LEARNING TO QUERY HISTORY: NONSTATIONARY CLASSIFICATION VIA LEARNED RETRIEVAL

Jimmy Gammell[*α] **Bishal Thapaliya**[*β] **Bilel Fehri**[β]

Riyasat Ohib[γ] **Deepayan Chakrabarti**[β]

ABSTRACT

Nonstationarity is ubiquitous in practical classification settings, leading deployed models to perform poorly even when they generalize well to holdout sets available at training time. We address this by reframing nonstationary classification as time series prediction: rather than predicting from the current input alone, we condition the classifier on a sequence of historical labeled examples that extends beyond the training cutoff. To scale to large sequences, we introduce a learned discrete retrieval mechanism that samples relevant historical examples via input-dependent queries, trained end-to-end with the classifier using a score-based gradient estimator. This enables the full corpus of historical data to remain on an arbitrary filesystem during training and deployment. Experiments on synthetic benchmarks and Amazon Reviews '23 (electronics category) show improved robustness to distribution shift compared to standard classifiers, with VRAM scaling predictably as the length of the historical data sequence increases.

Track: Research

1 INTRODUCTION

Nonstationarity is a persistent problem in practical classification settings such as policy violation detection, fraud classification, and compliance, where violation patterns evolve as trends change and bad actors adapt. We consider nonstationary classification settings where $X_t, Y_t \sim p_t$ and aim to predict Y_t from X_t. Models train on early data ($t < t_{\text{cutoff}}$) and must generalize to later times.

Deployed systems often accumulate a growing corpus of labeled historical examples that extends beyond the training cutoff up to the current time. We propose to leverage this by reframing nonstationary classification as time-series classification: rather than predicting Y_{t_0} solely from X_{t_0}, we also feed the classifier a long sequence of historical examples: $(X_{t_{-1}}, Y_{t_{-1}}, \ldots, X_{t_{-K}}, Y_{t_{-K}})$. This allows the model to adapt to distribution shift without re-training, by observing the relationship between inputs and labels for relevant recent examples.

Such sequences may be so long that it is impractical to naively feed them as an auxiliary input to the classifier, or even to store them in accelerator memory. Additionally, the number of examples which are relevant to the current query may be small. We thus propose a system with a learned retrieval mechanism which X_{t_0}-conditionally samples a small number of relevant items from the full sequence, then feeds these to a downstream classifier which predicts Y_t. The full system is trained end to end to optimize the classification objective, and the sequence may remain on an arbitrary filesystem during both training and deployment, with only the selected items and a small per-item key stored in accelerator memory.

To our knowledge, our work is the first to explore nonstationary retrieval-augmented classification by leveraging a large and growing corpus of historical data. We validate this approach on controlled synthetic benchmarks and a variant of the electronics category of Amazon Reviews '23 (Hou et al., 2024). Our experiments demonstrate that 1) the joint training procedure successfully learns to retrieve and exploit relevant historical context, 2) this improves robustness to distribution shift

[*]Equal contribution. [α] Purdue University, Elmore Family School of Electrical and Computer Engineering. Work done during Amazon internship. [β] Amazon. [γ] Georgia Institute of Technology. Correspondence to Jim Gammell: jgammell@purdue.edu, Bishal Thapaliya: bishath@amazon.com.

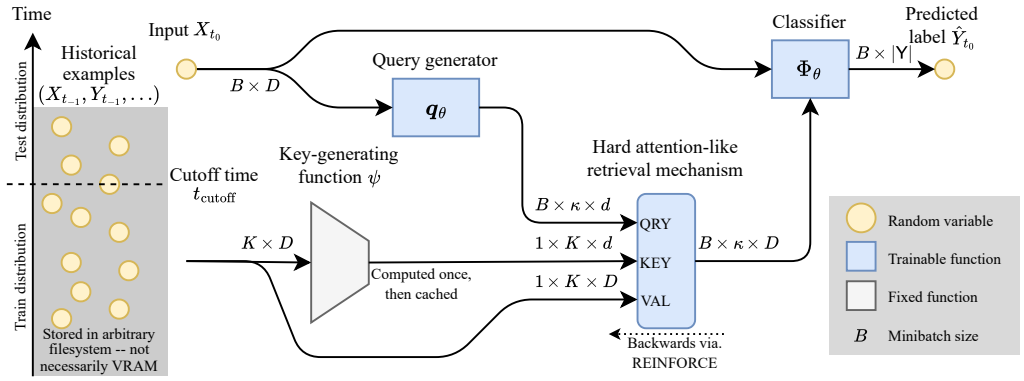

Figure 1: A diagram illustrating the architecture of our system. We learn input-dependent queries, which are used to sample from a massive corpus of historical data in a manner similar to stochastic hard attention. The input and selected corpus items are then fed to a downstream classifier which predicts the label. Through hard sampling at training time and attention masking, our implementation allows full corpus to stay on an arbitrary filesystem at both training and inference time, facilitating scaling to huge corpora.

compared to standard classifiers, and 3) we can use corpora which are too large to fit in available accelerator memory.

Related work Nonstationary classification has previously been addressed through continual learning, which incrementally updates model weights as new data arrives (Kirkpatrick et al., 2017; Wang et al., 2024), and test-time adaptation, which adapts models during inference (Liang et al., 2025) using a limited sample of data from the test distribution. Our approach is complementary: we train the model to condition on a large quantity of recent context which may be from the training or test distribution, without the cost or risk of forgetting associated with weight updates.

We build on prior work in retrieval-augmented learning (Lewis et al., 2020; Guu et al., 2020), which has proven effective for classification under distribution shift (Xu et al., 2023; Long et al., 2023; Lee et al., 2025; Fan et al., 2025) and weight update-free continual learning for LLMs (Gutierrez et al., 2025). We extend retrieval-augmented learning to nonstationary classification settings where the corpus is a growing body of labeled historical examples which are too numerous to fit in accelerator memory.

2 METHOD

See Fig. 1 for a diagram of our system, and Algorithm 1 for pseudo-code of a training step. We consider a nonstationary supervised classification setting with time t-indexed inputs $X_t \in \mathbb{R}^D$ and labels $Y_t \in \mathsf{Y}$ where Y is a finite set. These are jointly-distributed according to a time-dependent distribution: $X_t, Y_t \sim p_t$. Given $X_{t_0}, Y_{t_0} \sim p_{t_0}$, we aim to train a classifier which can predict Y_{t_0} from X_{t_0} and a series $(X_{t_{-1}}, Y_{t_{-1}}, \ldots, X_{t_{-K}}, Y_{t_{-K}})$ where $t_{-K} \leq t_{-K+1} \leq \cdots < t_0$. Our model is trained solely on data where $t_0 < t_{\text{cutoff}}$, then deployed on data where $t_0 \geq t_{\text{cutoff}}$.

We associate each historical datapoint with a low-dimensional 'key': $k_{t_{-m}} \coloneqq \psi(X_{t_{-m}}, Y_{t_{-m}})$ where $\psi : \mathbb{R}^D \times \mathsf{Y} \to \mathbb{R}^d$ is a frozen function. For example, when the inputs are text data, we may generate keys with a text embedding model (Zhang et al., 2025). We assume that $d \ll D$ is sufficiently small that we can store all K keys from our historical example sequence in VRAM. For large K there may be a significant up-front cost to computing these keys. However, because ψ is frozen, each key can be computed once and subsequently cached and reused.

Our system consists of several modules, each parameterized by weights $\boldsymbol{\theta}$. A 'query generator' $\boldsymbol{q}_\theta : \mathbb{R}^D \to \mathbb{R}^{\kappa \times d}$ generates κ input-dependent queries: $(q_1, \ldots, q_\kappa) = \boldsymbol{q}_\theta(X_{t_0})$. For each query we compute logits over the set of historical example indices $\{1, \ldots, K\}$ as $u_m \coloneqq (q_m^\top k_n / \sqrt{d} : n = 1, \ldots, K)$ (as in an attention mechanism). We then sample κ timesteps without replacement

by iteratively sampling $\hat{t}_m \sim \text{Categorical}(\text{Softmax}(u_m))$, then setting the logit for index $\hat{t}_m$ to $-\infty$ in all subsequent queries. After sampling these timesteps, we load the historical examples $(X_{\hat{t}_1}, Y_{\hat{t}_1}, \ldots, X_{\hat{t}_\kappa}, Y_{\hat{t}_\kappa})$ into accelerator memory.

Finally, we predict the label with a classifier $\Phi_\theta : \mathbb{R}^D \times (\mathbb{R}^D \times \mathsf{Y})^\kappa \to \Delta^{|\mathsf{Y}|-1}$. We train the full system end-to-end to minimize the modeled negative log-likelihood of the data:

$$\min_{\boldsymbol{\theta}} \quad \mathcal{L}(\boldsymbol{\theta}) \coloneqq -\,\mathbb{E} \log \Phi_\theta\big(Y_{t_0} \mid X_{t_0}, X_{\hat{t}_1}, Y_{\hat{t}_1}, \ldots, X_{\hat{t}_\kappa}, Y_{\hat{t}_\kappa}\big) \tag{1}$$

where $\mathbb{E}$ is taken over the data distribution and the randomness of the selection mechanism.

Backpropagation through the discrete retrieval mechanism In order to solve Eqn. 1 with stochastic gradient-based optimizers, we must backpropagate through a discrete categorical distribution. We do so using a score-based gradient estimator (Williams, 1992) with a greedy baseline (Rennie et al., 2017). This is conceptually similar to the truncated marginalization approximation of Lewis et al. (2020); Guu et al. (2020). Straight-through (Bengio et al., 2013) and relaxation (Jang et al., 2017)-based gradient estimators are more popular for use cases of this nature. However, we cannot use them here because they entail approximating the 'hard' discrete distribution with a 'soft' continuous one during training, and would thus require storing all K historical examples in accelerator memory.

Memory footprint We attain significant memory savings by storing the broader corpus of historical examples off-device. We can render the memory footprint of the keys independent from minibatch size by storing a single copy of all $\tilde{K} \in \Theta(K)$ keys in our dataset in VRAM, and setting elements of our retrieval mechanism's logits u_m to $-\infty$ where they correspond to keys not in $k_{\hat{t}_1}, \ldots, k_{\hat{t}_K}$. We can further reduce the VRAM consumption due to the logits over the corpus from $O(\kappa K)$ to $O(K)$ by computing them one at a time and freeing them after sampling corpus indices. When using a minibatch size B, this leads to a memory complexity of

$$O\Big(\underbrace{dK}_{\text{keys}} + \underbrace{BK}_{\text{logits over corpus}} + \underbrace{B\kappa D}_{\text{retrieved items}} + \underbrace{f(B, \kappa, d, D)}_{\text{model weights + activations}}\Big). \tag{2}$$

3 Experiments

We run several experiments to verify that our system is scalable and improves robustness over standard classifiers in nonstationary settings. We provide a high-level summary here and defer details to Appendix B.

Controlled synthetic settings We validate our system in two synthetic settings (Fig. 2). The first 'rotating decision boundary' setting is a binary classification problem where the optimal decision boundary smoothly rotates π radians as time varies from $t = 0$ to 1. Our system mitigates performance degradation due to this distribution shift by leveraging historical context. In the second 'needle in haystack' setting, the label is encoded in a single historical example, and our system cannot surpass random performance unless it learns to retrieve this example. Our system learns to identify and retrieve the correct example, demonstrating that we can successfully jointly train the retrieval mechanism and classifier using only the classification objective.

Sentiment classification on Amazon Electronics Reviews We evaluate scalability using subsets of the electronics category of Amazon Reviews '23 (Hou et al., 2024) (43.9M total reviews spanning 1996–2023), training on pre-2014 data. We do sentiment classification where the classification threshold changes over time (ratings > 3 positive pre-2014, > 1 positive post-2014), creating substantial distribution shift. Text features are embedded using Qwen3-Embedding-8B (Zhang et al., 2025), with truncated versions of these used as keys. Our classifier and query generator are implemented with Perceiver blocks (Jaegle et al., 2021) and trained for 100 epochs with batch size 256 using AdamW. Fig. 3 illustrates that peak VRAM during training scales affinely with the salient parameters of Eqn. 2, and that our system successfully exploits historical labels to mitigate performance degradation on the out-of-distribution post-2014 data. In Fig. 3 (right) we configure our system to retrieve only historical labels $Y_{t_{-m}}$ rather than full examples $(X_{t_{-m}}, Y_{t_{-m}})$. While

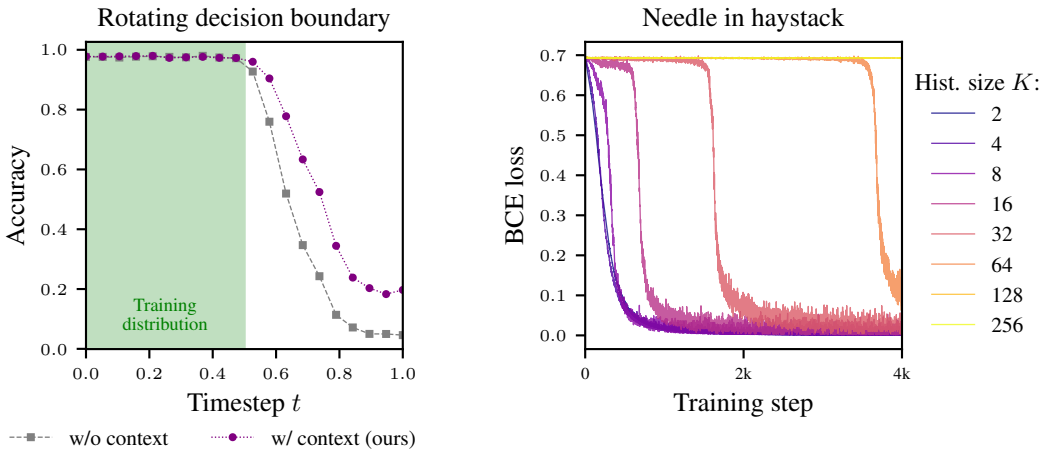

Figure 2: Synthetic settings. (**left**) Accuracy vs. time in a nonstationary binary classification setting with rotating decision boundary. Models train on $t \in [0, 0.5]$ and test on $t \in [0, 1]$. Our method leverages historical context to mitigate performance degradation beyond the training distribution. (**right**) Training dynamics on a 'needle in haystack' task where the label is encoded in one historical item. The system learns to retrieve the correct item after a random exploration phase with duration scaling linearly with the number of items.

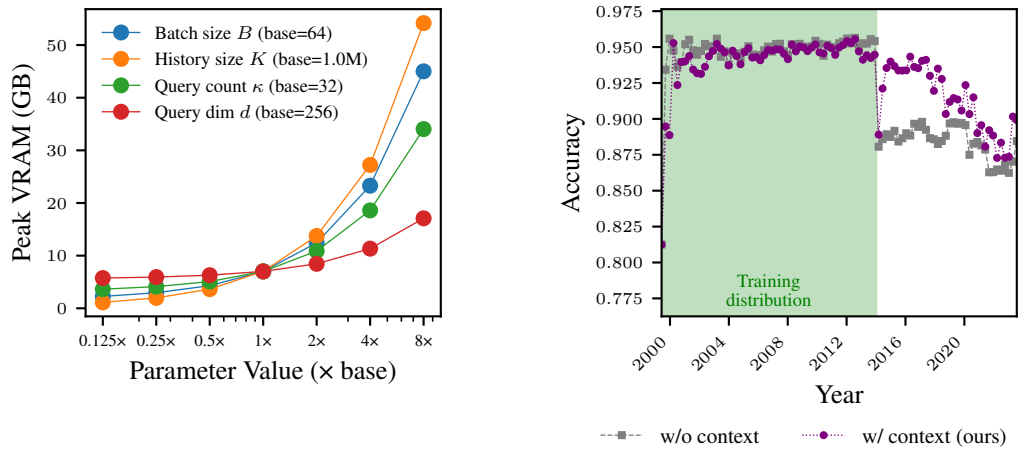

Figure 3: Amazon Electronics Reviews. (**left**) Peak VRAM consumption during training vs. hyperparameters. Scaling is consistent with Eqn. 2. (**right**) Accuracy over time for models trained on pre-2014 data. Our method leverages historical context to mitigate performance degradation on out-of-distribution data beyond 2014.

our framework supports both, we find that in this setting label-only retrieval improves robustness whereas full-feature retrieval does not (see Appendix B.3.3).

4 Limitations and Future Directions

Our work motivates several future directions for retrieval-augmented nonstationary classification: First, our system requires careful tuning to manage the high variance of the score-based gradient estimator and to prevent the classifier from ignoring historical context early in training. Exploring alternate gradient estimators, variance-reducing baselines, and regularization strategies could improve training stability. Second, our assumption of a frozen key-generating function limits retrieval expressiveness; periodic key updates would likely enhance performance. Third, while our

approach improves robustness, performance still degrades on out-of-distribution data. It could complement continual learning methods by reducing the minimum weight-update frequency to maintain performance. Finally, retrieval from large corpora unavoidably incurs computational and memory overhead relative to standard classifiers, which must be traded off with robustness improvements.

REFERENCES

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Xinqi Fan, Xueli Chen, Luoxiao Yang, Chuin Hong Yap, Rizwan Qureshi, Qi Dou, Moi Hoon Yap, and Mubarak Shah. Test-time retrieval-augmented adaptation for vision-language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8810–8819, October 2025.

Bernal Jimenez Gutierrez et al. HippoRAG 2: From RAG to memory: Non-parametric continual learning for large language models. In *International Conference on Machine Learning*, 2025.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR, 2020.

Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. Bridging language and items for retrieval and recommendation. *arXiv preprint arXiv:2403.03952*, 2024.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

Youngjun Lee, Doyoung Kim, Junhyeok Kang, Jihwan Bang, Hwanjun Song, and Jae-Gil Lee. Ra-tta: Retrieval-augmented test-time adaptation for vision-language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 133(1):31–64, 2025.

Quanyu Long, Wenya Wang, and Sinno Pan. Adapt in contexts: Retrieval-augmented domain adaptation via in-context learning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6525–6542, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.402. URL https://aclanthology.org/2023.emnlp-main.402/.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7008–7024, 2017.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5362–5383, 2024.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. doi: 10.1007/BF00992696. URL https://link.springer.com/article/10.1007/BF00992696.

Benfeng Xu, Chunxu Zhao, Wenbin Jiang, PengFei Zhu, Songtai Dai, Chao Pang, Zhuo Sun, Shuohuan Wang, and Yu Sun. Retrieval-augmented domain adaptation of language models. In Burcu Can, Maximilian Mozes, Samuel Cahyawijaya, Naomi Saphra, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Chen Zhao, Isabelle Augenstein, Anna Rogers, Kyunghyun Cho, Edward Grefenstette, and Lena Voita (eds.), *Proceedings of the 8th Workshop on Representation Learning for NLP (RepL4NLP 2023)*, pp. 54–64, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.repl4nlp-1.5. URL https://aclanthology.org/2023.repl4nlp-1.5/.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

A Algorithm Pseudocode

Algorithm 1: Pseudocode for a single training step.

Input : Initial weights $\theta_{\text{in}} \in \Theta$, input $x \in \mathsf{X}$, label $y \in \mathsf{Y}$, corpus $\boldsymbol{c} \equiv (c^{(1)}, \ldots, c^{(K)}) \in \mathsf{C}^K$
Output: Updated weights $\theta_{\text{out}} \in \Theta$

$(q^{(1)}, \ldots, q^{(\kappa)}) \leftarrow (q^{(1)}_{\theta_{\text{in}}}(x), \ldots, q^{(\kappa)}_{\theta_{\text{in}}}(x))$; // Compute query vectors
$(k^{(1)}, \ldots, k^{(K)}) \leftarrow (\psi(c^{(1)}), \ldots, \psi(c^{(K)}))$; // Compute corpus keys (once, then cache + re-use)

// Sample from corpus based on learned notion of relevance:
for $\alpha = 1, \ldots, \kappa$ **do**
  $u^{(\alpha)} \leftarrow \frac{1}{\sqrt{d}} \left(q^{(\alpha)\top} k^{(\beta)} : \beta = 1, \ldots, K\right)$; // Scaled query-key dot products
  $u^{(\alpha, i^{(1:\alpha-1)})} \leftarrow -\infty$; // Enforce sampling without replacement
  $p^{(\alpha)} \leftarrow \text{Softmax}(u^{(\alpha)})$; // Per-item probability of selection
  $i_{\text{s}}^{(\alpha)} \sim \text{Cat}(p^{(\alpha)})$; // Stochastically sample corpus index
  $i_{\text{g}}^{(\alpha)} \leftarrow \text{argmax}_{\beta=1,\ldots,K}\, u^{(\alpha,\beta)}$; // Greedily select highest-mass index
  $c_{\text{s}}^{(\alpha)} \leftarrow \text{ToVRAM}(c^{(i_{\text{s}}^{(\alpha)})})$, $c_{\text{g}}^{(\alpha)} \leftarrow \text{ToVRAM}(c^{(i_{\text{g}}^{(\alpha)})})$; // Load selected items

$l_{\text{s}} \leftarrow \log \Phi_{\theta_{\text{in}}}\left(y \mid x, c_{\text{s}}^{(1)}, \ldots, c_{\text{s}}^{(\kappa)}\right)$, $l_{\text{g}} \leftarrow \log \Phi_{\theta_{\text{in}}}\left(y \mid x, c_{\text{g}}^{(1)}, \ldots, c_{\text{g}}^{(\kappa)}\right)$; // Classifier cross-entropy loss
$l_{\text{rfc}} \leftarrow -\sum_{\alpha=1}^{\kappa} \log p^{(\alpha, i_{\text{s}}^{(\alpha)})}\, \text{StopGrad}(l_{\text{s}} - l_{\text{g}})$; // REINFORCE pseudo-loss
$g \leftarrow \text{AutoDiff}_{\theta_{\text{in}}}(l_{\text{g}} + l_{\text{rfc}})$; // Unbiased estimate of $\nabla\mathcal{L}(\theta)$
$\theta_{\text{out}} \leftarrow \text{OptimizerStep}(\theta_{\text{in}}, g)$; // Update weights

return θ_{out}

In Algorithm 1 we provide pseudocode for a single training step of our method. For clarity we omit details of minibatch use and additional implementation details which are described below.

B Experimental Details

B.1 Needle in Haystack

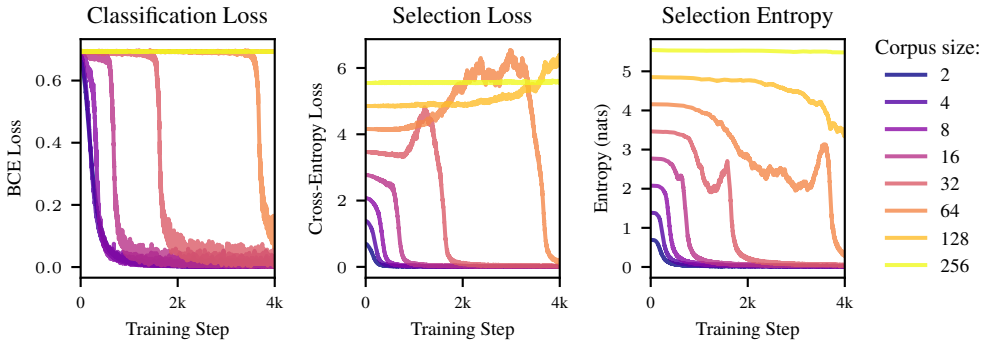

Figure 4: Additional results in the 'needle in haystack' setting. (**left**) Binary cross-entropy classification loss vs. training steps. Loss remains random for some exploration period which scales linearly with the number of historical examples, after which it sharply decreases as the query mechanism learns to retrieve the correct item. (**center**) Cross-entropy loss of the *retrieval mechanism* over time, when viewing the index of the key-containing historical item as the 'true label'. This loss decreases at around the same point at which the classification loss decreases. Note that we do not directly train on this objective – the retrieval mechanism learns to select the correct item because this leads to lower classification loss. (**right**) Entropy of the distribution over historical examples from which the retrieval mechanism samples. This sharply increases and then decreases at around the time the classification loss decreases.

Note that the framing of our method in Sec. 2 can be generalized to cases where the historical data is sampled from a different distribution than (X_{t_0}, Y_{t_0}). Here we aim to construct our setting where our retrieval mechanism must select a particular item from history for the system to achieve nontrivial performance. Towards this end, we relax our setting and allow the sequence of historical items $(X_{t_{-m}}, Y_{t_{-m}}) \to U_{t_{-m}}$ to come from a different distribution.

We generate our dataset in the following manner: we first choose a history size $K \in \mathbb{Z}_{++}$. We then randomly sample a label $Y_{t_0} \sim \text{Uniform}\{0, 1\}$ and a historical example index $m^* \in \text{Uniform}\{1, \ldots, K\}$. We let each historical item consist of bitstrings $U_{t_{-m}} = (U^{(1)}_{t_{-m}}, \ldots, U^{(D')}_{t_{-m}})$ for $m = 1, \ldots, K$, where $U^{(1)}_{t_{-m^*}} = Y_{t_0}$ and $U^{(n)}_{t_{-m}} \sim \text{Uniform}\{0, 1\}$ for all $(m, n) \neq (m^*, 1)$. For each historical item we assign a key $\psi(U_{t_{-m}}) \sim \text{Normal}(0, 1)^d$. We let X_{t_0} be a noisy and linearly-transformed version of this key: $X_{t_0} \coloneqq T\psi(U_{t_{-m}}) + \epsilon$ where $T \sim \text{Normal}(0, \frac{1}{d})^{d \times d}$ is a fixed linear transform and $\epsilon \sim \text{Normal}(0, \sigma^2_{\text{obs}})^d$ is random noise. Our system can thereby predict Y by predicting $\psi(U_{t_{-m^*}})$ from X_{t_0}, retrieving the historical item whose key has maximum cosine similarity with this prediction, then predicting Y_{t_0} equal to its first bit.

In our experiments we sweep $K \in \{2, 4, 8, \ldots, 256\}$, and set $D' = 8$, $d = 64$, and $\sigma_{\text{obs}} = \sqrt{0.1}$. We train our models for 4k steps with minibatch size 1k, using the AdamW optimizer with `lr=2e-4` and other hyperparameters left at their default PyTorch values. Our query generator and classifier are implemented with ReLU MLP blocks, each with a single 512-width hidden layer, and both are fed by a shared 1-layer ReLU MLP input stage. Our system retrieves $\kappa = 1$ historical items.

Additional results are shown in Fig. 4. The classification loss is characterized by an exploration phase where the classifier achieves random performance, before its loss sharply decreases. The length of the exploration phase is roughly linear in K. This is consistent with the facts that 1) the classifier can only achieve random performance unless the retrieval mechanism returns the correct item, and 2) the retrieval mechanism has no learning signal unless the classifier has learned to exploit the correct item. Thus, early on learning can only happen when the retrieval mechanism selects the correct item by random chance.

Interestingly, for larger values of K, the selection loss increases during the random exploration phase, suggesting that the mechanism is assigning increasing probability mass to an incorrect subset of historical items. During this time, the selection entropy decreases. At the end of the random exploration phase, the selection entropy sharply increases before decreasing again, suggesting that the retrieval mechanism is unlearning to select the incorrect items before learning to select the correct ones. This phenomenon merits further exploration and likely slows down learning by reducing the rate at which the correct item is retrieved during exploration.

B.2 Rotating Decision Boundary

Here we define a binary classification problem where each class is sampled from a distinct Gaussian distribution. The centers of these distributions rotate smoothly around an axis as we vary a time parameter, so that for any given time the optimal decision boundary is a hyperplane, and this hyperplane rotates by π radians over the interval of times.

To generate this dataset, we randomly sample an orthonormal pair of vectors $\theta_0 = \theta_0'/\|\theta_0'\|_2$ where $\theta_0' \sim \mathrm{Normal}(0,1)^D$, and $\theta_1 = \theta_1'/\|\theta_1'\|_2$ where $\theta_1' = \theta_1'' - ((\theta_1'')^\top \theta_0)\theta_0$ and $\theta_1'' \sim \mathrm{Normal}(0,1)^D$. We then define the rotating class-separation vector $\Delta(t) = \delta(\theta_0 \cos(\pi\tau) + \theta_1 \sin(\pi\tau))$ where $\tau = 0.5\cos(\pi t)+0.5$, for $t \in [0,1]$, where $\delta > 0$. For each time t we sample data by first sampling a label $Y_{t_0} \sim \mathrm{Uniform}\{0,1\}$, then covariates $X_{t_0}' = \epsilon + \frac{1}{2}\Delta(t_0)(2Y-1)$ where $\epsilon \sim \mathrm{Normal}(0, \sigma_{\mathrm{obs}})^D$. We also concatenate a sinusoidal time embedding to the input, as in this setting our system struggles to retrieve appropriate historical items otherwise. Thus, we have $X_{t_0} = (X_{t_0}', \mathrm{embed}(t_0))$. We let $\psi(X_{t_0}, Y_{t_0}) = X_{t_0}$.

For training samples we let $t_0 \in [0, 0.5]$, and for test samples we let $t_0 \in [0,1]$. We feed our system sequences of K ordered pairs $(X_{t_{-m}}, Y_{t_{-m}})$ where each $t_{-m} \sim \mathrm{Uniform}[0, t_0)$. We set $\delta = 4$, $\sigma_{\mathrm{obs}} = 1$, $D = 64$, 8 Fourier time embedding frequencies, history size $K = 128$, and let our system retrieve $\kappa = 16$ items. Models are trained for 2500 steps with minibatch size 4096, using the AdamW optimizer with `lr=5e-5` and other hyperparameters left at their default PyTorch values. Our query generator and classifier are implemented with ReLU MLP blocks, each with 2 512-width hidden layers, and both are fed by a shared 1-layer ReLU MLP input stage. Our 'no context' baseline is a ReLU MLP with 3 512-width hidden layers.

B.3 Amazon Electronics Reviews

B.3.1 Dataset

We use the electronics category of Amazon Reviews '23 (Hou et al., 2024), which consists of 43.9M reviews collected from 1998–2023. We sample our training and validation sets from the 4.0M reviews posted before 2014, with test sets sampled from 100 bins uniformly spaced in time from 1998–2023. We use 100k examples as training data which our model learns to predict, and a disjoint set of 1.9M examples as a corpus of historical items to use during training. Our validation set consists of 20k examples which the model trains to predict, and a disjoint 1.9M examples as a corpus of historical items. We generate test data by creating 100 uniformly-spaced time bins and sampling up to 10k examples from each (less for the early time bins which contain <10k reviews). For each test set, our historical data comes from a sliding window of the past 1.9M examples. When classifying X_{t_0}, we use attention masking to constrain the model to only retrieve historical items from times less than t_0. The historical example corpora are stored in the filesystem as a `np.memmap` object, with retrieved items loaded into VRAM during the forward pass.

Each review consists of 6 fields: `title`, `text`, `helpful_vote`, `verified_purchase`, and `timestamp`. We embed the `title` and `text` fields to 4096-dimensional vectors using Qwen3-Embedding-8B (Zhang et al., 2025), using the task 'Embed the Amazon review for sentiment classification.' and generating the prompt with the Python format string `f'<title>{review["title"]}</title><text>{review["text"]}</text>'`.
We use `helpful_vote` and `verified_purchase` as covariates for our classifier, and generate binary classification labels of 1 (positive sentiment) when `rating` is above a threshold, and 0 (negative sentiment) when equal or below. While the dataset spans a long duration of time, we do not observe significant performance degradation for classifiers trained on pre-2014 data and evaluated on our test sets, possibly due to the text embedding model being trained on data beyond

2023. Thus, to increase the distribution shift, we set our positive sentiment threshold to 3 for pre-2014 reviews and 1 for post-2014 reviews.

B.3.2 Architecture and Implementation Details

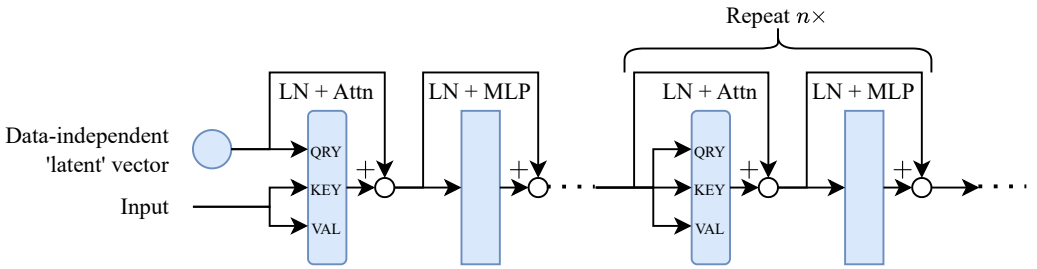

Figure 5: Diagram of a Perceiver block (Jaegle et al., 2021). Rather than attending directly to the full input sequence like a standard transformer, Perceivers use cross-attention to combine the input with a learned data-independent 'latent' vector. They then apply a sequence of standard transformer blocks to this sequence. This approach reduces the computational cost from L^2 to LM where L is the input sequence length and M is the latent sequence length (a hyperparameter). The Perceiver architecture is appealing for our use case because it was proposed with multimodal feature fusion in mind, and because it is more-scalable to long sequences than standard transformers.

We implement the query generator and classifier of our system using Perceiver blocks (Jaegle et al., 2021) (illustrated in Fig. 5). We prefer these over standard transformer blocks because they were proposed with multimodal data in mind, and have better scalability to long input sequences. However, our method is architecture-agnostic. We use an embedding dimension of 512, a latent sequence length of 128, and stages consisting of 1 cross-attention block followed by 2 self-attention blocks. We retrieve $\kappa = 4$ historical items.

Corpus keys are generated by taking the first 64 dimensions of the embedded text features. Note that the Qwen3-Embedding models use Matryoshka representation learning (Kusupati et al., 2022), which is designed to facilitate dimensionality reduction through truncation in this manner.

We preprocess text embeddings by standardizing them, then linearly projecting them to the embedding dimension of our Perceiver blocks. For the `helpful_votes` feature, we standardize it, apply a Fourier embedding with 48 frequency bands and a maximum frequency of 64, then linearly project it to the embedding dimension. For the `verified_purchase` feature, we simply linearly project it to the embedding dimension.

We apply the following bag of tricks, which were found helpful in preliminary experiments on similar datasets:

- We find that pure cosine similarity between the embeddings of the text feature of X_{t_0} and those of the historical items performs quite well. Thus, instead of training our retrieval mechanism from scratch, we train it to learn the residual with this cosine similarity: we parameterize it as $q_\theta(x) = \alpha\tilde{q}_\theta(x) + (1-\alpha)q_{\text{sim}}(x)$, where $\alpha \in (0,1)$ is learned and q_{sim} simply returns the first 64 dimensions of the text embedding.
- We randomly drop out the direct input $\to$ classifier connection with probability `cls_stage_dropout_rate = 0.9` (similarly to stochastic depth (Huang et al., 2016)), forcing the model to rely solely on the retrieved items for classification. This is helpful because it counteracts a tendency for the classifier to learn to ignore retrieved items early in training when they are not relevant, thereby depriving the retrieval mechanism of a learning signal.
- We randomly drop out some of the retrieved items with rate `query_dropout_rate = 0.1`.
- We use a $10\times$ higher learning rate for the parameters of the retrieval mechanism than for the rest of the system.

- Over the course of training we exponentially decay the temperature of the retrieval mechanism's softmax distribution from a starting temperature of `max_temperature = 0.01` to a final temperature of `min_temperature = 0.001`. This is essentially a form of curriculum learning, where early in training we have low-variance gradients which are easy to train with, and late in training we have higher-variance but retrieval mechanism behavior closer to the zero-temperature limit used at test time.
- We use AdamW-style weight decay with strength 10^{-4}, apply gradient norm clipping with maximum norm 1, and a linear learning rate warmup for the first 10% of epochs, followed by cosine decay down to $0.1\times$ the base learning rate for the remaining epochs.

We also explored the following tricks, but found them unhelpful in this setting:

- In order to prevent premature collapse of the retrieval mechanism to a bad solution, we add a term proportional to the negative entropy of the retrieval mechanism's softmax distribution to the loss, with strength which decreases over the course of training.
- In order to encourage the κ queries to retrieve diverse historical examples rather than selecting based on similar criteria, we add a term proportional to the mean probability mass assigned by a given query to the items selected by different queries.
- In order to bias the retrieval mechanism towards selecting based on the cosine similarity between the first 64 dimensions of the text embeddings of X_{t_0} and those of the historical items, we add a term proportional to the cosine similarity between our query vectors and the first 64 text embedding dimensions of X_{t_0}.

We train our system for 100 epochs with minibatch size 1024 using the AdamW optimizer with base learning rate `lr=1e-3`.

B.3.3 Impact of Retrieving Full Historical Examples

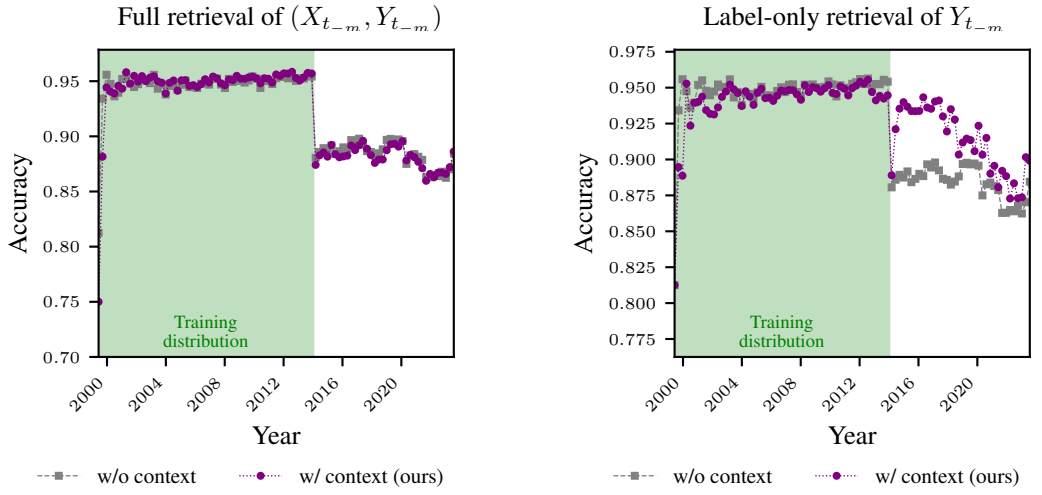

Figure 6: Accuracy over time for models trained on pre-2014 data from the electronics category of Amazon Reviews '23. (**left**) Our retrieval mechanism returns full historical examples $(X_{t_{-m}}, Y_{t_{-m}})$. (**right**) Our retrieval mechanism returns only the labels $Y_{t_{-m}}$ of historical examples. In the former case our system has similar robustness to a standard classifier, while in the latter case it is able to leverage historical context to mitigate performance degradation on the out-of-distribution data beyond 2014.

As shown in Fig. 6, retrieving full historical examples $(X_{t_{-m}}, Y_{t_{-m}})$ does not improve robustness over the baseline. However, retrieving only the labels $Y_{t_{-m}}$ does improve performance. This suggests that in this particular setting, the full features of historical examples have little utility for classifying X_{t_0}, and drown out the useful signal provided by the historical labels with noise or alternate

non-generalizing signals. This indicates the need for better regularization or architectural strategies to enable the classifier to leverage both historical features and labels.

C Derivation of Score-Based Gradient Estimator and Application to Selection Mechanism

The derivations below are not novel, but for completeness and ease of reading we include them here in our own notation. For brevity we omit engineering details such as the baseline used to reduce the variance of the gradient estimator.

Note that this gradient estimator is sometimes referred to as 'REINFORCE', and is a building block for many reinforcement learning algorithms. Our setup can be viewed as episodic reinforcement learning with episode length 1, the action being the selection of items from the corpus, and the reward given by the negative loss of the downstream classifier.

C.1 Gradient Estimation in the Normal Deep Learning Context

For context, we will first discuss gradient estimation in the normal supervised deep learning setting. Suppose we have a space of inputs $\mathsf{X} \subset \mathbb{R}^D$ and outputs $\mathsf{Y} \subset \mathbb{R}^M$. We view inputs and outputs as jointly-distributed random variables $X \in \mathsf{X}$ and $Y \in \mathsf{Y}$ with $(X, Y) \sim p$ for some data-generating distribution p. We aim to train a classifier $\Phi_\theta : \mathsf{X} \to \hat{\mathsf{Y}}$ parameterized by weights $\theta \in \Theta \subset \mathbb{R}^P$, where $\hat{\mathsf{Y}} \subset \mathbb{R}^{M'}$ is a space of predicted outputs which can be mapped to outputs in Y. The classifier is trained to minimize some loss function $\ell : \Theta \times \mathsf{X} \times \mathsf{Y} \to \mathbb{R}$ in expectation over the data distribution:

$$\min_{\theta \in \Theta} \quad \mathcal{L}(\theta) \coloneqq \mathbb{E}_{X,Y \sim p}\, \ell(\theta; X, Y). \tag{3}$$

For example, when training an ImageNet classifier, $\mathsf{X} \subset \mathbb{R}^{3 \cdot 224 \cdot 224}$ might represent the space of images, $\mathsf{Y} \equiv \{1, \ldots, 1000\}$ the space of labels, $\hat{\mathsf{Y}} \equiv \Delta^{999}$ the probability simplex over 1000 classes, and $\ell(\theta; x, y) \coloneqq -\log \Phi_\theta(y \mid x)$ the cross-entropy loss of the classifier.

We approximately solve this optimization problem using stochastic gradient descent. Note that because p does not depend on θ, we can exchange the order of differentiation and expectation:

$$\nabla \mathcal{L}(\theta) = \nabla_\theta\, \mathbb{E}_{X,Y \sim p}\, \ell(\theta; X, Y) = \mathbb{E}_{X,Y \sim p}\, \nabla_\theta \ell(\theta; X, Y). \tag{4}$$

While analytically integrating over the data distribution p is intractable, we can use a finite number of datapoints $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)}) \overset{\text{i.i.d.}}{\sim} p$ to approximate the expectation:

$$\nabla \mathcal{L}(\theta) = \mathbb{E}_{X,Y \sim p}\, \nabla_\theta \ell(\theta; X, Y) \approx \frac{1}{N} \sum_{\alpha=1}^{N} \nabla_\theta \ell(\theta; x^{(\alpha)}, y^{(\alpha)}). \tag{5}$$

Each term in the summation is now straightforward to compute with automatic differentiation (Paszke et al., 2019).

C.2 Gradient Estimation when our Model contains a Discrete Stochastic Layer

Now suppose Φ_θ contains a discrete stochastic layer. We will denote by $\phi_\theta : \mathsf{X} \to \Delta^{N-1}$ a deterministic function in our model which generates an X-dependent random variable U such that $U \mid X \sim p_\theta(\cdot \mid X) \coloneqq \mathrm{Cat}(\phi_\theta(X))$, and $\psi_\theta : \mathsf{X} \times \{1, \ldots, N\} \to \hat{\mathsf{Y}}$ a downstream function which predicts the output based on X and a sample from $p_\theta(\cdot \mid X)$. Now $\Phi_\theta(X)$ is a random variable $\psi_\theta(X, U)$ where $U \sim p_\theta(\cdot \mid X)$. We let our loss function depend on the sample from our stochastic layer, $\ell : \Theta \times \mathsf{X} \times \mathsf{Y} \times \{1, \ldots, N\} \to \mathbb{R}$, and our objective now contains an expectation over both the data distribution and the randomness of our model:

$$\mathcal{L}(\theta) \coloneqq \mathbb{E}_{X,Y \sim p,\, U \sim p_\theta(\cdot \mid X)}\, \ell(\theta; X, Y, U). \tag{6}$$

In the ImageNet example, we would have $\ell(\theta; x, y, u) = -\log \psi_\theta(y \mid x, u)$.

Here both expectations are intractable or impractical to compute analytically. However, we cannot simply exchange the order of differentiation and expectation with respect to the second integral the way we did above, i.e.

$$\nabla \mathcal{L}(\theta) = \nabla_\theta \, \mathbb{E}_{X,Y\sim p,\, U\sim p_\theta(\cdot|X)} \, \ell(\theta; X, Y, U) \tag{7}$$

$$= \mathbb{E}_{X,Y\sim p} \, \nabla_\theta \, \mathbb{E}_{U\sim p_\theta(\cdot|X)} \, \ell(\theta; X, Y, U) \tag{8}$$

$$\neq \mathbb{E}_{X,Y\sim p,\, U\sim p_\theta(\cdot|X)} \, \nabla_\theta \ell(\theta; X, Y, U), \tag{9}$$

because $p_\theta(\cdot \mid X)$ depends on θ. Instead, we use the REINFORCE trick:

$$\nabla_\theta \, \mathbb{E}_{U\sim p_\theta(\cdot|X)} \, \ell(\theta; X, Y, U) \tag{10}$$

$$= \nabla_\theta \sum_{u=1}^{N} p_\theta(u \mid X) \ell(\theta; X, Y, u) \tag{11}$$

$$= \sum_{u=1}^{N} \Big[\nabla_\theta p_\theta(u \mid X) \ell(\theta; X, Y, u) + p_\theta(u \mid X) \nabla_\theta \ell(\theta; X, Y, u) \Big] \tag{12}$$

$$= \sum_{u=1}^{N} \Big[p_\theta(u \mid X) \nabla_\theta \log p_\theta(u \mid X) \ell(\theta; X, Y, u) + p_\theta(u \mid X) \nabla_\theta \ell(\theta; X, Y, u) \Big] \tag{13}$$

$$= \mathbb{E}_{U\sim p_\theta(\cdot|U)} \Big[\nabla_\theta \log p_\theta(u \mid X) \ell(\theta; X, Y, u) + \nabla_\theta \ell(\theta; X, Y, u) \Big]. \tag{14}$$

This expression allows us to approximate the expectation with a finite number of datapoints $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)}) \overset{\text{i.i.d.}}{\sim} p$ and samples from our stochastic layer $u^{(1)} \sim p_\theta(\cdot \mid x^{(1)}), \ldots, u^{(N)} \sim p_\theta(\cdot \mid x^{(N)})$, similarly to above:

$$\nabla \mathcal{L}(\theta) = \mathbb{E}_{X,Y\sim p,\, U\sim p_\theta(\cdot|X)} \Big[\nabla_\theta \log p_\theta(U \mid X) \ell(\theta; X, Y, U) + \nabla_\theta \ell(\theta; X, Y, U) \Big] \tag{15}$$

$$\approx \frac{1}{N} \sum_{\alpha=1}^{N} \Big[\nabla_\theta \log p_\theta(u^{(\alpha)} \mid x^{(\alpha)}) \ell(\theta; x^{(\alpha)}, y^{(\alpha)}, u^{(\alpha)}) + \nabla_\theta \ell(\theta; x^{(\alpha)}, y^{(\alpha)}, u^{(\alpha)}) \Big]. \tag{16}$$

We can compute the quantities $\nabla_\theta \log p_\theta(u \mid X) = \nabla_\theta \log \phi_\theta(u \mid X)$ with automatic differentiation; thus, Eqn. 16 enables us to train our model with stochastic gradient descent. In PyTorch, this can be implemented by backpropagating through the pseudo-loss function

$$\ell_{\text{pseudo}}(\theta; x, y, u) = \log p_\theta(u \mid x) \, \text{StopGrad}(\ell(\theta; x, y, u)) + \ell(\theta; x, y, u) \tag{17}$$

$$\text{where} \quad u = \text{StopGrad}(\tilde{u}),\ \tilde{u} \sim p_\theta(\cdot \mid x). \tag{18}$$

C.3 Application to our Retrieval Mechanism

Our method fits into the above framework as follows: we let the tuple $X \leftarrow (X_{t_0}, X_{t_{-1}}, Y_{t_{-1}}, X_{t_{-2}}, Y_{t_{-2}}, \ldots, X_{t_{-K}}, Y_{t_{-K}})$ be the input and $Y \leftarrow Y_{t_0}$ be the output. Our stochastic layer is represented by the retrieval mechanism, which we will denote $p_\theta(X)$, which gives the distribution over the historical example indices $U \leftarrow \boldsymbol{I} \equiv (I^{(1)}, \ldots, I^{(\kappa)})$ where each $I^{(m)} \in \{1, \ldots, K\}$. Note that $\boldsymbol{I}$ fits into the above framing because it is a bijection of a categorical random scalar over $N \equiv \prod_{m=1}^{\kappa} (K - m + 1)$ categories. The downstream function is then given by $\psi_\theta(X, U) \leftarrow \Phi_\theta(Y_{t_0} \mid X_{t_0}, X_{\hat{t}_1}, Y_{\hat{t}_1}, \ldots, X_{\hat{t}_\kappa}, Y_{\hat{t}_\kappa})$.

Thus, by sampling a finite number of inputs and historical sequences $(x_{t_0}, y_{t_0}, x_{t_{-1}}, y_{t_{-1}}, \ldots, x_{t_{-K}}, y_{t_{-K}})$ from the data distribution, we can perform stochastic gradient descent by leveraging Eqn. 16.