Probabilistic Hash Embeddings for Online Learning of Categorical Features

Aodong Li, Abishek Sankararaman, Balakrishnan Murali Narayanaswamy

Amazon Web Services
aodongli@amazon.com, abisanka@amazon.com, muralibn@amazon.com

Abstract

We study streaming data with categorical features where the vocabulary of categorical feature values is changing and can even grow unboundedly over time. Feature hashing is commonly used as a pre-processing step to map these categorical values into a feature space of fixed size before learning their embeddings. While these methods have been developed and evaluated for offline or batch settings, in this paper we consider online settings. We show that deterministic embeddings are sensitive to the arrival order of categories and suffer from forgetting in online learning, leading to performance deterioration. To mitigate this issue, we propose a probabilistic hash embedding (PHE) model that treats hash embeddings as stochastic and applies Bayesian online learning to learn incrementally from data. Based on the structure of PHE, we derive a scalable inference algorithm to learn model parameters and infer/update the posteriors of hash embeddings and other latent variables. Our algorithm (i) can handle an evolving vocabulary of categorical items, (ii) is adaptive to new items without forgetting old items, (iii) is implementable with a bounded set of parameters that does not grow with the number of distinct observed values on the stream, and (iv) is invariant to the item arrival order. Experiments in classification, sequence modeling, and recommendation systems in online learning setups demonstrate the superior performance of PHE while maintaining high memory efficiency (consumes as low as 2∼4% memory of a one-hot embedding table).

Code — https://github.com/aodongli/probabilistic-hash-embeddings

Extended version — https://arxiv.org/abs/2511.20893

1 Introduction

Categorical features occur in many high-value ML applications: finance (Clements et al. 2020), fraud detection (Al-Hashedi and Magalingam 2021), anomaly detection (Han et al. 2022), cybersecurity (Sarker et al. 2020), medical diagnosis (Shehab et al. 2022), recommendation systems (Ko et al. 2022; Lai et al. 2023) etc. Large vocabularies and embedding tables are strong characteristics of these categorical feature-intensive applications. While assigning each item[1] its own row in a large embedding table typically improves accuracy, this comes at a cost. Larger embedding tables require more resources and memory to deploy and slow down execution.

A common and well-established solution for learning categorical features at scale, without maintaining a large vocabulary and embedding table, is the use of hashing techniques (Weinberger et al. 2009; Tito Svenstrup, Hansen, and Winther 2017; Shi et al. 2020b; Kang et al. 2021; Lai et al. 2023; Coleman et al. 2024). For any categorical value (typically a string), a hash function maps it to an index in a small, fixed range. This index addresses a row in a much smaller embedding table. These rows, referred to as *hash embeddings*, are used in subsequent model training and inference.

Hash collisions–when different items map to the same hashed value and thus share an embedding–can degrade model performance. This issue can be mitigated in two ways: first, by designing sophisticated operations on the hashed values (Weinberger et al. 2009); second, by using multiple hash functions (Tito Svenstrup, Hansen, and Winther 2017; Coleman et al. 2024). Large technology firms like Yahoo and Google have successfully incorporated this approach in their applications (Weinberger et al. 2009; Coleman et al. 2024).

A major limitation of prior works is they focused on *offline* settings where data distributions are stationary and the entire test set is available in batch. In other words, the vocabulary of categorical items is fixed with no new or out-of-vocabulary items occurring during testing.

In many real applications, data arrives in a streaming fashion: *(i)* the vocabulary of categorical items can change, and/or *(ii)* the meaning of an item can evolve. These characteristics present challenges to offline predictive models. Failure to adapt to the expanding vocabulary leads to a loss in predictive performance, as shown in Fig. 1–in streaming binary classifications, new groups or sets of categorical items occur sequentially, and modeling these new items incrementally leads to significant accuracy improvement. This problem of vocabulary expansion is fairly common in practice: new products are added to grocery stores (Cheng et al. 2023), new usernames and application names appear in intrusion detection systems (Siadati and Memon 2017; Le et al. 2022), new patients arrive at hospitals, and so on.

More critically, as our analysis and experiments show (Secs. 3.4 and 4), naively adapting to new items while using hashing techniques suffers from *catastrophic forgetting* (Kirkpatrick et al. 2017). In hash embeddings, different items

[1] We use the term *item* to refer to a categorical value.

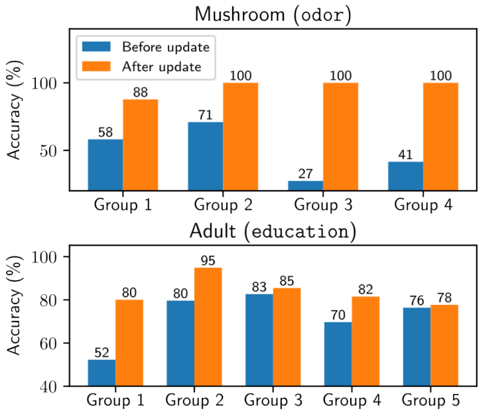

Figure 1: On two tabular datasets, Mushroom and Adult, we split the data into groups based on random partitions of categorical column vocabularies, such that each group has disjoint vocabulary sets. We report the results before and after online learning on each group. The performance gaps motivate the need to learn representations of new items. Bracketed values indicate the splitting columns. Results are averaged over five runs. Partition details are in Fig. 9.

may share embeddings, updating one can adversely affect another, causing the model to effectively "forget." Furthermore, this forgetting can be exacerbated depending on the arrival order. Consequently, hash embeddings are not yet fit for online learning in their vanilla form.

In this paper, we employ Bayesian online learning to mitigate the forgetting issue. This approach is theoretically shown to be as effective as offline batch learning, regardless of the order in which items arrive (Opper and Winther 1999). Specifically, we model hash embeddings as random variables and perform posterior inference upon observing new data. This probabilistic treatment enables continual adaptation to new items while preserving knowledge of previously seen ones.

Main Contributions: Our work proposes *probabilistic hash embeddings* (PHE) with Bayesian updates to handle dynamic vocabularies effectively and efficiently. The intuition behind PHE stems from its benefits in *(i)* efficiency, as memory and the number of model parameters is bounded, with only a small number of parameters requiring online updates (ie., less forgetting), and *(ii)* accuracy, as the Bayesian treatment provides implicit regularization that balances forgetting and adaptation while maintaining invariance to item arrival order, without dataset-specific regularization design.

We highlight PHE as a plug-in module, which can be applied to other probabilistic models such as Deep Kalman Filters (Krishnan, Shalit, and Sontag 2015) and Neural Collaborative Filtering (He et al. 2017). The usage of PHE allows these models to handle unbounded items in their domains in a principled way. We derive scalable variational inference algorithms for learning PHE and provide theoretical analysis demonstrating PHE's superiority for online learning. Empirically, our method outperforms baselines in three setups: online supervised learning with new items, conditional sequence modeling with growing vocabularies, and a recommendation system with evolving user-item interactions.

Organization: We survey related work in Sec. 2, present PHE, derive its inference algorithms in Sec. 3, demonstrate PHE's efficacy in Sec. 4, and conclude in Sec. 5. More details and limitation discussions are in supplementary materials.

2 Related Work

Hashing trick. Weinberger et al. (2009) first proposed using hashing to handle an unbounded number of categorical items. To mitigate degradation due to hash collisions, Serrà and Karatzoglou (2017) used bloom filters. More recently, Tito Svenstrup, Hansen, and Winther (2017); Cheng et al. (2023); Coleman et al. (2024) proposes shared embeddings across all categorical features for efficiency and multiple hashing functions to reduce collisions. However, unlike our method, these approaches are deterministic and are developed in offline settings. As shown in our experiments, deterministic hashing embeddings are vulnerable to evolving vocabularies.

Continual learning. Previous continual learning methods (Kirkpatrick et al. 2017; Nguyen et al. 2018; Lopez-Paz and Ranzato 2017) were designed for continuous-valued features and do not address how to learn evolving categorical features. This only existing approach, architecture-expanding methods that dynamically expand the embedding tables (see EE in experiments), leads to unbounded memory usage (Rusu et al. 2016; Yoon et al. 2017; Jerfel et al. 2019). In contrast, PHE is the first to maintain constant memory while preserving accuracy and invariance to category arrival order.

Temporal and recommendation models. Temporal and recommendation models are important applications for categorical feature embeddings. We extend Deep Kalman Filters (DKF) (Krishnan, Shalit, and Sontag 2015) to handle evolving vocabularies, whereas the original DKF assumes a fixed vocabulary. Similarly, previous recommendation methods (Ko et al. 2022; Shi et al. 2020b; Kang et al. 2021; Coleman et al. 2024) assume training data is given in batch and the item vocabulary remains fixed.

Tabular data models. Categorical features are ubiquitous in tabular datasets. In online and continual learning settings, deep learning-based methods have been studied in recent years (Huang et al. 2020; Du et al. 2021; Liu, Di, and Chen 2023). However, all of these works assume the vocabulary of categorical items is known and fixed in advance. *Ours is the first online learning method for tabular data that can handle increasing and unbounded items.* While Kim, Grinsztajn, and Varoquaux (2024) use string embeddings from language models for open-vocabulary categorical features in an offline setting, we focus on online settings. Moreover, language model representations cannot be exploited when strings lack semantic meaning (such as anonymized ZIP codes, IP addresses, etc.). We survey additional related work in Supp. D.

3 Methodology

We first establish the necessary notations and introduce our proposed probabilistic hash embedding (PHE) module. Next, we derive an online inference algorithm for PHE. Finally, we analyze and explain why PHE is superior in online learning.

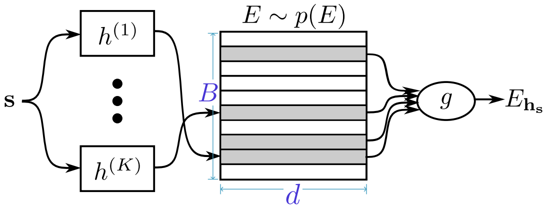

Figure 2: PHE architecture. The input $\mathbf{s}$ can be usernames or anonymized strings. The entire module represents $p(E_{\mathbf{h}_\mathbf{s}})$.

3.1 Notations and Problem Setup

We consider the problem of learning predictive ML models with categorical features in an online learning environment. We assume the model input and output dimensions are fixed but new categorical features may occur over time.[2]

We denote categorical items by $s \in \mathcal{S}$. Typically, practitioners aim to predict certain target features using ML models based on other input features. We denote these target features by y. Depending on the task, y can be either numerical or categorical. We use subscript i to denote the i-th data point.

Hashing techniques use hash functions (e.g., MD5) to map categorical items to hashed values. These hashed values are then used to index rows of an embedding table. The range of hash values is usually much smaller than the vocabulary size of the categorical items, thus achieving memory and computational efficiency. We define a hash function $h : \mathcal{S} \to \mathbb{N}_{<B}$ that maps an item to a hash value, where B is a pre-defined upper bound known as the "bucket size." For simplicity, we use h_s to denote the hash value $h(s)$ of an item s. The hash value h_s indexes a row in the hash embedding table $E \in \mathbb{R}^{B\times d}$, yielding the embedding representation E_{h_s} of s. We use the same notation for both random variables and their realizations when the meaning is clear from context.

3.2 Probabilistic Hash Embeddings (PHE)

We first introduce our proposed encoding module for categorical items–probabilistic hash embeddings (PHE). A basic PHE has two components: a fixed hash function h and a hash embedding table E with a prior distribution $p(E)$. In this work, we assume each entry of E is independently Gaussian-distributed. Given an item s, PHE looks up the h_s-th row of E as its embedding E_{h_s}, whose distribution is $p(E_{h_s})$.

A single hash function may result in distinct inputs having the same hashing value, known as a hash collision, resulting in indistinguishable hash embeddings. With bucket size B, the collision probability is $O(1/B)$. To further reduce the collision rate, we employ universal hashing (Carter and Wegman 1977). Namely, instead of using one hash function, we use K hash functions with different random seeds but the same range. The collision probability then reduces to $O(1/B^K)$. Moreover, we share the same hash embedding table E across all K hash functions, which keeps the model size bounded.

We now describe how PHE encodes an item s. This procedure is illustrated in Fig. 2. With K hash functions, a categorical feature s results in K hash values $\mathbf{h}_s := \{h_s^{(1)}, \ldots, h_s^{(K)}\}$, which index K embeddings $\{E_{h_s^{(1)}}, \ldots, E_{h_s^{(K)}}\}$ where $E_{h_s^{(k)}}$ is the embedding retrieved using the k-th hash value $h_s^{(k)}$. The final representation $E_{\mathbf{h}_s} := g(E_{h_s^{(1)}}, \ldots, E_{h_s^{(K)}})$ of s is obtained using an assembly function $g : \mathbb{R}^{K\times d} \to \mathbb{R}^d$ to combine the K embeddings. Typical choices of g involve coordinate-wise summation, average, maximum, or minimum; other parametric choices of g include weighted sums where the weights come from another parametric model. The output of PHE is a probabilistic embedding $E_{\mathbf{h}_s}$ with distribution $p(E_{\mathbf{h}_s})$. Due to the shared embedding table, the memory cost of PHE is $O(Bd)$, independent of the number of hash functions K.

One core ML task is to model correlations between two variables. A common query is to compute the probability of observing a value of y given a categorical item s, i.e., $p(y|s)$. With PHE, we can compute it as

$$p(y|s) = \mathbb{E}_{p(E)}[p(y|E, \mathbf{h}_s)] = \mathbb{E}_{p(E)}[p(y|E_{\mathbf{h}_s})] \qquad (1)$$

where we follow the data generating process and treat $\mathbf{h}_s$ to be the same as s (which holds in the absence of hash collisions). To compute Eq. (1), we use Monte Carlo sampling to estimate the expectation. (See Supp. G.1 for practical implementation details.) This approach enables probabilistic inference conditioned on categorical features.

3.3 Online Learning of PHE

In this subsection, we derive scalable inference algorithms for PHE in a simple yet general model. These algorithms can be adapted to other model variants. We first focus on the static setting before expanding to the online setting.

Given observations $\mathcal{D} := \{(y_i, \mathbf{s}_i)\}_{i=1}^N$ where $\mathbf{s}_i$ represents a set of categorical features, we want to learn correlations between y and $\mathbf{s}$ using PHE. We assume observations are conditionally independently and identically distributed (i.i.d.) given E, with likelihood $p(y|E_{\mathbf{h}_\mathbf{s}})$.[3] PHE places a prior $p(E)$ over the hash embedding table E and infers the posterior given the observations $\mathcal{D}$. By Bayes rule, the posterior is $p(E|\mathcal{D}) \propto p(E)p(\mathcal{D}|E) = p(E)\prod_{i=1}^N p(y_i|E_{\mathbf{h}_{\mathbf{s}_i}})$, which is typically intractable with complex likelihoods.

Therefore, we turn to variational inference (Blei, Kucukelbir, and McAuliffe 2017; Zhang et al. 2018) and learn an approximate posterior by minimizing the Kullback-Leibler (KL) divergence $D_{\mathrm{KL}}(q_\lambda(E)|p(E|\mathcal{D}))$ between a variational distribution and the true posterior. We assume the variational posterior factorizes as $q_\lambda(E) = \prod_{b=1}^B \prod_{j=1}^d q_{\lambda_{bj}}(E_{bj})$ where each factor $q_{\lambda_{bj}}(E_{bj})$ is Gaussian with parameters $\lambda_{bj} := \{\mu_{bj} \in \mathbb{R}, \sigma_{bj} \in \mathbb{R}\}$. Finding the optimal variational distribution that minimizes the KL divergence is equivalent to finding the optimal parameters $\lambda^* := \{\lambda_{bj}^*\}$.

We assume the prior has the same factorization, with each entry independently following a standard Gaussian, i.e.,

[2] We also assume categorical features are single-valued, that is, each feature is assigned exactly one category. However, our work is compatible with multi-valued features.

[3] We use $E_{\mathbf{h}_\mathbf{s}}$ to denote the concatenation of hash embeddings for each item if $\mathbf{s}$ contains more than one categorical items.

$p(E_{bj}) = \mathcal{N}(0, 1)$. Substituting $p(E)$ and $q_\lambda(E)$ into the KL divergence yields an objective function $\mathcal{L}(\lambda)$ (also known as the evidence lower bound, ELBO) to be maximized (see derivations in Supp. A):

$$\mathcal{L}(\lambda) := \mathbb{E}_{q_\lambda(E)}\left[\sum_{i=1}^{N} \log p(y_i|E_{\mathbf{h}_{s_i}})\right] - \sum_{b=1}^{B}\sum_{j=1}^{d} D_{\mathrm{KL}}(q_{\lambda_{bj}}(E_{bj})|p(E_{bj})), \quad (2)$$

where the KL divergence between two Gaussians have a closed-form solution. The KL term also serves as regularization, preventing the variational posterior from deviating too far from the prior. We optimize Eq. (2) using the reparametrization trick (Rezende, Mohamed, and Wierstra 2014; Kingma and Welling 2013) and gradient-based methods. Let $\lambda_0^* := \arg\max \mathcal{L}(\lambda)$ denote the optimal parameters, which define the approximate posterior $q_{\lambda_0^*}(E)$ that minimizes $D_{\mathrm{KL}}(q_{\lambda_0^*}(E)|P(E|\mathcal{D}))$. For prediction on an unseen data point $(\hat{y}, \hat{s})$, the predictive distribution is $p(\hat{y}|\hat{s}, \mathcal{D}) = \mathbb{E}_{p(E|\mathcal{D})}[p(\hat{y}|E_{\mathbf{h}_{\hat{s}}})] \approx \mathbb{E}_{q_{\lambda_0^*}(E)}[p(\hat{y}|E_{\mathbf{h}_{\hat{s}}})]$.

Suppose we observe a second dataset $\mathcal{D}_1 := \{(y_i, \mathbf{s}_i)\}_{i=1}^{N_1}$ to which we would like our model to adapt while maintaining effectiveness on $\mathcal{D}$. Bayes rule provides a principled online learning approach: $p(E|\mathcal{D}_1, \mathcal{D}) \propto p(E|\mathcal{D})p(\mathcal{D}_1|E)$. This allows us to accommodate both datasets without storing them simultaneous. We iteratively replace the prior with the previous posterior and repeat the inference procedure. This iteration assumes $\mathcal{D}$ and $\mathcal{D}_1$ are conditionally i.i.d. given E.

Variational inference produces an approximation of the true posterior. We thus use $q_{\lambda_0^*}(E)$ in place of $p(E|\mathcal{D})$ and infer $\tilde{p}(E|\mathcal{D}_1, \mathcal{D}) \propto q_{\lambda_0^*}(E)p(\mathcal{D}_1|E)$. This again faces intractability, so we apply variational inference with a new variational distribution $q_\lambda(E)$ to minimize $D_{\mathrm{KL}}(q_\lambda(E)|\tilde{p}(E|\mathcal{D}_1, \mathcal{D}))$. Rewrite the KL divergence gives us a new ELBO (the objective function)

$$\mathcal{L}^{(1)}(\lambda; \lambda_0^*) := \mathbb{E}_{q_\lambda(E)}\left[\sum_{i=1}^{N_1} \log p(y_i|E_{\mathbf{h}_{s_i}})\right] - \sum_{b=1}^{B}\sum_{j=1}^{d} D_{\mathrm{KL}}(q_{\lambda_{bj}}(E_{bj})|q_{\lambda^*_{0,bj}}(E_{bj})). \quad (3)$$

Compared to Eq. (2), the original prior $p(E)$ of E is replaced with the previous posterior approximation $q_{\lambda_0^*}(E)$. Upon optimization convergence, the new optimal variational distribution approximates the posterior given both datasets ($\mathcal{D}$ and $\mathcal{D}_1$). When new datasets arrive in the future, we repeat the above online learning iteration to accommodate new datasets. Although this procedure resembles traditional continual learning (Wang et al. 2023; Nguyen et al. 2018; Li et al. 2021), it differs in focusing on evolving discrete vocabularies rather than continuous feature distributions. For any specified ML task, we can plug the corresponding likelihood model into Eqs. (2) and (3) and optimize.

Variational EM. Most expressive models contain learnable parameters θ in their likelihood $p_\theta(y|E_{\mathbf{h}_s})$. One needs to learn θ in addition to inferring the posterior of E. Fortunately, a minor modification to our previously derived objective functions (Eqs. (2) and (3)) can achieve this. We replace $p(y|E_{\mathbf{h}_s})$ with the parametric one $p_\theta(y|E_{\mathbf{h}_s})$ and maximize $\mathcal{L}(\lambda, \theta)$ with respect to $\{\lambda, \theta\}$ jointly. These new objective functions are viable and correspond to variational EM algorithms (Rezende, Mohamed, and Wierstra 2014; Kingma and Welling 2013), for which we provide proofs in Supp. A.3. Let $\{\lambda_0^*, \theta^*\}$ denote the optimizers of the modified Eq. (2), $\mathcal{L}(\lambda, \theta)$. During online learning, we fix both θ^* and λ_0^* in the modified objective function Eq. (3), $\mathcal{L}^{(1)}(\lambda; \lambda_0^*, \theta^*)$, to efficiently update the posterior over hash embeddings as new categorical items arrive.

Benefits of PHE in online learning. In data streaming or continual learning setup, PHE has natural benefits in reducing catastrophic forgetting: 1) only a few embeddings need to be updated online. This sparse updating scheme seldom affects other item representations, thus having less forgetting and more computing efficiency. 2) The online updates apply Bayesian online learning, in which the prior distribution serves as a regularization of previous knowledge that also reduces forgetting. Although continually expanding the embedding table with rows for new items typically improves accuracy, it comes at a cost. A larger embedding table requires more resources to deploy, reduces memory efficiency, and slows down execution. In contrast, PHE's memory/storage cost is bounded and does not increase with the number of distinct categorical values.

3.4 Why Is PHE Superior for Online Learning?

We showcase the benefits of PHE with Bayesian online learning: 1) It is equivalent to batch learning where all data are available at once. 2) This equivalence is independent of the data arrival order.

Given a dataset $\mathcal{D} = \{(y_i, s_i)\}_{i=1}^N$ and a prior over E. Assume data points are conditionally i.i.d. and have likelihood $\prod_{i=1}^N p(y_i|E, s_i)$. The posterior obtained from Bayesian *batch* learning (where we assume the whole dataset is available at once) is $p_{\text{batch}}(E|\mathcal{D}) \propto p(E)\prod_{i=1}^N p(y_i|E, s_i)$.

Now, suppose $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_N)$ is any permutation of the sequence $(1, \ldots, N)$, and let data in $\mathcal{D}$ arrive sequentially per the order $\boldsymbol{\pi}$ in an online learning setup. Let $\mathcal{D}_{\boldsymbol{\pi}}$ denote the dataset that has the arrival order of $\boldsymbol{\pi}$. Suppose the posterior obtained by Bayesian *online* learning is $p(E|\mathcal{D}_{\boldsymbol{\pi}})$.

Proposition 3.1. *For every permutation $\boldsymbol{\pi}$, the posterior $p_{batch}(E|\mathcal{D}) = p(E|\mathcal{D}_{\boldsymbol{\pi}})$ almost-everywhere.*

We provide the proof in Supp. B. Prop. 3.1 shows Bayesian online learning has the same power as Bayesian batch learning irrespective of the data arrival order.

However, for point estimation, such as maximum likelihood estimation, in order to achieve the same optimality as offline batch learning, an online learning algorithm has to be carefully designed and has to have a sophisticated learning rate schedule (Opper and Winther 1999; Orabona 2019).

Demo. To demonstrate Prop. 3.1, we design a simple online learning experiment involving two categorical items that arrive alternately. Each item is repeated ten times before the other one arrives. Each item is associated with a target value, and we use both traditional deterministic hash embeddings and PHE to learn these two values in an online fashion. We plot the results in Fig. 5 in Supp. F. The results show that while deterministic hash embeddings with a popular online learning algorithm incur large prediction errors on recurrent items, PHE makes much more accurate predictions, corroborating Prop. 3.1. Detailed settings and error analysis in Supp. F show that the *forgetting* phenomenon seen in traditional hash embeddings is caused by *parameter interference*, and that PHE alleviates the forgetting issue through adaptive regularization of the updated beliefs of hash embeddings.

Practical implementation Prop. 3.1 holds true for exact Bayesian inference, while our real-world data experiments utilize variational inference (VI) for approximate inference. There are approximation gaps (e.g., mean-field approximation), optimization gaps, and amortization gaps between VI and exact inference. In fact, we use Bayesian inference as a guide, and our experiments conform to the expectations. Secondly, unlike Bayesian neural networks that take long time to converge, PHE has sparse gradient updates and converges much faster. This is because PHE can be set to update only the embedding module while freezing other network parameters. Since each item activates at most K embeddings, by assuming independent embedding slots, the gradient updates on the embedding table are essentially sparse and data efficient. The elapsed time can be found in Sec. F.2.

4 Experiments

In this section, we conduct experiments to demonstrate the efficacy and memory efficiency of PHE in online learning settings where categorical features change. As follows, we introduce experimental protocols in Sec. 4.1. We then showcase the broad applicability of PHE as a plug-in module for a spectrum of models - both deterministic and probabilistic - and benchmark it against baselines in classification, multi-task sequence modeling, and recommendation systems in Secs. 4.2 to 4.4. Additional experiments and experimental settings are put in Sec. 4.5 and Supp. G. All results show that PHE outperforms its deterministic counterpart and performs similarly to the upper-bound collision-free embeddings in various domains and applications.

4.1 Experimental Protocols and Baselines

Experimental settings. We conducted our experiments using public datasets that contain categorical features and simulated them in data streaming environments where data points arrive sequentially. Our goal is to mimic practical settings where 1) new feature items can emerge and need to be learned, and 2) seen items can recur and need to be remembered.

Upon each data arrival, we conducted three operations in order: predict, evaluate, and update (embeddings). We report sequential results in plots and overall averaged results in tables. We repeated all experiments five times with different parameter initialization while keeping other settings fixed.

Baselines. Our work is the first work to handle open vocabulary in online dynamic environments. Appropriate baselines require handling both unbounded feature items and online updates simultaneously. To construct baselines for our setup, we need to combine existing embedding methods designed for offline inference with online updating techniques.

For embedding baselines, we use both hash embeddings (Ada) and collision-free expandable embeddings (EE). While EE is not desirable in practice due to its unbounded memory requirements and slow-down execution, we include it as a baseline to understand the performance gap (if any) resulting from our method's memory efficiency. We also implement a probabilistic version of EE, which we refer to as P-EE, to mitigate potential overfitting through Bayesian treatment.

For Ada, we choose fine-tuning as the online learning strategy, but vary the hyperparameter-training epoch-to give many candidates. We select different training epochs to account for various distribution shifts: FastAda assumes shifts are rapid while SlowAda assumes shifts happen slowly, and MedAda is in between. At each time step, FastAda trains 15 epochs as EE, while Med/SlowAda reduce training epochs to five and one, respectively. For all Ada baselines, we use the same learning rate as EE. We maintain this protocol across all applications to highlight the robustness of our method.

Training protocols. For all methods, we search hyperparameters on a validation set for each application. We test three training epochs (Fast/Medium/Slow) for Ada baselines and report the best results, giving them a competitive advantage. In all experiments, we employ a single shared hash embedding table (Coleman et al. 2024) for all categorical features. During online learning, we update only the hash embedding table while freezing all other model parameters, which were learned on an initial dataset. To determine the hash embedding table size, We selected $K = 3$ and a prime number B to support 10 times the expected size of the total vocabulary (where all tabular datasets use $B = 7$, the Retail dataset uses $B = 109$, and the MovieLens dataset uses $B = 10009$).

4.2 Application 1: Classification

We apply PHE to online learning classification tasks on four public datasets from scientific domains.

Methods. We now specify the likelihood model for classification tasks. Throughout the four datasets in use, we assume classification target variables follow categorical distributions and have a likelihood function $p_\theta(y|\mathbf{s}, \mathbf{x}) = \mathrm{Cat}(K, \mathrm{softmax}(\theta^\top[\mathbf{x}, E_{\mathbf{h_s}}]))$ where $\mathbf{x}$ are numeric features. This likelihood is then plugged into Eqs. (2) and (3).

Datasets. We apply four public static tabular datasets that are available in UCI Machine Learning Repository: Adult, Bank, Mushroom, and Covertype. These datasets contain a mixture of discrete and continuous columns and are collected for classification problems in various domains. For training stability, we normalized all continuous columns.

Experimental setups. To simulate the data-streaming setup, at each step we present a randomly sampled data mini-batch to the model and evaluate the online learning performance. We require only one column's item embeddings be updated, mimicking that column has a changing vocabulary. Besides, we initialize the model (both embeddings and neural network weights) with a separate random portion of the data.

Results. We reported the data-streaming online classification accuracy in Fig. 3 (left) and Fig. 6 in supplement. The

Table 1: Online learning results on all datasets. Adult, Bank, Mushroom, and Covertype are classification tasks evaluated by average accuracy, the larger the better. Retail and MovieLens-32M use mean absolute error, lower the better. All results are multiplied by 100 except Retail for visual clarity. PHE achieves the best performance among all hash embedding-based methods. We also include the compression ratios of PHE with respect to P-EE, which are computed by dividing the number of embedding parameters of PHE by the one of P-EE. (See details in Tab. 4 and Supp. G.6.) Notably, PHE takes as low as 2% memory of P-EE.

	Hash Embedding					Collision-Free Embedding	
	SlowAda	MediumAda	FastAda	PHE (ours)	Compression Ratio of PHE	EE	P-EE
Adult (↑)	82.2 ± 0.7	74.8 ± 4.5	71.1 ± 4.0	**84.1 ± 0.2**	0.09	84.2 ± 0.0	84.8 ± 0.0
Bank (↑)	**89.7 ± 0.1**	89.0 ± 0.9	86.9 ± 1.6	**89.6 ± 0.0**	0.2	90.0 ± 0.0	90.1 ± 0.0
Mushroom (↑)	97.7 ± 0.7	97.9 ± 0.5	98.3 ± 0.3	**98.8 ± 0.0**	0.62	98.8 ± 0.0	98.8 ± 0.0
CoverType (↑)	63.5 ± 0.5	59.1 ± 1.2	55.3 ± 1.2	**64.3 ± 0.2**	0.2	64.3 ± 0.1	64.0 ± 0.4
Retail (↓)	49.1±82.9	22.7±20.3	-	**3.0±0.2**	0.02	3.7±0.1	3.2±0.4
MovieLens (↓)	15.3±0.1	15.1±0.1	15.1±0.1	**14.7±0.0**	0.04	15.1±0.0	14.7±0.0

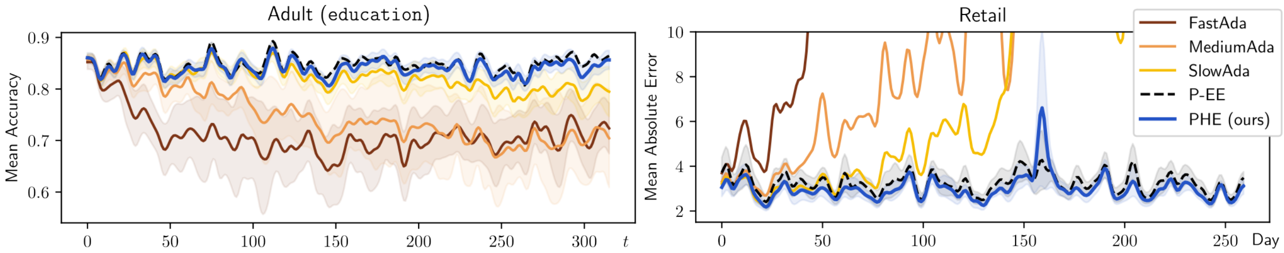

Figure 3: (**left**) Online classification results on `Adult` tabular data streams. In the parentheses is the column whose items embeddings get updated. The Ada results show a downward trend although there are no new items to learn, suggesting the deterministic hash embeddings suffer from forgetting during the learning. In contrast, the proposed PHE mitigates the forgetting issue and keeps performing as good as the upper-bound method P-EE. Other datasets in Fig. 6 in Supp. G.3 show similar conclusions. (**right**) Results of sequence modeling on `Retail` data-streams. It shows that PHE outperforms all Ada baselines that are sensitive to their optimization hyperparameters. Moreover, it is remarkable to note that PHE performs slightly better than the collision-free P-EE baseline, especially considering PHE consumes only 2% of the memory of P-EE.

facts that 1) any items seen during online learning have been learned at the initialization and that 2) the accuracy curves of Ada methods have a downward trend suggest hash embeddings suffers from forgetting. In fact, the forgetting is caused by *parameter interference* in shared hash embeddings: suppose items A and B share parameters in the hash embedding table, then updating A's embedding affect B's embedding.

We further reported an overall averaged accuracy in Tab. 1. The results show that our proposed PHE performs similarly with the upper-bound collision-free embeddings (EE), and the gap between PHE and all other deterministic counterparts proves the effectiveness of PHE in online learning. Besides, PHE is more stable and has a smaller variance. Notably, PHE applies the same set of hyperparameters and outperforms all Ada baselines across all datasets. The varying performances of the Ada baselines highlight the importance and sensitivity of hyperparameter tuning for deterministic hash embeddings. In contrast, the only demand of our method is to train the model until convergence–a simpler optimization criterion. Lastly, as summarized in Tabs. 1 and 4, PHE consumes noticeably lower memory than P-EE.

4.3 Application 2: Multi-Task Sequence Modeling

Sequence models can exploit temporal correlations among observations to make predictions based on histories. We now switch to a more sophisticated multi-task sequence modeling problem where each task has its own sequential characteristics, and we aim to personalize the sequence model for each task. Sequential Tasks are identified by categorical features.

Methods. We model sequences using deep Kalman filters (DKF) (Krishnan, Shalit, and Sontag 2015), which are latent variable models and can handle uncertainties and non-stationary processes. In sequence data, we have an additional timestamp feature t. We model the dependency between neighboring data points by a latent time variable $\mathbf{z}$. Specifically, we assume $\mathbf{z}$ is Gaussian distributed and follows the distribution $p(\mathbf{z}_i|\mathbf{z}_{i-1}, \Delta_i; \theta_z) = \mathcal{N}(\mathbf{z}_i|f_{\theta_z}(\mathbf{z}_{i-1}, \Delta_i))$ where $f_{\theta_z} := \{\mu_{\theta_z}, \Sigma_{\theta_z}\}$ is a multi-layer perceptron that outputs mean and covariance of $\mathbf{z}_i$. Δ_i is the difference in timestamp between the i-th and $i-1$-th observations. We assume the conditional likelihood model is $p(y_i|\mathbf{x}_i, E_{\mathbf{h}_{\mathbf{s}_i}}, \mathbf{z}_i)$, which is a parametric Poisson distribution.

In the above model, we need to infer the posteriors of E and $\mathbf{z}$. We assume the variational posterior distribution factorizes as $q_{\lambda,\phi}(E, \mathbf{z}_{\leq N}|\mathbf{x}_{\leq N}, \mathbf{y}_{\leq N}, \mathbf{h}_{\mathbf{s}_{\leq N}}) = q_\lambda(E) \prod_{i=1}^{N} q_\phi(\mathbf{z}_i|\mathbf{x}_{\leq i}, \mathbf{y}_{\leq i}, E_{\mathbf{h}_{\mathbf{s}_{\leq i}}})$ where $q_\lambda(E)$ is the same as before and $q_\phi(\mathbf{z}_i|\mathbf{x}_{\leq i}, \mathbf{y}_{\leq i}, E_{\mathbf{h}_{\mathbf{s}_{\leq i}}})$ is a Gaussian with diagonal covariance implemented as a recurrent neural network with parameters ϕ. Concretely, we use gated recurrent unit (GRU) (Chung et al. 2014).

We repeat the KL divergence minimization derivation procedure to obtain the following ELBO objective function $\mathcal{L}(\theta, \lambda, \phi) := \mathbb{E}_{q_\lambda(E)} \left[\sum_{i=1}^{N} \mathcal{L}_i(\theta, \phi|E)\right] - D_{\mathrm{KL}}(q_\lambda(E)|p(E))$ where $\mathcal{L}_i(\theta, \phi|E)$ is the conditional ELBO of the i-th data's log-likelihood $\mathcal{L}_i(\theta, \phi|E) := \mathbb{E}_{q_\phi(\mathbf{z}_i)}[\log p(\mathbf{y}_i|\mathbf{z}_i, \mathbf{x}_i, E_{\mathbf{h}_{\mathbf{s}_i}}; \theta_y)] - \mathbb{E}_{q_\phi(\mathbf{z}_{i-1})}[D_{\mathrm{KL}}(q_\phi(\mathbf{z}_i)|p(\mathbf{z}_i|\mathbf{z}_{i-1}; \theta_z))]$. We provide the full derivation of these ELBOs in Supp. A. After learning the initial dataset, we will fix all model parameters except the hash embedding table E in learning future datasets.

Datasets. We apply a public large-scale time-stamped tabular dataset, Retail. A snippet of this dataset can be found in Tab. 3. This dataset records all online transactions between 01/12/2010 and 09/12/2011 in a retail store. There are over 4,000 products and over 540K time-stamped invoice records in total. The task is to predict the sales for each product shown in each invoice given the product's historical sales.

Experimental setups. We use the first three month data to initialize the model. Then we make predictions on a daily basis following the invoice timestamp. And at each step, we predict the sales quantity for each product on invoices based on their sale history. After that, we will receive the prediction error and use it to update the product embeddings. We use mean absolute errors for evaluation (see Supp. G.4).

Results. Fig. 3 (right) shows the running performance (smoothed by a 1-D Gaussian filter): the Ada-family baselines favor shorter optimization time for Retail, as long optimization time like FastAda explodes after 50 days. (The error bar is omitted as it is too large to be meaningful.) On the other hand, PHE has lower error and is stable across all learning steps. Remarkably, on the average performance in Tab. 1, PHE significantly outperforms all baselines, including collision-free P-EE with only 2% memory usage. One possible reason is that P-EE initializes new embeddings from scratch and thus gets slow in warm-up, while PHE uses shared parameters from initial training. Similar observations also occur in the continual learning setup (see Fig. 11 and Tab. 5 in Supp. G.4). and the recommendation task below.

4.4 Application 3: Large-Scale Recommendation

Large-scale recommendation systems have seen quite a bit of change in categorical features. For example, new users or movies (categorical items) reach a streaming service, the recommender needs to incorporate them and make recommendations. We now demonstrate how PHE can assist recommendation systems in online learning.

Methods. We treat the recommendation problem as a rating prediction problem, where the task is predicting the rating a user gives to a movie. We combine PHE and Neural Collaborative Filtering (He et al. 2017) as the backbone model. We assume all ratings are iid Gaussian distributed conditioned on user, movie, and movie-genre embeddings. And we model user and movie embeddings through PHE, denoted by $E_{\mathbf{h}_u}$ and $E_{\mathbf{h}_m}$, while movie-genres are encoded as multi-hot embeddings denoted by $\mathbf{x}$. Then the likelihood is $p_\theta(y|E_{\mathbf{h}_u}, E_{\mathbf{h}_m}, \mathbf{x})$ with learnable parameters θ. θ are the weights of a two-layer neural network. We model the mean of y as the network output conditioned on input $[E_{\mathbf{h}_u}, E_{\mathbf{h}_m}, E_{\mathbf{h}_u} \odot E_{\mathbf{h}_m}, \mathbf{x}]$.

Datasets. We apply the largest MovieLens-32m (Harper and Konstan 2015) which contains 32 million ratings across over 87k movies and 200k users. These data were recorded between 1/9/1995 and 10/12/2023 for about 28 years. Each piece of data is a tuple of `(userId, movieId, rating, timestamp)`, recording when and which rating a user gave a movie. Ratings range from 0 to 5 stars with half-star increments.

Experimental setups. In implementation, ratings are normalized to $[0, 1]$ and are taken to be continuous albeit their increments are discrete. We simulate the experiment as in production – online prediction along the timestamp. The model is pre-trained on the first five years of data and then perform predict-update online learning on a daily basis. In this setup, both forgetting and adaptation in the hash embeddings are measured: the model should avoid forgetting for recurring users/movies and adapt for new users/movies. Prediction error is evaluated by mean absolute error.

Results. The results of all compared methods are shown in Fig. 6 in supplement and the memory efficiency of PHE is reported in Tabs. 1 and 4. It shows that PHE outperforms all deterministic hash embedding baselines (Fast/Medium/SlowAda) that have various forgetting-adaptation trade-offs. PHE also significantly outperforms the collision-free P-EE baseline. This is remarkable considering PHE consumes only 4% of the memory of P-EE. EE, the deterministic version of P-EE, has worse performance, possibly due to overfitting.

4.5 Additional Results

We conducted additional experiments and presented the results in Supp. G.6. We showcased additional motivating examples beside Fig. 1; demonstrated the **memory and hardware efficiency** of PHE in Tab. 4; analyzed **adaptation and forgetting** separately in Fig. 10; investigated classification and sequence modeling in classical **continual learning setup** in Fig. 11; performed **ablation studies** on the hash size B, the number of hash functions K, compared multiple **update schemes**, and the performance of **double-size** Ada baselines.

5 Conclusions

In this work we unveiled the ineffectiveness of hash embeddings in online learning of categorical features. We proposed probabilistic hash embeddings (PHE) and addressed the problem of online learning. We showcased PHE is a plug-in module for multiple ML models in various domains and applications, allowing these models to learn categorical features in a streaming fashion. We derive scalable inference algorithms to simultaneously learn the model parameters and infer the latent embeddings. Through Bayesian online learning, the model is able to adapt to new vocabularies without additional hyperparameters in a changing environment. We benchmark PHE and baselines on large-scale public datasets to demonstrate the efficacy of our method.

References

Al-Hashedi, K. G.; and Magalingam, P. 2021. Financial fraud detection applying data mining techniques: A comprehensive review from 2009 to 2019. *Computer Science Review*, 40: 100402.

Arik, S. Ö.; and Pfister, T. 2021. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 6679–6687.

Blei, D. M.; Kucukelbir, A.; and McAuliffe, J. D. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518): 859–877.

Carter, J. L.; and Wegman, M. N. 1977. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, 106–112.

Cheng, D. Z.; Wang, R.; Kang, W.-C.; Coleman, B.; Zhang, Y.; Ni, J.; Valverde, J.; Hong, L.; and Chi, E. 2023. Efficient Data Representation Learning in Google-scale Systems. In *Proceedings of the 17th ACM Conference on Recommender Systems*, 267–271.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Clements, J. M.; Xu, D.; Yousefi, N.; and Efimov, D. 2020. Sequential deep learning for credit risk monitoring with tabular financial data. *arXiv preprint arXiv:2012.15330*.

Coleman, B.; Kang, W.-C.; Fahrbach, M.; Wang, R.; Hong, L.; Chi, E.; and Cheng, D. 2024. Unified Embedding: Battle-tested feature representations for web-scale ML systems. *Advances in Neural Information Processing Systems*, 36.

Du, L.; Gao, F.; Chen, X.; Jia, R.; Wang, J.; Zhang, J.; Han, S.; and Zhang, D. 2021. TabularNet: A neural network architecture for understanding semantic structures of tabular data. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 322–331.

Dulac-Arnold, G.; Evans, R.; van Hasselt, H.; Sunehag, P.; Lillicrap, T.; Hunt, J.; Mann, T.; Weber, T.; Degris, T.; and Coppin, B. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.

Girin, L.; Leglaive, S.; Bie, X.; Diard, J.; Hueber, T.; and Alameda-Pineda, X. 2021. Dynamical Variational Autoencoders: A Comprehensive Review. *Foundations and Trends in Machine Learning*, 15(1-2): 1–175.

Han, S.; Hu, X.; Huang, H.; Jiang, M.; and Zhao, Y. 2022. Adbench: Anomaly detection benchmark. *Advances in Neural Information Processing Systems*, 35: 32142–32159.

Harper, F. M.; and Konstan, J. A. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4): 1–19.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, 173–182.

Huang, X.; Khetan, A.; Cvitkovic, M.; and Karnin, Z. 2020. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*.

Iida, H.; Thai, D.; Manjunatha, V.; and Iyyer, M. 2021. TABBIE: Pretrained Representations of Tabular Data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 3446–3456.

Jerfel, G.; Grant, E.; Griffiths, T.; and Heller, K. A. 2019. Reconciling meta-learning and continual learning with online mixtures of tasks. *Advances in neural information processing systems*, 32.

Kang, W.-C.; Cheng, D. Z.; Yao, T.; Yi, X.; Chen, T.; Hong, L.; and Chi, E. H. 2021. Learning to embed categorical features without embedding tables for recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 840–850.

Kim, M. J.; Grinsztajn, L.; and Varoquaux, G. 2024. CARTE: Pretraining and Transfer for Tabular Learning. In *Forty-first International Conference on Machine Learning*.

Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kireev, K.; Andriushchenko, M.; Troncoso, C.; and Flammarion, N. 2023. Transferable Adversarial Robustness for Categorical Data via Universal Robust Embeddings. *arXiv preprint arXiv:2306.04064*.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13): 3521–3526.

Ko, H.; Lee, S.; Park, Y.; and Choi, A. 2022. A survey of recommendation systems: recommendation models, techniques, and application fields. *Electronics*, 11(1): 141.

Kotelnikov, A.; Baranchuk, D.; Rubachev, I.; and Babenko, A. 2023. Tabddpm: Modelling tabular data with diffusion models. In *International Conference on Machine Learning*, 17564–17579. PMLR.

Krishnan, R. G.; Shalit, U.; and Sontag, D. 2015. Deep kalman filters. *arXiv preprint arXiv:1511.05121*.

Lai, F.; Zhang, W.; Liu, R.; Tsai, W.; Wei, X.; Hu, Y.; Devkota, S.; Huang, J.; Park, J.; Liu, X.; et al. 2023. {AdaEmbed}: Adaptive Embedding for {Large-Scale} Recommendation Models. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 817–831.

Le, F.; Srivatsa, M.; Ganti, R.; and Sekar, V. 2022. Rethinking data-driven networking with foundation models: challenges and opportunities. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, 188–197.

Li, A.; Boyd, A.; Smyth, P.; and Mandt, S. 2021. Detecting and adapting to irregular distribution shifts in bayesian online learning. *Advances in neural information processing systems*, 34: 6816–6828.

Liu, H.; Di, S.; and Chen, L. 2023. Incremental Tabular Learning on Heterogeneous Feature Space. *Proceedings of the ACM on Management of Data*, 1(1): 1–18.

Liu, T.; Fan, J.; Li, G.; Tang, N.; and Du, X. 2023. Tabular data synthesis with generative adversarial networks: design space and optimizations. *The VLDB Journal*, 1–26.

Lopez-Paz, D.; and Ranzato, M. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.

Nguyen, C. V.; Li, Y.; Bui, T. D.; and Turner, R. E. 2018. Variational Continual Learning. In *International Conference on Learning Representations*.

Opper, M.; and Winther, O. 1999. A Bayesian approach to on-line learning.

Orabona, F. 2019. A modern introduction to online learning. *arXiv preprint arXiv:1912.13213*.

Rezende, D. J.; Mohamed, S.; and Wierstra, D. 2014. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, 1278–1286. PMLR.

Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; and Hadsell, R. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Sarker, I. H.; Kayes, A.; Badsha, S.; Alqahtani, H.; Watters, P.; and Ng, A. 2020. Cybersecurity data science: an overview from machine learning perspective. *Journal of Big data*, 7: 1–29.

Serrà, J.; and Karatzoglou, A. 2017. Getting deep recommenders fit: Bloom embeddings for sparse binary input/output networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, 279–287.

Shehab, M.; Abualigah, L.; Shambour, Q.; Abu-Hashem, M. A.; Shambour, M. K. Y.; Alsalibi, A. I.; and Gandomi, A. H. 2022. Machine learning in medical applications: A review of state-of-the-art methods. *Computers in Biology and Medicine*, 145: 105458.

Shi, H.-J. M.; Mudigere, D.; Naumov, M.; and Yang, J. 2020a. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 165–175.

Shi, S.; Ma, W.; Zhang, M.; Zhang, Y.; Yu, X.; Shan, H.; Liu, Y.; and Ma, S. 2020b. Beyond user embedding matrix: Learning to hash for modeling large-scale users in recommendation. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, 319–328.

Siadati, H.; and Memon, N. 2017. Detecting structurally anomalous logins within enterprise networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 1273–1284.

Tito Svenstrup, D.; Hansen, J.; and Winther, O. 2017. Hash embeddings for efficient word representations. *Advances in neural information processing systems*, 30.

Wang, L.; Zhang, X.; Su, H.; and Zhu, J. 2023. A comprehensive survey of continual learning: Theory, method and application. *arXiv preprint arXiv:2302.00487*.

Weinberger, K.; Dasgupta, A.; Langford, J.; Smola, A.; and Attenberg, J. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, 1113–1120.

Xu, L.; Skoularidou, M.; Cuesta-Infante, A.; and Veeramachaneni, K. 2019. Modeling tabular data using conditional gan. *Advances in neural information processing systems*, 32.

Yin, P.; Neubig, G.; Yih, W.-t.; and Riedel, S. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 8413–8426.

Yoon, J.; Yang, E.; Lee, J.; and Hwang, S. J. 2017. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*.

Zenke, F.; Poole, B.; and Ganguli, S. 2017. Continual learning through synaptic intelligence. In *International conference on machine learning*, 3987–3995. PMLR.

Zhang, C.; Bütepage, J.; Kjellström, H.; and Mandt, S. 2018. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8): 2008–2026.

Zhao, Z.; Kunar, A.; Birke, R.; and Chen, L. Y. 2021. Ctabgan: Effective table data synthesizing. In *Asian Conference on Machine Learning*, 97–112. PMLR.

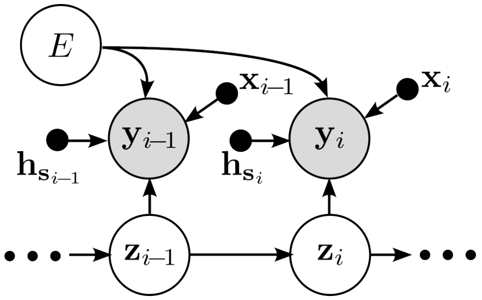

Figure 4: A graphical model of a temporal sequence with PHE. The changing categorical values are contained in $\mathbf{h}_\mathbf{s}$.

A Evidence Lower Bounds

A.1 Variational inference as KL divergence minimization

$$\min_q D_{\mathrm{KL}}(q_\lambda(E)|p(E|\mathcal{D})) \tag{4}$$

$$= \min_q \mathbb{E}_{q_\lambda(E)}[\log q_\lambda(E) - \log p(E|\mathcal{D})] \tag{5}$$

$$= \min_q \mathbb{E}_{q_\lambda(E)}[\log q_\lambda(E) - \log p(E) - \log p(\mathcal{D}|E) + \log p(\mathcal{D})] \tag{6}$$

$$= \max_q \mathbb{E}_{q_\lambda(E)}[\log p(\mathcal{D}|E)] - D_{\mathrm{KL}}(q_\lambda(E)|p(E)) \tag{7}$$

$$= \max_\lambda \mathcal{L}(\lambda) \tag{8}$$

A.2 Derivation of $\mathcal{L}(\theta, \lambda, \phi)$ in Sec. 4.3

The data generating process is summarized in Fig. 4. We denote the model parameters relevant to the generating process by $\theta := \{\theta_z, \theta_y\}$. To learn the model parameters, we maximize the marginal likelihood $p(\mathbf{y}_{\leq N}|\mathbf{x}_{\leq N}, \mathbf{h}_{\mathbf{s}_{\leq N}}; \theta)$. Directly optimizing this marginal likelihood with the Expectation-Maximization (EM) algorithm is intractable. Therefore, we jointly learn the model parameters θ and infer the variational posteriors of latent variables $\{E, \mathbf{z}_{\leq N}\}$ using the variational EM algorithm. That is, we maximize the evidence lower bound (ELBO) $\mathcal{L}(\theta, \lambda, \phi)$ with respect to model parameters θ and variational parameters $\{\lambda, \phi\}$.

Denote all the history $\{\mathbf{x}_{\leq i}, \mathbf{y}_{\leq i}, E_{\mathbf{h}_{\mathbf{s}_{\leq i}}}\}$ until row i by O_i. We find the optimal parameters by maximizing the marginal evidence $p(\mathbf{y}_{\leq N}|\mathbf{x}_{\leq N}, \mathbf{h}_{\mathbf{s}_{\leq N}}; \theta)$. We take the logarithm of marginal evidence

$$\log p(\mathbf{y}_{\leq N}|\mathbf{x}_{\leq N}, \mathbf{h}_{\mathbf{s}_{\leq N}}; \theta) \tag{9}$$

$$= \log \int \frac{p(\mathbf{y}_{\leq N}, E|\mathbf{x}_{\leq N}, \mathbf{h}_{\mathbf{s}_{\leq N}}; \theta)}{q_\lambda(E)} q_\lambda(E) dE \tag{10}$$

$$\geq \mathbb{E}_{q_\lambda(E)}[\log p(\mathbf{y}_{\leq N}, E|\mathbf{x}_{\leq N}, \mathbf{h}_{\mathbf{s}_{\leq N}}; \theta) - \log q_\lambda(E)] \tag{11}$$

$$= \mathbb{E}_{q_\lambda(E)}[\log p(\mathbf{y}_{\leq N}|\mathbf{x}_{\leq N}, E_{\mathbf{h}_{\mathbf{s}_{\leq N}}}; \theta)] - D_{\mathrm{KL}}(q_\lambda(E)|p(E)) \tag{12}$$

where the inequality follows from Jensen's inequality. Next, we apply the same trick for another time to find a lower bound of Eq. (12). Specifically, we will find a tractable lower bound to the conditional likelihood $\log p(\mathbf{y}_{\leq N}|\mathbf{x}_{\leq N}, E_{\mathbf{h}_{\mathbf{s}_{\leq N}}}; \theta)$.

In the filtering setup, we note that $\log p(\mathbf{y}_{\leq N}|\mathbf{x}_{\leq N}, E_{\mathbf{h}_{\mathbf{s}_{\leq N}}}; \theta) = \sum_{i=1}^N \log p(\mathbf{y}_i|\mathbf{y}_{<i}, \mathbf{x}_{\leq i}, E_{\mathbf{h}_{\mathbf{s}_{\leq i}}}; \theta)$. If we can find a lower bound for each $\log p(\mathbf{y}_i|\mathbf{y}_{<i}, \mathbf{x}_{\leq i}, E_{\mathbf{h}_{\mathbf{s}_{\leq i}}}; \theta)$, then the summation of the lower bounds is also a valid lower bound for $\log p(\mathbf{y}_{\leq N}|\mathbf{x}_{\leq N}, E_{\mathbf{h}_{\mathbf{s}_{\leq N}}}; \theta)$.

$$\log p(\mathbf{y}_i|\mathbf{y}_{<i}, \mathbf{x}_{\leq i}, E_{\mathbf{h}_{\mathbf{s}_{\leq i}}}; \theta) \tag{13}$$

$$= \log \int \frac{p(\mathbf{y}_i, \mathbf{z}_i|\mathbf{y}_{<i}, \mathbf{x}_{\leq i}, E_{\mathbf{h}_{\mathbf{s}_{\leq i}}}; \theta)}{q_\phi(\mathbf{z}_i|O_i)} q_\phi(\mathbf{z}_i|O_i) d\mathbf{z}_i \tag{14}$$

$$\geq \mathbb{E}_{q_\phi(\mathbf{z}_i|O_i)}[\log p(\mathbf{y}_i|\mathbf{y}_{<i}, \mathbf{x}_{\leq i}, E_{\mathbf{h}_{\mathbf{s}_{\leq i}}}, \mathbf{z}_i; \theta)] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}_i|O_i)|p(\mathbf{z}_i|O_{i-1})) \tag{15}$$

$$\geq \mathbb{E}_{q_\phi(\mathbf{z}_i|O_i)}[\log p(\mathbf{y}_i|\mathbf{y}_{<i}, \mathbf{x}_{\leq i}, E_{\mathbf{h}_{\mathbf{s}_{\leq i}}}, \mathbf{z}_i; \theta)] - \mathbb{E}_{q(\mathbf{z}_{i-1}|O_{i-1})} D_{\mathrm{KL}}(q_\phi(\mathbf{z}_i|O_i)|p(\mathbf{z}_i|\mathbf{z}_{i-1}; \theta_z)) \tag{16}$$

Eq. (15) to Eq. (16) follows from the following inequality:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}_i|O_i)|p(\mathbf{z}_i|O_{i-1})) \leq \mathbb{E}_{q(\mathbf{z}_{i-1}|O_{i-1})} D_{\mathrm{KL}}(q_\phi(\mathbf{z}_i|O_i)|p(\mathbf{z}_i|\mathbf{z}_{i-1}; \theta_z)) \tag{17}$$

because

$$\begin{aligned}
&D_{\mathrm{KL}}(q_\phi(\mathbf{z}_i|O_i)|p(\mathbf{z}_i|O_{i-1})) && (18)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}_i|O_i)}[\log q_\phi(\mathbf{z}_i|O_i) - \log p(\mathbf{z}_i|O_{i-1})] && (19)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}_i|O_i)}\left[\log q_\phi(\mathbf{z}_i|O_i) - \log \mathbb{E}_{q(\mathbf{z}_{i-1}|O_{i-1})}[p(\mathbf{z}_i|\mathbf{z}_{i-1};\theta_z)]\right] && (20)\\
&\le \mathbb{E}_{q_\phi(\mathbf{z}_i|O_i)}\left[\log q_\phi(\mathbf{z}_i|O_i) - \mathbb{E}_{q(\mathbf{z}_{i-1}|O_{i-1})}[\log p(\mathbf{z}_i|\mathbf{z}_{i-1};\theta_z)]\right] && (21)\\
&= \mathbb{E}_{q(\mathbf{z}_{i-1}|O_{i-1})q_\phi(\mathbf{z}_i|O_i)}[\log q_\phi(\mathbf{z}_i|O_i) - \log p(\mathbf{z}_i|\mathbf{z}_{i-1};\theta_z)] && (22)\\
&= \mathbb{E}_{q(\mathbf{z}_{i-1}|O_{i-1})} D_{\mathrm{KL}}(q_\phi(\mathbf{z}_i|O_i)|p(\mathbf{z}_i|\mathbf{z}_{i-1};\theta_z)). && (23)
\end{aligned}$$

Then Eq. (16) is the conditional ELBO $\mathcal{L}_i(\theta, \phi|E)$ in Sec. 4.3. Plug Eq. (16) in Eq. (12), we have

$$\begin{aligned}
&\mathbb{E}_{q_\lambda(E)}[\log p(\mathbf{y}_{\le N}|\mathbf{x}_{\le N}, E_{\mathbf{h}_{\mathbf{s}_{\le N}}};\theta)] - D_{\mathrm{KL}}(q_\lambda(E)|p(E)) && (24)\\
&\ge \mathbb{E}_{q_\lambda(E)}\left[\sum_{i=1}^{N} \mathcal{L}_i(\theta, \phi|E)\right] - D_{\mathrm{KL}}(q_\lambda(E)|p(E)) && (25)
\end{aligned}$$

which is our objective function $\mathcal{L}(\theta, \phi, \lambda)$ in Sec. 4.3.

A.3 $\mathcal{L}(\theta, \phi, \lambda)$ as a Variational EM Algorithm

Why is maximizing $\mathcal{L}(\theta, \phi, \lambda)$ a meaningful objective as a variational expectation-maximization algorithm? We start with a general latent variable model $p_\theta(x, z) = p(z)p_\theta(x|z)$ and infer the posterior $p_\theta(z|x)$.

$$\begin{aligned}
&D_{\mathrm{KL}}(q_\lambda(z)|p_\theta(z|x))\\
:=& \mathbb{E}_{q_\lambda(z)}[\log q_\lambda(z) - \log p_\theta(z|x)]\\
=& \mathbb{E}_{q_\lambda(z)}[\log q_\lambda(z) - \log p_\theta(x, z) + \log p_\theta(x)]\\
=& -\mathcal{L}(\lambda, \theta) + \log p_\theta(x)
\end{aligned}$$

Re-ordering the equation yields

$$\mathcal{L}(\lambda, \theta) = \log p_\theta(x) - D_{\mathrm{KL}}(q_\lambda(z)|p_\theta(z|x)),$$

which shows that maximizing the ELBO $\mathcal{L}(\lambda, \theta)$ is equivalent to both maximizing the marginal likelihood $p_\theta(x)$ and minimizing the inference gap $D_{\mathrm{KL}}(q_\lambda(z)|p_\theta(z|x))$.

Then, with the same procedure as above, two facts follow: 1) maximizing $\mathcal{L}_i(\theta, \phi|E)$ is equivalent to maximizing the conditional likelihood $\log p(\mathbf{y}_i|\mathbf{y}_{<i}, \mathbf{x}_{\le i}, E_{\mathbf{h}_{\mathbf{s}_{\le i}}};\theta)$ and minimizing the inference gap $D_{\mathrm{KL}}(q_\phi(\mathbf{z}_i|O_i)|p(\mathbf{z}_i|O_i;\theta))$ simultaneously; 2) maximizing Eq. (12) is equivalent to maximizing $\log p(\mathbf{y}_{\le N}|\mathbf{x}_{\le N}, \mathbf{h}_{\mathbf{s}_{\le N}};\theta)$ and minimizing the inference gap $D_{\mathrm{KL}}(q_\lambda(E)|p(E|\mathbf{y}_{\le N}, \mathbf{x}_{\le N}, \mathbf{h}_{\mathbf{s}_{\le N}};\theta))$ simultaneously. Since maximizing $\mathcal{L}(\theta, \phi, \lambda)$ optimizes both $\mathcal{L}_i(\theta, \phi|E)$ and Eq. (12), we conclude our objective function will optimize all the mentioned aspects above.

A.4 Derivation of $\mathcal{L}^{(1)}(\lambda; \theta^*, \lambda_0^*, \phi^*)$ in Sec. 4.3

We only adapt the probabilistic hash embedding E. Similar to Bayesian online learning where the previous posterior is used as the new prior, we use the previous approximate posterior $q_{\lambda_0^*}(E)$ as the new prior for dataset $\mathcal{D}_1$ and fix all the other model parameters θ^*, ϕ^*. The derivation is the same as the one for Eq. (12) except we replace $p(E)$ with $q_{\lambda_0^*}(E)$. We only update λ to acquire the new posterior in the optimization.

B Proof of Theorem 3.1

Proof. The proof is simple and based on repetitive applications of Bayes rule. Let $\mathcal{D}_{\pi_i}$ denote the subset of data arrived before (and includes) π_i. We assume the data arrive one by one and we cannot store previous data.

$$\begin{aligned}
p(E|\mathcal{D}_{\pi_N}) &\propto p(E|\mathcal{D}_{\pi_{N-1}})p(y_{\pi_N}|E, s_{\pi_N})\\
&\propto p(E|\mathcal{D}_{\pi_{N-2}})p(y_{\pi_N}|E, s_{\pi_N})p(y_{\pi_{N-1}}|E, s_{\pi_{N-1}})\\
&\dots\\
&\propto p(E)\prod_{i=1}^{N} p(y_{\pi_i}|E, s_{\pi_i})
\end{aligned}$$

Because $\prod_{i=1}^{N} p(y_{\pi_i}|E, s_{\pi_i}) = \prod_{i=1}^{N} p(y_i|E, s_i)$, we conclude $p(E|\mathcal{D}_{\pi_N})$ is the same as $p_{\text{batch}}(E|\mathcal{D})$. □

C Limitation

We discuss two limitations in this study. 1) The need to pre-specify embedding parameters K and B can limit model adaptability if actual category counts exceed expectations (though this can be mitigated through retraining or adding a forgetting mechanism to KL term in Eq. (3); 2) Restricting updates to embeddings E while fixing network parameters θ may speed up the consumption of E under strong distribution shifts, blocking new learnings and forgetting old. Meanwhile, Our ablation in Tab. 6 suggests that updating θ requires more sophisticated strategies to prevent forgetting.

D Related work

	$D0$	$D1$	$D2$	$D3$	$D4$
Changing vocabulary			✓	✓	✓
Timestamped		✓		✓	✓
Multi-task					✓

Table 2: Tabular datasets can be categorized into five categories ($D0 - D4$) based on combinations of three characteristics, i.e., whether their categorical feature vocabulary dynamically expands over time, whether they contain a specific timestamp column, and whether their nature is multi-task. For example, datasets without all these characteristics are considered static ($D0$). While existing works mainly consider $D0$ and $D1$, PHE fits all dataset types ($D0 - D4$) and specifically highlights the unique applicability for dynamic and temporal tabular data types ($D1 - D4$).

We extend the discussion in Sec. 2 and survey more related works. In a nutshell, our PHE applies to all tabular data types in Tab. 2 (i.e., $D0 - D4$) while existing works are targeted to $D0$ or $D1$.

Our work deals with multi-task dynamic temporal tabular data. Our method has two major components: the probabilistic hash embeddings that learn categorical feature representations and the latent variable model for multi-task temporal tabular data. Next, we discuss the main related works.

Hash features. PHE is motivated by hashing tricks. Weinberger et al. (2009) proposed to use one hash function to map categorical features to a one-hot hash embedding of length B, which is the bucket size. The drawback is the embedding size is too large because there is only one hash function and that requires a large bucket size B to get rid of collision. Bloom Embeddings (Serrà and Karatzoglou 2017) is based on Bloom filters and achieves efficient computation while maintaining a compact model size. Other previous work on using hashing tricks to generate features focuses on using a smaller number of embedding-related parameters to achieve the same performance as using one-hot encoding. Hash embeddings or unified embeddings (Tito Svenstrup, Hansen, and Winther 2017; Cheng et al. 2023) use a shared embedding table for all categorical features and multiple hashing functions as indices of the embedding table, reducing the possibility of collision. Hash embeddings are designed for stationary vocabularies, emphasizing small parameter sizes. We generalize hash embeddings to a probabilistic version that enables us to learn changing vocabularies via Bayesian online learning. Composition Embeddings (Shi et al. 2020a) use multiple hash embedding tables; in contrast, PHE uses one shared embedding table, further reducing the memory cost. Wolpertinger (Dulac-Arnold et al. 2015) and Deep Hash embedding (Kang et al. 2021) use a deep neural network to encode features into real-valued embeddings. In a changing vocabulary setup, the drawback is the need to modify the whole neural network to incorporate new string features, even though there is only one new feature. Different from previous works, our method emphasizes the usage of hash embeddings in dynamic tabular data with changing vocabularies. In the meantime, the model architecture remains stable, and only partial parameter updates are required.

Generative models for tabular data. Recent research on generative models of non-temporal tabular data focuses on modeling multi-modality or heterogeneity but overlooks the sustainable representations for dynamically expanded vocabularies. These works rely on one-hot encoding for categorical features. Xu et al. (2019) learns VAE and GAN-based tabular data generator while conditioning on discrete categorical features. Later works rely on GAN to design tabular data generators (Liu et al. 2023; Zhao et al. 2021). Kotelnikov et al. (2023) extend diffusion models to tabular data.

Temporal tabular data models. To our knowledge, there isn't a sequence model designed for multi-task temporal tabular data, although some previous works have the potential to extend to tabular data. PHE extends Deep Kalman Filters (Krishnan, Shalit, and Sontag 2015) to be applicable for multi-task, temporal, and dynamic tabular data, while the original Deep Kalman Filters do not explicitly consider the multi-task and dynamic vocabulary property of the tabular data. Girin et al. (2021) survey a list of latent variable sequence models that are possible to be extended to tabular data, although most of them are designed for speech or video data.

Others. The setup of learning dynamic tabular data with changing vocabularies shares the similarity to continual learning and Bayesian online learning (Kirkpatrick et al. 2017; Wang et al. 2023; Zenke, Poole, and Ganguli 2017; Nguyen et al. 2018; Li et al. 2021), but the difference is our formulation is a novel dictionary- or vocabulary-incremental setup for tabular data. Besides,

Table 3: A tabular data snippet from the Retail dataset. The columns are either categorical, numeric, or timestamp. The rows corresponds to sale records. `StockCode` stores product ID. `Quantity` stores the sales. “?” denotes missing values. The task is to predict the sales for each product.

StockCode	Date	UnitPrice	CustomerID	Country	Quantity
85123A	2010-12-01 08:26:00	2.55	17850	United Kingdom	6
84406B	2010-12-01 08:26:00	2.75	17850	United Kingdom	6
21724	2010-12-01 08:45:00	0.85	12583	France	12
21791	2010-12-01 10:03:00	1.25	12431	Australia	12
22139	2010-12-01 11:52:00	0.55	?	United Kingdom	56

Kireev et al. (2023) learn transferable robust embeddings for categorical features. Yin et al. (2020); Iida et al. (2021) design objective functions for representation learning on tabular data using large-language models. Arik and Pfister (2021) and Huang et al. (2020) use the one-hot encoder to learn categorical feature embeddings before input to a transformer module.

Discussions on alternative designs and shortcomings. We acknowledge that alternative solutions may exist, e.g., encoding string features with a character-level recurrent neural network or using a popularity-based token-level one-hot encoder. In our considered aspects, for example, long-tailed data distributions are commonly seen in applications, probabilistic hash embedding stands out with simplility and continual learning capability. Hash features (Weinberger et al. 2009; Cheng et al. 2023) is memory inefficient. Incremental one-hot embeddings are also inefficient for dynamic tabular data, because the model parameters expand unbounded, resulting in storage inefficient and slow computation. Deep hash embedding (Kang et al. 2021) and other methods in the same fashion are computationally inefficient. One needs to adapt the whole neural network even when adding one new category. In contrast, one only needs to adapt the corresponding embeddings in probabilistic hash embedding.

Handling hashing value collisions. Collision of hash values could happen among popular, important categories. To address this issue, we can select the desired hash functions that avoid important collisions before applying the hash functions. In addition, users come and go fast, and collisions may become unimportant over time.

E An example tabular data of changing categorical features

We will explain the concepts related to this work through an example tabular data snippet (Tab. 3). Tabular data contains two dimensions–rows and columns. Any stored information can be located by specifying the row and column indices. We can classify columns into three types: *categorical*, *numeric*, and *timestamp*. A categorical column represents a discrete nominal feature, usually recorded in text strings and therefore hashable; A numeric column corresponds to a numeric feature, usually represented by float or integer values; and a timestamp column records the timestamp when a row is created. For instance, in Tab. 3, there are six columns, among which `StockCode`, `CustomerID`, and `Country` are categorical columns, `UnitPrice` and `Quantity` are numeric columns, and `Date` is a timestamp column. Some columns are of particular interest and one may want to predict those based on others. We refer to those columns as *predicted columns*. Predicted columns can be either categorical or numeric, depending on task requirements. Rows with similar timestamps usually exhibit correlations. But these correlations may change over time.

Some tabular data is multi-task-oriented. For example, in Tab. 3, one may be interested in predicting future selling quantity based on historical transactions for each product. In this case, different product IDs in `StockCode` suggest different tasks. We refer to the categorical columns consisting of task identifiers as *global columns* and other categorical columns as *local columns*. We express this type of tabular data *multi-task*. Each task may have specific column relationships.

All unique items in a categorical column constitute its *vocabulary*. When new items join into the column, we say it has a *changing* or *dynamic vocabulary*. [4]

F A simple example to understand why PHE is superior to DHE

In this section, we consider a simple linear Gaussian model that we can analyze in closed form to illustrate why having deterministic hash embeddings that are updated in an online fashion is prone to *forgetting*. The crux of our calculations is the fact that distinct categorical items share representations due to partial hash collisions. Thus, when trained online, the shared features shows a bias to work well for the categorical item that was most recently seen, rather than be optimized for the overall data distribution seen so far, leading to the forgetting behaviour. However, we will show that Bayesian hash embedding does not suffer this, because it is well known that if exact online posterior can be computed (which in our linear Gaussian setup is easy to do), the online posterior is identical to the offline one.[5]

[4] We assume the tabular structure is fixed, i.e., the number of columns, column names, and types are fixed. We also assume categorical features are single-valued. But our work is compatible with multi-valued features.

[5] Note that this section has a slightly different notation from the main text, but the content is self-contained. Readers can also match the notation by noting $X := \mathbf{m}$ and the input variable value has $0 := \mathbf{m}_0$ and $1 := \mathbf{m}_1$.

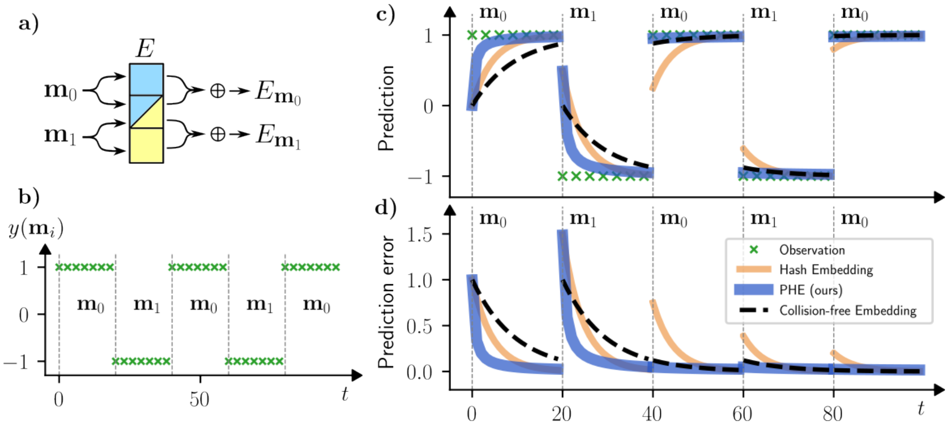

Figure 5: Forgetting in online learning using deterministic hash embedding on synthetic data. (The complete setting is described in Sec. 3.4.) The task is predicting a scalar (regression problem) with the covariate being a categorical variable that takes one of two values of $\mathbf{m}_0$ or $\mathbf{m}_1$. **a)** shows the embedding matrix E of size 3×1. Here the number of buckets $B = 3$ and $d = 1$. The two hash function maps $\mathbf{m}_0$ to 0 and 1 respectively and maps $\mathbf{m}_1$ to 1 and 2 respectively. **b)** shows the online samples where the covariate alternates between $\mathbf{m}_0$ and $\mathbf{m}_1$ and the corresponding target $y(\mathbf{m}_i)$ takes values in 1 and -1. **c)** shows the prediction of a probabilistic hash embedding table (blue) trained using Bayesian online learning and a deterministic hash embedding (DHE) table (yellow) trained using online gradient descent. **d)** plots the prediction error. From these figures we observe that PHE's prediction error converges to 0 much quicker than DHE. After every 20 samples when the covariate changes, there is a big jump in DHE error, exhibiting forgetting while the PHE has no error spikes after it has encountered both the categorical values.

A simple linear-Gaussian model

Consider a simple situation of regression with input variable $X \in \{0, 1\}$ taking one of two categorical values and the target $Y \in \mathbb{R}$ is real-valued. The conditional distribution of Y is a gaussian distribution with the mean being 1 when $X = 0$ and mean being -1 when $X = 1$. We further assume that the variance of Y is $\sigma^2 \approx 0$ is tiny, In notation terms, the true distribution of $Y|X = 0 \sim \mathcal{N}(1, \sigma^2)$, while the distribution of $Y|X = 1 \sim \mathcal{N}(-1, \sigma^2)$, where σ is a fixed and small. We do not specify the distribution of the covariate X just yet and defer that to the sequel.

The predictive model based on hash embedding

Given labeled data (X, Y), we aim to learn a predictor $f(X)$ that predicts Y given X. To build the predictor we use a simple hash embedding model. Specifically, we assume that the predictor $f(\cdot)$ is parameterized by a 3×1 embedding matrix E. Although technically this is a vector, we still denote it as an 'embedding matrix' to be consistent with the rest of the exposition. Denote by $e^{(0)}, e^{(1)}$ and $e^{(2)}$ as the three rows of this matrix which are the 'embedding vectors' of the three hash values. Thus, in the notation of our model, this embedding matrix is made of $B = 3$ buckets with the dimension $d = 1$. The model $f(\cdot)$ uses two hash functions $h_i(\cdot) : \{0, 1\} \to \{0, 1, 2, \}$ to map the categorical variable X into a hash value. Without loss of generality, we assume that $h_1(0) = 0, h_2(0) = 1, h_1(1) = 1, h_2(1) = 2$. Given this, the predictive model $f(X) := e^{(h_1(X))} + e^{(h_2(X))}$ is a simple linear sum of the two hash embedding of the input based on the two hash functions $h_1(\cdot)$ and $h_2(\cdot)$. This is a simple example of the general class of models where the predictor Y is a linear function of the embedding vectors of the categorical input X computed using the different hash functions. Although simple, this example illustrates the phenomenon that emerges of learning categorical variables in an online fashion since the embedding vector $e^{(1)}$ influences both $X = 0$ through hash function $h_1(\cdot)$ and $X = 1$ through hash function $h_2(\cdot)$.

An online interaction setting

We consider the following online prediction protocol. At each time $t = 1, 2, \cdots$, the environment samples X_t from a distribution over $\{0, 1\}$ and produces to the predictor. The predictor then predicts $\widehat{Y}_t := f_t(X_t)$ and is then shown the true label $Y \in \mathbb{R}$. The predictor incurs loss $l_t := \frac{1}{2}(Y_t - \widehat{Y}_t)^2$ and uses the observed Y_t to update the predictor to $f_{t+1}(\cdot)$.

The only learnable parameters of the predictor is the embedding matrix E. Thus the predictor at time t denoted by $f_t(\cdot)$ is parametrized by the state of the embedding matrix E_t with its three rows denoted by $e_t^{(i)}$ for $i \in \{0, 1, 2\}$.

Update the hash embedding matrix through Online Gradient Descent (OGD)

In order to demonstrate that the hash embeddings can lead to forgetting, we will assume that they are updated through standard online gradient descent. Observe that at time t, if $X_t = 0$, then $\widehat{Y}_t = e_t^{(0)} + e_t^{(1)}$. The instantaneous loss at time t is given by $l_t = \frac{1}{2}(\widehat{Y}_t - Y_t)^2$. Thus, the gradients $\frac{\partial l_t}{\partial e^{(0)}} = \frac{\partial l_t}{\partial e^{(1)}} = (e_t^{(0)} + e_t^{(1)} - Y_t)$, if $X_t = 0$. Thus, assuming that the embedding matrix

E_t is updated online using OGD at a fixed learning rate $\eta \in \mathbb{R}$ leads to the following update equations

$$e_{t+1}^{(0)} = \begin{cases} e_t^{(0)} - \eta((e_t^{(0)} + e_t^{(1)} - Y_t)), & X_t = 0 \\ e_t^{(0)}, & X_t = 1. \end{cases}$$

Similarly the update equations for the other two embedding vectors are as follows.

$$e_{t+1}^{(1)} = \begin{cases} e_t^{(1)} - \eta((e_t^{(0)} + e_t^{(1)} - Y_t)) & X_t = 0 \\ e_t^{(1)} - \eta((e_t^{(1)} + e_t^{(2)} - Y_t)) & X_t = 1 \end{cases}$$

$$e_{t+1}^{(2)} = \begin{cases} e_t^{(2)} & X_t = 0 \\ e_t^{(2)} - \eta((e_t^{(1)} + e_t^{(2)} - Y_t)) & X_t = 1 \end{cases}$$

These update equations for the embedding shows that $e^{(1)}$ which is shared for both $X = 0$ and $X = 1$ gets updated all the time, while $e^{(0)}$ is only updated if $X = 0$ and similarly $e^{(1)}$ is only updated if $X = 1$.

A non-stationary distribution for the co-variates X

Consider a setting where the first N inputs consists of $X_t = 0$ for all $t \in \{1, \cdots, N\}$, followed by another N inputs consisting of $X_t = 1$ for all $t \in \{N+1, \cdots, 2N\}$. In these discussions we will assume N is large enough and the learning rate η is appropriately tuned to make the variance of the predictor to be small. If all the $2N$ samples were shown to a training algorithm, it could have (near) perfectly estimated the embedding matrix $\widehat{E}$, i.e., for a X that is sampled from $\{0, 1\}$ that is equally likely (matching the training data distribution of equal number of 0 and 1), the expected excess loss will be arbitrarily small (assuming N is sufficiently large). We will show in the calculations below that if instead the embedding matrix was learnt using OGD, even if N is large enough, the learnt model at the end will have a constant excess risk when the test input X is sampled with equal probability among $\{0, 1\}$.

Analyzing the OGD update equations

To see this, we make some simplifying assumptions. First is that $\sigma = 0$, i.e., conditioned on X, Y is deterministic. Second is a symmetric starting point of $e_0^{(i)} = 0$ for all $i \in \{0, 1, 2\}$. It is easy to observe that both of these assumptions do not change the the observation we will make, but makes the exposition easier. Thus, at the end of the first N samples, we will have $e_{N+1}^{(2)} = 0$ and $e_{N+1}^{(0)} = e_{N+1}^{(1)} \approx 1/2$. This follows as N is large and the noise σ is 0, thus leading OGD to converge to a local minima of the loss function. Any embedding matrix with $e^{(0)} + e^{(1)} = 1$ is a local-minimum of the loss function and thus at the end of time $N+1$, OGD will result in $e_{N+1}^{(0)} + e_{N+1}^{(1)} \approx 1$. Since the initialization and the loss function is symmetric in the arguements $e_t^{(0)} = e_t^{(1)}$ will hold for all $t \leq N$.

At time $t = N+1$, the N observed samples corresponds to $X = 0$. Thus, the prediction error for $X = 0$ by this learnt model $\widehat{f}_{N+1}(X)$ is small, i.e., the excess risk $(f_{N+1}(X) - 1)^2 \approx 0$.

Now consider the times $t = N+1$ till $t = 2N$. During this period, the gradients will not impact $e^{(0)}$, i.e., $e_{N+1}^{(0)} = e_{2N+1}^{(0)} \approx 1/2$. However, $e^{(2)}$ and $e^{(3)}$ are no longer symmetric. But one can work out the recursion for their evolution since the gradients are the same.

In particular, for any time $t \in \{N+1, \cdots, 2N\}$, the observed $X_t = 1$. Thus, the gradient of $e^{(1)}$ and $e^{(2)}$ at all times $t \in \{N+1, \cdots, 2N\}$ is the equal to $(e_t^{(1)} + e_t^{(2)} + 1)$. Thus, under the OGD update equations, for all times $t \in \{N+1, \cdots, 2N\}$, the equality $e_{t+1}^{(1)} - e_{t+1}^{(2)} = e_t^{(1)} - e_t^{(2)}$, holds. Since at time $N+1$, we have $e_{N+1}^{(1)} \approx 1/2$ and $e_{N+1}^{(2)} = 0$, we have that $e_{2N+1}^{(1)} - e_{2N+1}^{(2)} \approx 1/2$. On the other hand, if N is large, we know that OGD will converge to a local minima, i.e., $e_{2N+1}^{(1)} + e_{2N+1}^{(2)} \approx -1$. These two equations in the variables $e_{2N+1}^{(1)}, e_{2N+1}^{(2)}$ gives $e_{2N+1}^{(1)} \approx -1/4$ and $e_{2N+1}^{(2)} \approx -3/4$.

Concluding that the updates leads to forgetting the representation for $X = 0$

Thus at the end at time $2N+1$, after having seen the first N samples of $X = 0$ and the last N samples of $X = 1$, the predictor is such that $\widehat{f}_{2N+1}(0) \approx 1/4$ and $\widehat{f}_{2N+1}(1) \approx -1$. However, note that the true label when $X = 0$ is 1 while when $X = 1$ is -1. Thus, the predictor $\widehat{f}_{2N+1}(\cdot)$ has near zero prediction error when $X = 1$. However, when $X = 0$, the loss given by $(\widehat{f}_{2N+1}(0) - Y)^2 \approx (1/4 - 1)^2 \approx 9/16$ is a constant.

This shows the discrepancy between a model trained offline using all the $2N$ samples and the model trained online where the first N samples all correspond to $X = 0$ and the last N samples correspond to $X = 1$. The offline model will converge to a local minima in which the prediction error for both $X = 0$ and $X = 1$ will be small, while the online model converges to a solution where the prediction error for the categorical variable that was not seen recently is high.

Arguing that online Bayesian model does not lead to forgetting

A Bayesian method to 'learn' the embedding matrix is to posit a prior distribution $p(E)$ for the emebedding matrix and then given the data $\mathbf{X}$ compute the posterior distribution $p(E|\mathbf{X})$. We will say that the Bayesian learning does not forget, if the posterior distribution computed based on all the $2N$ samples $(X_1, Y_1), \cdots, (X_{2N}, Y_{2N})$ shown up-front matches the posterior distribution computed in an online fashion. However, from classical results in online Bayesian learning, it is well known that if one can compute the exact posterior $p(E|X_1, \cdots, X_t)$ at all times t, then the posterior at time $2N$ is identical to the one that an offline algorithm would have computed had it seen all the $2N$ samples at once. Thus, if the exact posterior can be computed at each time, then there is no forgetting in the Bayesian mechanism.

Thus in this section, we showed through a simple linear-gaussian model, that online updating of hash embedding matrix leads to forgetting while a bayesian updating of the embedding matrix does not lead to forgetting. In order to demonstrate this, we defined forgetting to not occur if the model learnt at the end of seeing each online sample one by one is close to the model learnt had all the samples been available up-front. Further, we show in experiments that this insight holds even in more complex scenarios where exact Bayesian posterior cannot be computed, but only an approximation through variational inference can be done.

G Experimental Details

G.1 An efficient embedding fetch schemes

When implementing the hash embedding fetching module, there are two available schemes: scheme one is first to sample a whole hash table E and then fetch the corresponding embeddings $E_{\mathbf{h_s}}$ (as Eq. (26)); scheme two is first to fetch the distribution $p(E_{\mathbf{h_s}})$ and then sample $E_{\mathbf{h_s}}$ (as Eq. (27)).

$$\begin{aligned} p(\mathbf{x}|\mathbf{s}) = p(\mathbf{x}|\mathbf{h_s}) &= \mathbb{E}_{p(E)}[p(\mathbf{x}|E, \mathbf{h_s})] \approx p(\mathbf{x}|E, \mathbf{h_s}) \qquad (26) \\ &= \mathbb{E}_{p(E_{\mathbf{h_s}})}[p(\mathbf{x}|E_{\mathbf{h_s}})] \approx p(\mathbf{x}|E_{\mathbf{h_s}}) \qquad (27) \end{aligned}$$

The two schemes lead to the same results, but scheme two is more memory-efficient as it does not need to sample the whole embedding table. Thus in practice, we apply Eq. (27).

G.2 Hardware Information

We train and test our model on GPUs (RTX 5000) and use the deep learning framework PyTorch to enable efficient stochastic backpropagation. In all supervised learning experiments, the total elapsed wall time (training and testing) for PHE is less than half an hour, and the finetune baseline runs slightly faster. In the sequence modeling experiments, PHE runs about one hour since Retail is a large dataset and has over 500k records. In the recommendation experiments, it takes about two hours for all methods.

G.3 Details for Classification Experiments

The four public datasets all can be found online: Adult[6], Bank[7], Mushroom[8], and Covertype[9]. Specifically, Adult has 14 columns and 48,842 rows containing demographic information. The task is to predict whether or not a person makes over $50K a year; Bank has 16 columns and 45,211 rows to predict if a client will subscribe to a term deposit; In Mushroom, of 22 discrete columns and 8,124 rows, the goal is to predict whether a mushroom is poisonous; Covertype, involving 12 columns and 581,012 rows, is to predict which forest cover type a pixel in a satellite image belongs to.

Regarding model architecture, we concatenate all category embeddings as well as continuous features as input to a deterministic one-layer neural network, followed by a softmax activation function. For PHE and P-EE, we stress that only embeddings are probabilistic and neural network weights are deterministic. We use negative cross entropy as the objective function assuming the targets follow categorical distributions.

We apply the following criterion when selecting a categorical column to have a dynamic vocabulary. We select the column to be dynamic if the weights of the column features have large scales when fitting a logistic regression model on the outputs. Specifically, we first use one-hot encodings to represent categorical items, and then fit a logistic regression model on the targeted outcomes. Finally, we select a column to be incremental if its corresponding categorical features have large weights because the weights in linear regression models can be interpreted as feature importance. Following this procedure, we select `education`, `poutcome`, `odor`, and `wilderness` column for the four datasets respectively. See detailed group information in Fig. 9.

For the continual learning setup in Supp. G.6, we first randomly and evenly split the categorical features of the selected column into disjoint groups, then partition the original dataset according to the groups. Based on the column dictionary size, we split Adult/Bank/Mushroom/Covertype into five/four/four/four disjoint groups. We randomly split each group into training and testing subsets where the training subset takes two-thirds of the total data and the testing subset takes the remaining one-third. We sequentially fit the prediction model to each non-overlapped group. The goal is to have high accuracy for all groups after sequential updates. Therefore, after fitting the model on the current group's training data, we report the average accuracy on all previous groups' test data.

[6] https://archive.ics.uci.edu/dataset/2/adult

[7] https://archive.ics.uci.edu/dataset/222/bank+marketing

[8] https://archive.ics.uci.edu/dataset/73/mushroom We also follow the recommendation and only use `odor` as the feature.

[9] https://archive.ics.uci.edu/dataset/31/covertype

Evidence lower bound. We first present the objective function of the latent variable supervised learning model. Similarly to Eq. (12), we can derive the objective function as the evidence lower bound of $\sum_{i=1}^{N} \log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{h}_{\mathbf{s}_i}; \theta)$:

$$\mathcal{L}(\theta, \lambda) = \mathbb{E}_{q_\lambda(E)} \left[\sum_{i=1}^{N} \log p(\mathbf{y}_i|\mathbf{x}_i, E_{\mathbf{h}_{\mathbf{s}_i}}; \theta) \right] - D_{\mathrm{KL}}(q_\lambda(E)|p(E)) \tag{28}$$

For online adaptation to dataset $\mathcal{D}_1$ of size N_1, we fix the classifier parameters and only adapt the hash embedding table E. Denote the pre-trained parameters by θ^* and λ_0^*. Treat the previous posterior $q_{\lambda_0^*}(E)$ as the current prior, we can write down the objective function

$$\mathcal{L}^{(1)}(\lambda; \theta^*, \lambda_0^*) = \mathbb{E}_{q_\lambda(E)} \left[\sum_{i=1}^{N_1} \log p(\mathbf{y}_i|\mathbf{x}_i, E_{\mathbf{h}_{\mathbf{s}_i}}; \theta^*) \right] - D_{\mathrm{KL}}(q_\lambda(E)|q_{\lambda_0^*}(E)) \tag{29}$$

Implementation details and hyperparameters. We implement the aggregation function g as a weighted sum where the weights are parameters of g. Specifically, we have another random table $W \in \mathbb{R}^{P \times K}$ whose distribution is $p(W)$ and a hash function $h^{(W)} : \mathcal{S} \to \mathbb{N}_{<P}$ such that $h_{\mathbf{s}}^{(W)}$ indexes the rows of W, noted by $W_{h_{\mathbf{s}}^{(W)}} \in \mathbb{R}^K$. $W_{h_{\mathbf{s}}^{(W)}}$ serves as the weights for the K hash embeddings (see Fig. 2). Then $g(E_{h_{\mathbf{s}}^{(1)}}, \ldots, E_{h_{\mathbf{s}}^{(K)}}) = \sum_{k=1}^{K} W_{h_{\mathbf{s}}^{(W)}}^{k} E_{h_{\mathbf{s}}^{(k)}}$ where $W_{h_{\mathbf{s}}^{(W)}}^{k}$ is the kth value of vector $W_{h_{\mathbf{s}}^{(W)}}$. During inference, we infer the posteriors of both E and W.

For all tabular datasets except Mushroom, we set $B = 7, K = 3, d = 20, P = 11$ (whose supported dictionary size is $P \times B^K = 3773$, which is ten times larger than the vocabulary size of the Adult dataset). We tried these values on Adult when setting the group size to be one (i.e., the static supervised learning setup) and found the resulting accuracy (about 84%) is comparable to the public results on this dataset[10]. We then use this same parameter setup on all other tabular data supervised learning experiments. For Mushroom, we use a much smaller model size and set $B = 5, K = 3, d = 5, P = 1$, because only one feature is used in the experiment.

Optimization. We use Adam stochastic optimization with a learning rate of 0.01 and a minibatch size of 128 in all experiments for both our method and baselines. For other hyperparameters of Adam, we apply the default values recommended in the PyTorch framework. When selecting these values, we fixed the minibatch size 128 and searched the learning rate (0.001, 0.005, 0.01, 0.05, 0.1) on the Adult dataset. We found the learning rate 0.01 leads to relatively fast and stable convergence. Then we apply the same values on all other datasets. For the first group training, we train PHE 100 epochs; for the remaining groups, we train PHE 15 epochs as we only need to update the hash embedding table E. Note that on every group, we train PHE until convergence.

Evaluation metric. We use accuracy as an evaluation metric. As we sequentially adapt the model on each vocabulary group's training set and test the model on the test set, we have running accuracies on each group.

Additional results. We add all datasets' online learning results in Fig. 6.

G.4 Details for Multi-Task Sequence Modeling Experiments

Datasets. We use the Retail dataset[11] as a multi-task TTD to demonstrate PHE. The dataset involves over 4,000 products indicated by `StockCode` column and the corresponding sale quantities represented by `quantity` column with invoice timestamps. We treat `quantity` as a time series and then track `quantity` for all 4,000 selling goods over time in a filtering setup. Prediction for each piece of product is regarded as one task and there are over 4,000 tasks in total. The task is to predict the sales quantity for the product shown in each invoice record given the product's previous sales.

For the continual learning setting in Supp. G.6, we treat all transactions as occurring at even time intervals. For each task, we randomly split the training and testing set with a ratio of 2:1. To get multi-tasks in a dynamic setting, we treat `StockCode` as the task identifier and evenly partition the products in `StockCode` into ten disjoint groups where each group involves about 400 goods, i.e., 400 new tasks. Correspondingly, the original dataset is converted into a task-incremental dataset where each task refers to predicting sale quantities (i.e., taking `Quantity` column values as $\mathbf{y}$) for one product, indicated by `StockCode` column. We normalize the `UnitPrice` column into the range $[0, 1]$ and do not use the `Description` column. We also drop cancellation transactions that have `Quantity` values smaller than zero. Therefore, we refer to `StockCode` as $\mathbf{u}$, `Quantity` as $\mathbf{y}$, `UnitPrice` as $\mathbf{x}$, $\{$`Country`, `CustomerId`$\}$ as $\mathbf{m}$, and `InvoiceDate` as t.

Evidence lower bound. We assume the sales quantity follows Poisson distribution, consequently using the Poisson likelihood.

Implementation details and hyperparameters. We also implement a weighted aggregation function g as above in the supervised learning setup. We did not try out different hyperparameter settings and directly set $B = 109, K = 3, d = 20, P = 109$ as these values can already support a large vocabulary (of size $P \times B^K$). We apply the same values to both PHE and the baselines.

[10] See the baseline model performance in https://archive.ics.uci.edu/dataset/2/adult

[11] http://archive.ics.uci.edu/dataset/352/online+retail

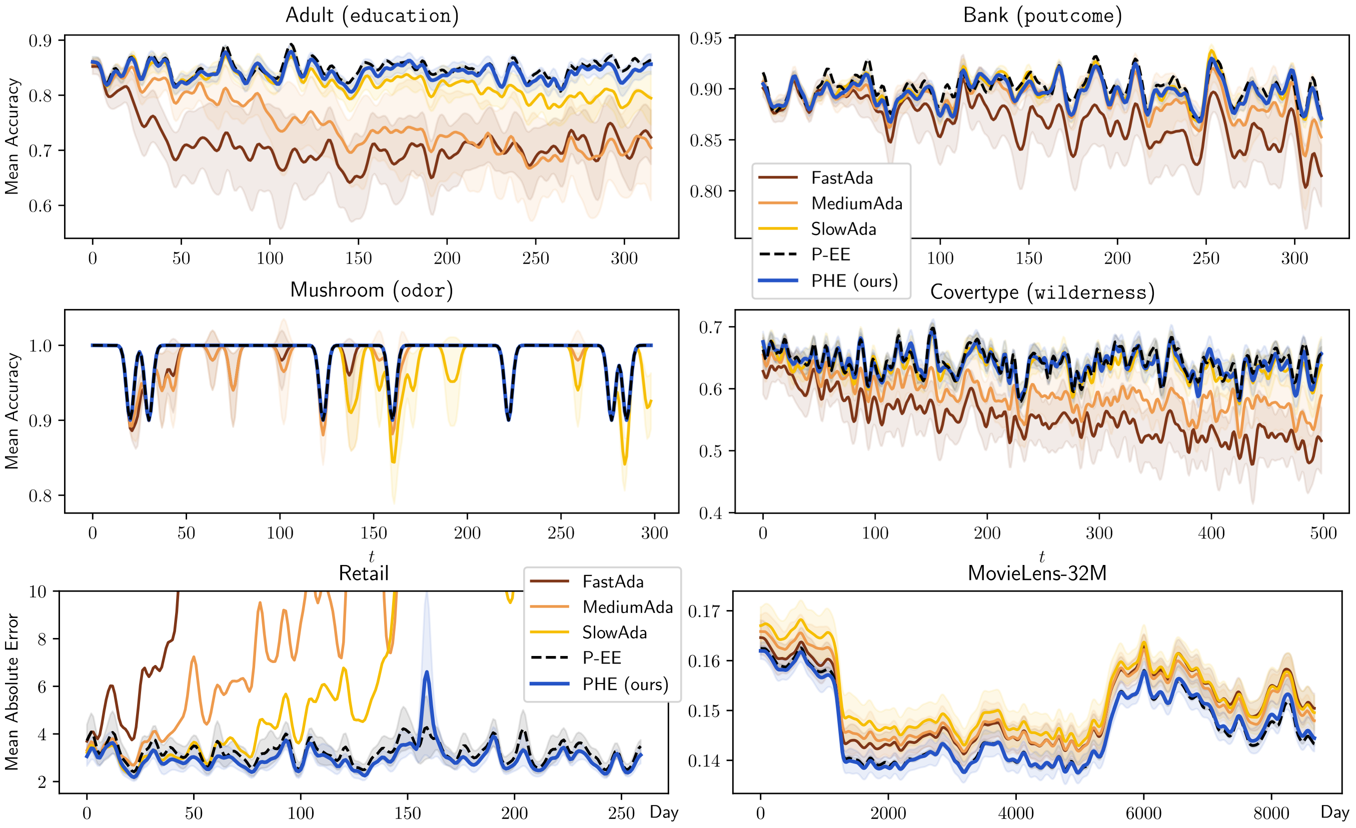

Figure 6: Results of online learning on all data.

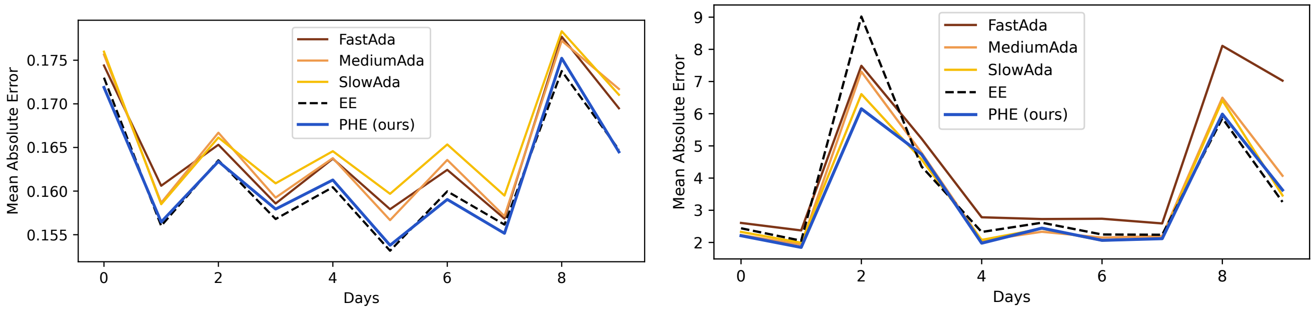

Figure 7: First ten day results of data-streaming movie recommendation and sales quantity sequence modeling.

Optimization. We use Adam stochastic optimization with the same learning rate of 0.005 and the same minibatch size of 128 as in supervised learning experiments. For other hyperparameters of Adam, we apply the default values recommended in the PyTorch framework. For the first task training, we train PHE 15 epochs; for the remaining tasks, we train PHE 5 epochs. Note that on every task, the epochs used are enough to train PHE until convergence.

Evaluation metric. We also evaluate the performance by the cumulative averages of errors. For each product, we use the first nine observations to predict the 10th observation and measure the absolute error on the 10th observation. Then, the average of all such absolute errors is the performance of this product. Since one group contains about 4,000 products, we further average each product's performance as the group's performance. Specifically, we have a prediction model that has a Poisson likelihood $p(\mathbf{y}_t|\mathbf{y}_{t-9:t-1}, \mathbf{x}_{t-9:t}, \mathbf{h}_{\mathbf{m}_{t-9:t}}, \mathbf{h}_{\mathbf{u}})$. We predict $\hat{\mathbf{y}}_t = \mathbb{E}[\mathbf{y}_t|\mathbf{y}_{t-9:t-1}, \mathbf{x}_{t-9:t}, \mathbf{h}_{\mathbf{m}_{t-9:t}}, \mathbf{h}_{\mathbf{u}}]$ as the mean value and then measure the absolute error between the ground-truth value $|\mathbf{y}_t - \hat{\mathbf{y}}_t|$.

For the continual learning setup, after learning group t, we can evaluate the performance of all previous and current groups, denoted by $R_{t,\leq t}$. We refer to the cumulative mean absolute error $\bar{R}_t = \sum_{a=1}^{t} R_{t,a}/t$ at group t as the performance at t. We report $\bar{R}_t$ as a function of group numbers in Fig. 11. We report $\bar{R}_T$ after learning the final group T in Tab. 5.

Additional results. Because we smoothed the results with a 1-D Gaussian filter in the main paper, we provide the first ten days' result without smoothing in Fig. 7.

G.5 Details for Recommendation Experiments

Beside the first five years, this up-to-date and largest MovieLens dataset [12] have 8688 days (time steps) with possibly no records on some days. Note after pre-training, all model parameters are fixed except the hash embeddings. Regarding the likelihood function, we assume the rating follows Gaussian distribution.

We randomly split the data into a validation (20%) and a test set (80%). We searched the learning rate, batch size, neural network size, and likelihood scale on the validation set and reported final results on the test set. PHE, EE, and FastAda train the hash embeddings for 5 epochs per time step while MediumAda trains 2 epochs and SlowAda trains 1 epoch.

We also use the mean absolute error as the evaluation metric.

Results. We plot the online learning results in Fig. 6. The curves in are smoothed with a 1-D Gaussian filter. The initial performance gap on Day 0 is an artifact of smoothing, in fact, all methods have similar performance initially (see Fig. 7 in Supp. G.5). An interesting observation: MovieLens made two major changes in their rating system around 2003 and 2014 https://grouplens.org/blog/movielens-datasets-context-and-history/(link), which is reflected in our results – two sharp changes in the online learning plot.

G.6 Additional Results

More Motivation Examples We report additional results in Fig. 8 as a complement to Fig. 1 in the main paper. Fig. 8 provides more evidence for the motivation of our work. For tabular data in a dynamic setting, not including the newly created categorical feature values in the prediction model will lead to a performance drop. Therefore, an efficient way to incorporate the new categorical features is necessary to maintain the efficacy of a prediction model. The "After update" performance in the plots demonstrates PHE is desirable for adapting to the new features. The splitting details are in Supps. G.3 and G.4 and Fig. 9.

Memory Efficiency Memory efficiency of PHE can be seen from the number of parameters in the embedding module, which we summarized for both PHE and P-EE in Tab. 4. Note that P-EE sets the performance upper bound but its size scales linearly with the vocabulary size. The fact that PHE on all datasets achieves the same performance as P-EE illustrates PHE's impressive memory efficiency, especially considering PHE only consumes as low as 2% memory of P-EE. Besides, being a

[12] https://files.grouplens.org/datasets/movielens/ml-32m.zip

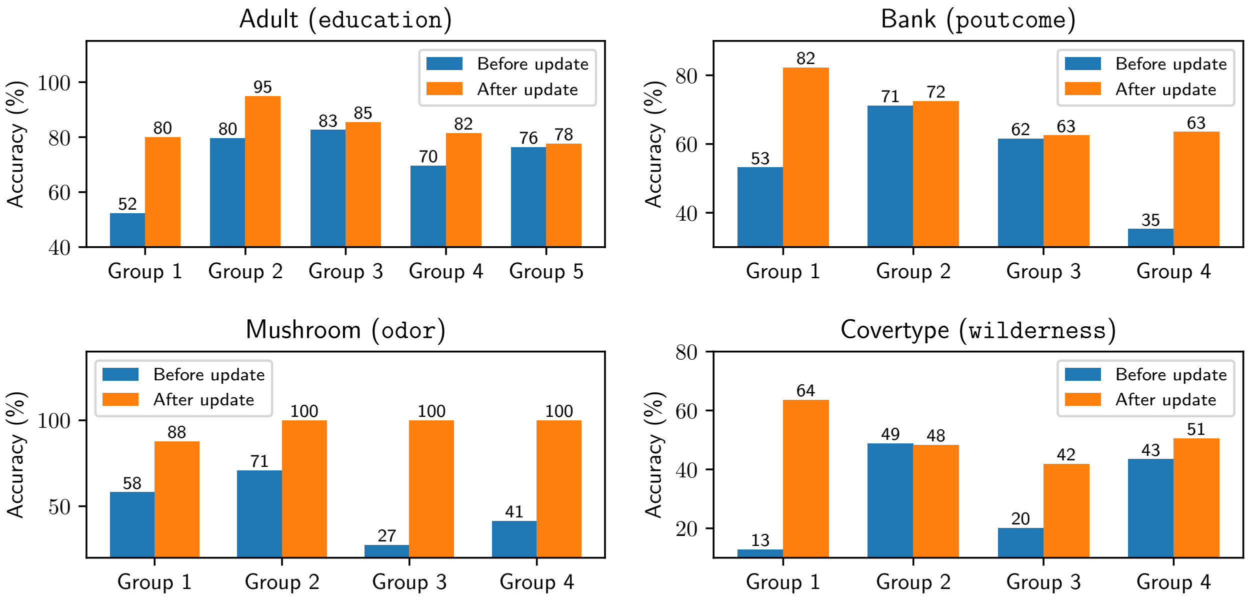

Figure 8: Adult dataset is randomly split into disjoint groups based on the `education` column. Groups arrive sequentially. We report results before and after the updates on the hash embeddings for each group to motivate the need to incorporate new groups into the model. Results are averaged on five independent runs with different random parameter initializations.

Adult

Group 1	`(Preschool, 5th-6th, Bachelors)`
Group 2	`(10th, 11th, 12th)`
Group 3	`(7th-8th, HS-grad, Prof-school)`
Group 4	`(9th, Assoc-voc, Doctorate)`
Group 5	`(1st-4th, Masters, Some-college, Assoc-acdm)`

Mushroom

Group 1	`(Musty, None)`
Group 2	`(Anise, Almond)`
Group 3	`(Spicy, Creosote)`
Group 4	`(Foul, Fishy, Pungent)`

CoverType

Group 1	`(A1)`
Group 2	`(A3)`
Group 3	`(A4)`
Group 4	`(A2)`

Bank

Group 1	`(Unknown)`
Group 2	`(Failure)`
Group 3	`(Other)`
Group 4	`(Successs)`

Figure 9: Group information for continual classification tasks.

Table 4: Number of parameters in the embedding module. Ratios are computed by dividing PHE by P-EE. The results show that PHE consumes as little as 2% of the number of parameters (i.e., hardware memory) of P-EE, demonstrating the memory-efficiency benefit of PHE. (See details in Supp. G.6.)

	Adult	Bank	Covertype	Mushroom	Retail	MovieLens-32M
PHE (ours)	346	346	346	56	5014	460414
P-EE	3920	1760	1760	90	332760	11541320
Compression Ratio	0.09	0.2	0.2	0.62	0.02	0.04

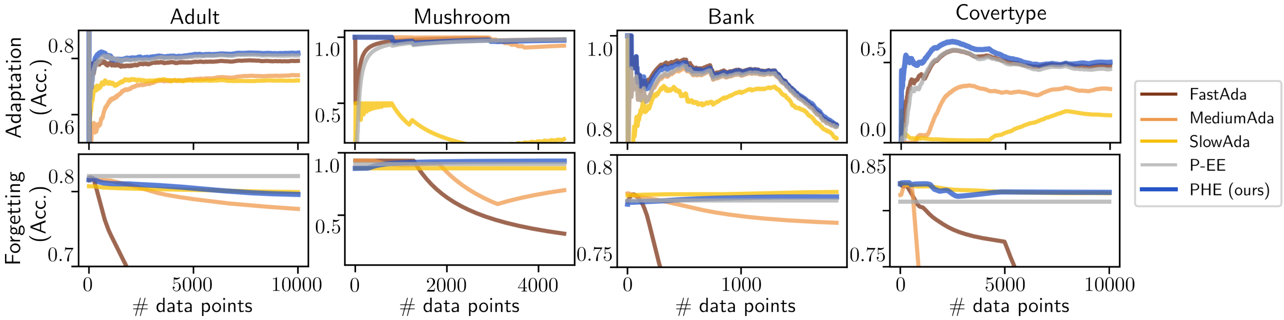

Figure 10: Comparision of online learning methods' adaptation and forgetting in a streaming online setup. Our PHE achieves similar performance with the collision-free P-EE on both metrics. Notably, SlowAda forgets the least but is slow in adaptation; FastAda is in the opposite regime.

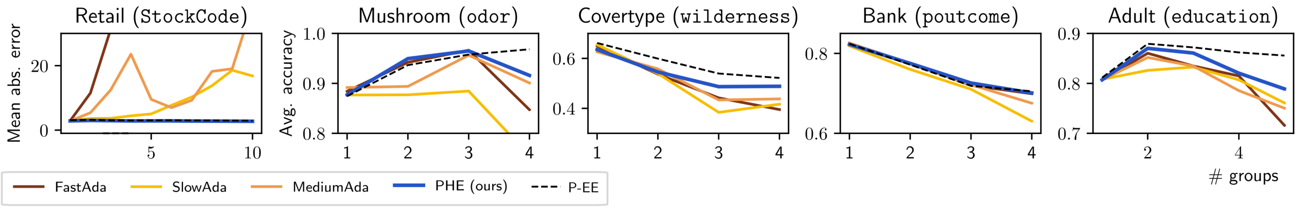

Figure 11: Cumulative average results in continual learning. Column names in the parentheses are the ones made to have changing vocabulary and used to split groups. PHE is closest to the performance upper-bound P-EE.

unified embedding where all categorical columns share the same embedding table (Coleman et al. 2024), PHE is compatible with modern hardware and can benefit from the hardware acceleration.

We multiply each number by two because every parameter has its mean and variance. 20 is due to each embedding has 20 dimensions. For PHE, refer to implementation details (Supp. G) for the number of parameters ($B \times d + P \times K$). We compute the P-EE parameter size by $V \times d$ where V is the vocabulary size.

Adaptation and Forgetting analysis We designed experiments to specifically measure the adaptation to new data and forgetting of old data. We split the data into two disjoint groups based on a random partition of one column's vocabulary. The model was initialized using the first group and online updated on the second group whose items are unseen in initialization. We let the data arrive one at a time. Adaptation is measured by the cumulative predictive accuracy of new datum and the forgetting by the accuracy of the first group's test data. Results in Fig. 10 show that our PHE has almost the best adaptation and forgetting performance on all four datasets. The P-EE while does not suffer forgetting, its adaptation to new categories is slow as each new embedding is initialized at random.

Regarding baselines, SlowAda uses a small learning rate (1e-4); MediumAda uses a medium learning rate (1e-3); FastAda uses a large learning rate (1e-2).

In Fig. 10, we compare on all four classification datasets used in the paper, our PHE against the four baselines. We observe from Fig. 10 that the SlowAda baseline with smaller LR (1e-4) leads to slower forgetting at the cost of slower adaptation, while larger LR (1e-2) has faster adaptation at the cost of faster forgetting (FastAda). Thus a data-stream dependent LR is needed for deterministic hash embeddings to trade off adaptation and forgetting. In contrast, our PHE has almost the best adaptation and forgetting performance on all four datasets due to the regularization from the posteriors. The EE while does not suffer forgetting as each category has a separate row in the embedding table, its adaptation to new categories is slow as each new embedding is initialized at random.

Table 5: Performance on classification and sequence modelling tasks. Adult, Bank, Mushroom, and Covertype are classification tasks and thus evaluated by average accuracy, which is larger the better. Retail is a regression task and we use the metric mean absolute error, lower the better.

	SlowAda	MediumAda	FastAda	P-EE (collision-free)	PHE (ours)
Adult	76.1±1.8	75.0±4.7	71.6±3.1	85.6±0.1	78.9±3.0
Bank	63.0±4.0	67.5±4.5	69.9±1.2	70.5±0.7	70.1±1.4
Mushroom	75.5±7.6	90.1±8.6	84.7±12.3	96.8±0.0	91.6±7.6
Covertype	41.7±4.0	43.8±5.7	39.5±5.1	52.2±1.1	48.8±2.3
Retail	16.8±17.6	38.9±50.9	-	2.92±0.16	2.73±0.23

Table 6: Comparison between updating all categorical columns' embeddings, only updating incremental columns' embeddings, and updating all model parameters. We used collision-free expandable embeddings in the experiments. The first two updating protocols have little difference but updating all parameters sometimes result in performance deterioration, possibly due to catastrophic forgetting in the network weights.

	Adult (Acc.)	Bank (Acc.)	Mushroom (Acc.)	Covertype (Acc.)	Retail (Err.)
Update all columns embeddings	84.7±0.0	90.0±0.0	98.8±0.0	64.1±0.0	3.4±0.3
Update incremental columns embeddings (in use)	84.8±0.0	90.1±0.0	98.8±0.0	64.0±0.4	3.2±0.4
Update all model parameters	83.3±0.1	89.5±0.0	98.8±0.0	64.0±0.1	287.9±125.5

Continual learning We also investigated classification and sequence modeling in the continual learning setup (Kirkpatrick et al. 2017), we split the dataset into disjoint groups based on a random partition of a selected column's vocabulary, assuming data distribution differs conditioned on each partition. This is similar to Supp. G.6. We then sequentially update the embeddings on each group's training data. After each group training, we evaluated the model performance on all previously seen groups' test data. The splitting details are in Supps. G.3 and G.4 and Fig. 9. While data-streaming setup aims to have good performance on the latest task, the goal of continual learning is to perform well on all groups after sequential training. Fig. 11 and Tab. 5 summarizes the results. Our PHE has the top performance among hash embedding methods.

Ablation studies

Variants of updating protocols. We provided evidence on our updating protocols in Tab. 6, showing updating incremental column's embeddings as well as fixing other parameters has the best performance. Tab. 6 presents the accuracy of multiple updating schemes, justifying this updating protocol in use achieves both high accuracy and computational efficiency.

The impact of potential hash collisions and the mitigation measures. We experimented on the large Retail dataset under the continual learning setup as in Supp. G.6. We varied the hyperparameters bucket size B and the number of hash functions K to control the potential number of hash collisions. In particular, we varied one hyperparameter when fixing the other.

We repeated each experiment five times with different random seeds. The tables below show the mean absolute errors (the lower the better) with standard deviation under each hyperparameter setting. In the first table, we varied bucket size B while fixing the number of hash functions to be K=2. In the second table, we fixed the bucket size B to 109, which is the same as in the paper, and changed the number of hash functions. The collision probability increases from right to left for both tables. The results in the first table show the more likely a hash collision, the more unstable the model performance. However, the deterioration is slow, showing the method's robustness to potential hash collisions and various hyperparameter settings. In the second table, although increasing K reduces the probability of hash collisions, increasing K also increases the number of effective parameters (related to model complexity) to fit in the model. It thereby increases the variance of the predictive performance. Thus, we recommend choosing a small K (such as 2-3) that trades off both hash-collision and predictive performance variance. Note when K=1, the hash collision will cause two items to have exactly the same resulting hash embeddings, leading to a high variance among all settings. We will add these results to the ablation section in the revised paper.

Ablation study on bucket size B

B=40,K=2	B=60,K=2	B=80,K=2	B=109,K=2
2.83±0.23	2.65±0.16	2.56±0.10	2.58±0.09

Ablation study on the number of hash functions K

B=109,K=1	B=109,K=2	B=109,K=3	B=109,K=4	B=109,K=5
2.66±0.34	2.58±0.09	2.63±0.16	2.78±0.18	2.76±0.13

Remedy. We use the standard trick of multiple independent hash functions to reduce the collision probability of two unique items. As is standard in universal hashing [Carter and Wegman, 1997], the probability of collision with all K hash functions each hashing into B buckets is proportional to (see section 3.3). Collision of hash values could happen among popular, important categories. To address this issue, we can select the desired hash functions that avoid important collisions before applying the hash functions. In addition, users come and go fast, and collisions may become unimportant over time.

Double memory of Ada baselines. In Tab. 1, Ada baselines do not have the same number (actually half) of parameters as PHE because PHE maintains a pair of mean and variance parameters for each embedding. We conducted another set of experiments on the Ada baselines using the same settings as in Tab. 1, except that we doubled the size of embedding tables, i.e., $B \to 2B$ and $P \to 2P$. This way, both PHE and the deterministic Ada baselines have the same number of parameters, but Ada has much lower collision rates. We report the results in the table below. It shows that: 1) PHE is still the top performer; 2) the average performance gap between the Ada baselines and PHE narrows; 3) no single Ada baseline works well across all datasets; and 4) PHE has smaller performance variation.

Results of double-memory Ada baselines

	SlowAda	MediumAda	FastAda	PHE (ours)
Adult	83.6 ± 0.9	79.9 ± 2.2	76.1 ± 2.5	**84.1 ± 0.2**
Bank	**89.7 ± 0.1**	89.4 ± 0.3	87.7 ± 1.6	**89.6 ± 0.0**
Mushroom	97.9 ± 1.5	98.4 ± 0.5	98.5 ± 0.2	**98.8 ± 0.0**
Covertype	**64.2 ± 0.6**	61.6 ± 1.0	55.8 ± 3.2	**64.3 ± 0.2**
Retail	6.4 ± 0.6	8.2 ± 2.3	-	**3.0 ± 0.2**
MovieLens	15.4 ± 0.1	15.1 ± 0.0	15.2 ± 0.1	**14.7 ± 0.0**