encoding stage to compute $s_t = f_\phi(o_t)$, and 3 residual blocks per internal model transition.

At the end of training, we choose the output of one of the f_ϕ network layers as a (pre)-embedding $\bar{g}_\phi(o)$. To reduce the dimensionality of $\bar{g}_\phi$, we project it using PCA into the first $d = 512$ principal components to obtain $g_\phi(o) = (\bar{g}_\phi(o) - \mu)V \in \mathbb{R}^d$, where V contains the first d eigenvectors of the covariance matrix of the pre-embedding vectors, estimated with a subset of the data after removing their mean μ.

In our experiments, we used the output of the 6th residual block of f_ϕ as $\bar{g}_\phi(o)$. We have not extensively searched for the best such embedding due to the expense of testing each – further investigation of this design choice will be the subject of future research.

A.5 Regularisation ablations

Here we provide more detail on techniques we investigated to improve network robustness to low quality retrieved neighbors (as introduced in Sec. 2.3).

Neighbor dropout

We randomly zero-out a subset of retrieved neighbors during training. For N retrieved neighbors, the number of neighbors to zero out M is uniformly randomly chosen from $[0, N]$. We then randomly choose M out of N neighbors, and mask out their contribution to the final invariant neighbor embedding (Sec. 2.1.3).

Neighbor randomisation

We randomly replace a subset of retrieved neighbors with the neighbors of a different observation within the mini-batch. This is implemented by randomly choosing M neighbors as described for dropout above, and then replacing each by a randomly chosen member of the mini-batch (which may with a small chance be the correct neighbor).

Neighbor regularisation

Inspired by [10], we also explore using a loss to regularise the embedding produced by the neighbor retrieval towards the embedding produced with the observation alone. In this case, we apply a convolutional layer to the embedding produced from the game state observation, and use this to

predict the output of the neighbor processing tower. We apply a mean-squared-error loss to encourage the base embedding to be predictive of the neighbor output, and similarly apply an equivalent loss to the neighbor output.

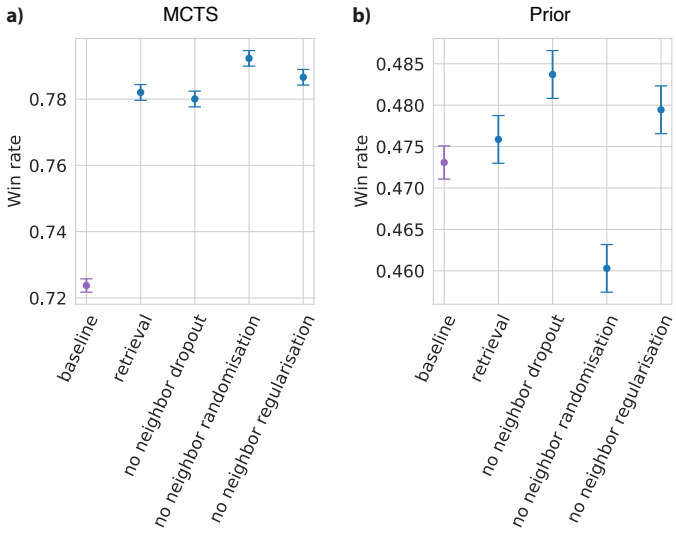

Figure 7: Ablations of different regularisation techniques explored in this study, as measured by evaluating the win rate against the Pachi reference opponent. Win rates are shown for (a) the MCTS policy π_s and (b) the policy prior $\hat{\pi}^0$. The most critical regularisation is neighbor randomisation. Without this randomisation, the retrieval network prior $\hat{\pi}^0$ performs significantly worse than the baseline non-retrieval network. The other regularisation methods do not have a significant effect on performance.

A.6 Additional results

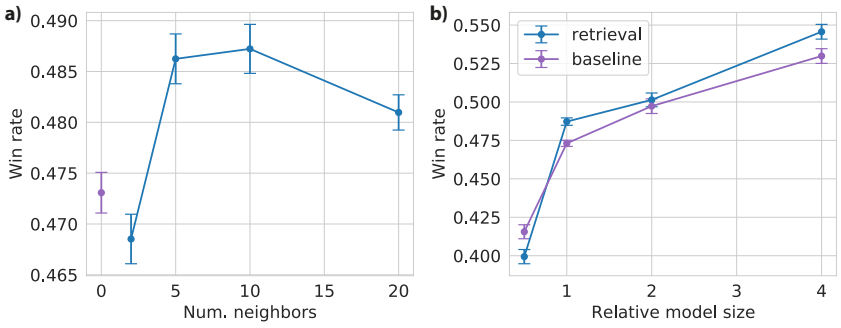

Figure 8: Equivalent figure to Fig. 5, but for playing with the policy prior $\hat{\pi}^0$, as opposed to with the MCTS policy π_s. Win rate against a fixed reference opponent (Pachi) for (a) retrieval networks using varying numbers of retrieved neighbors and a baseline non-retrieval network. (b) Win rate as a function of model size relative to the size used elsewhere in this study.

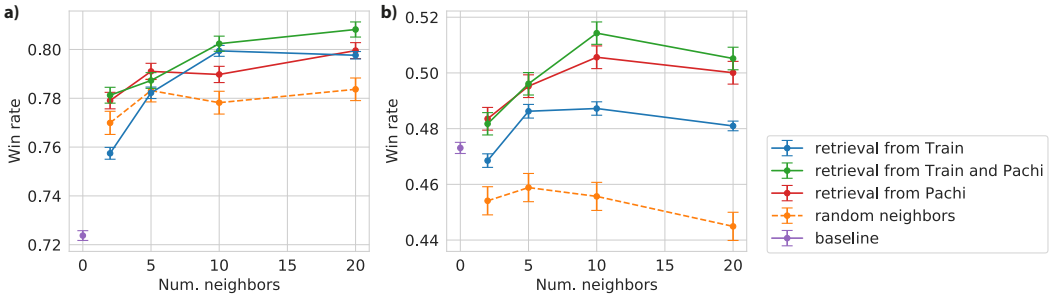

Figure 9: (a) Extended version of Fig. 6b: Changes to win rate for the MCTS policy π_s against the Pachi opponent without further training by adjusting the retrieval dataset. Augmenting the dataset with agent-Pachi games (green) leads to a consistent boost in performance over using the train retrieval dataset by itself (blue), and over retrieving from only agent-Pachi games (red). Using randomly chosen neighbors (dashed, yellow) hurts performance, but does not regress to the level of the non-retrieval network of the same capacity (purple). (b) Playing with the policy prior $\hat{\pi}^0$, as opposed to with the MCTS policy π_s. In this setting, the retrieval network performs at only a slightly higher level than the baseline, suggesting that prior play is less robust to the distributional shift from training to playing against Pachi than for MCTS. Nonetheless, as for MCTS, augmenting the dataset with games recorded against Pachi leads to a boost in performance. Using randomly chosen neighbors brings performance below the baseline, showing that our networks are not fully robust to invalid neighbors.

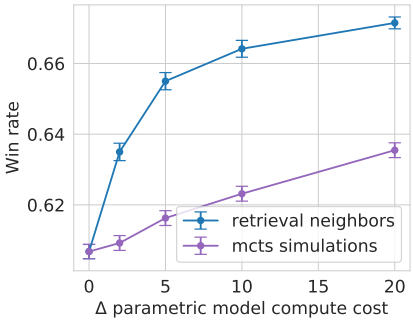

Figure 10: Change in win rate as a function of additional parametric-model computational cost for processing increasing numbers of retrieved neighbors versus increasing the number of MCTS simulations. The comparison is made relative to a baseline non-retrieval network using 50 MCTS simulations. In our configuration, each MCTS simulation uses the same amount of compute as processing one neighbor. Note that this comparison does not account for the computational cost of querying for and retrieving nearest neighbors.