Unum: A New Framework for Network Control

Jiayi Chen *UT Austin*
Nihal Sharma *Capital One*
Debajit Chakraborty *UT Austin*
Saurabh Agarwal *UT Austin*
Jeffrey Zhou *UT Austin*
Aditya Akella *UT Austin*
Sanjay Shakkottai *UT Austin*

Abstract

Modern network control tasks, such as congestion control and adaptive bitrate streaming, require accurate state estimation to adapt to heterogeneous and dynamic network conditions. Current approaches, whether manually engineered or machine learning (ML)-based, often rely on instantaneous or running-average metrics, resulting in imprecise approximations of the true network state. This hinders their ability to capture latent factors, such as application workloads or path dynamics, and adapt to non-stationary environments.

We present Unum, a new framework powered by a unified network state embedder leveraging Transformers' self-attention mechanism and diverse training datasets to learn rich, latent state representations. Unum processes historical RTT-timescale network statistics, models complete current state, and predicts future states using pre-trained embeddings from diverse network scenarios. We develop techniques to augment state-of-the-art controllers with Unum embeddings. Through experiments over real and synthetic settings, we show that using Unum state embeddings improves control performance across tasks, including congestion control and adaptive bitrate streaming.

1 Introduction

Modern network stacks perform various control tasks across application and transport layers to efficiently manage the underlying resources. Among the most well-studied tasks is congestion control, which aims to match sending rates with network capacity for smooth data delivery [3]. In recent years, the diversity of network control tasks has expanded significantly. Emerging applications like live video streaming require adaptive bitrate ladders that dynamically adjust quality levels to match fluctuating network conditions, ensuring seamless user experience [49]. Innovations such as 5G network slicing introduce the need to orchestrate multiple virtual "slices" over shared physical infrastructure, each tailored to specific performance needsA [7].

Unfortunately, network control algorithms underperform today, despite advancements spanning both hand-crafted heuristics and machine learning (ML)-based ones. We argue that this is largely due to their shared limitations in using *state*.

State estimation issues: Network control tasks share a common structure: (1) estimating the current network state (e.g., bandwidth, latency, packet loss, or user demands), often from signals collected at round trip time (RTT) timescales, and (2) using this state information to inform a network control policy that determines appropriate actions.

State estimation quality is pivotal to network control; accurate and robust state representations enable better policy decisions and provide essential feedback for adaptation when performance degrades. Unfortunately, current control approaches – whether manually designed or ML-based – suffer from suboptimal state estimation. They largely capture state through a collection of noisy and instantaneous or coarse running-average metrics derived from network signals. This raises two issues: (1) Due to the limited representational power of instantaneous/running-average metrics, state estimations fail to accurately represent *latent factors*, such as, application workloads, decision-making in other network controllers, or network path conditions. (2) The calculation of the metrics is necessarily delayed (by one or more RTTs) due to delayed network signals. As a result, state estimates are at best *imprecise approximations to the current state*, which determines control performance.

Modern networks are *heterogeneous and non-stationary*: they involve diverse devices, link technologies, and applications with variable requirements operating over network conditions and application workloads that change dynamically. The upshot of the above state estimation issues is that network controls cannot accommodate these network attributes well and often under-perform as we show in §2.

Global State Embeddings: To address this fundamental issue, we advocate a new approach to network control, rooted in a rich network state representation that captures latent behaviors and better approximates current state. We propose Unum, a unified network state embedding framework that relies on the power of the self-attention mechanism, the key element

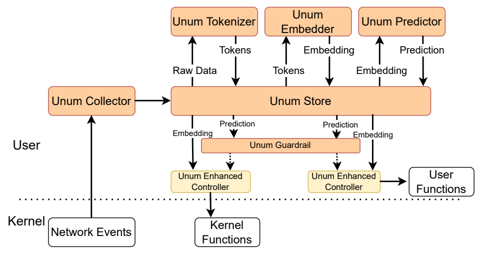


Figure 1: UNUM's components (orange) and controller integration.

of Transformer-based deep neural network models [51], and diverse training datasets to capture latent state relations and provide good predictions of the current state given a temporal history of observations.

UNUM's embedder (Figure 1) consumes *tokens* that encode RTT-timescale network statistics and is pre-trained using *next token prediction* — where the next state token is derived based on applying self-attention to a window of past RTTs' observations. The UNUM embedder's training data is collected from diverse settings that emphasize environment variations and inter-control task interactions. Researchers and developers can add novel datasets over time to improve UNUM's modeling capabilities. Our UNUM prototype is open-sourced at `https://github.com/ldos-project/UNUM`

Network Control with Embeddings: We demonstrate that UNUM's global state embeddings—and the future state estimates derived from them (Figure 1)—can enhance network control performance. To this end, we develop two general techniques to integrate the embeddings into existing learned network control frameworks. We study the techniques' effectiveness for two examples: Orca learned congestion control [3] and Penseive adaptive bitrate streaming [35].

Both integration techniques use embeddings as inputs, but at different points in the control pipeline. The first feeds UNUM embeddings in conjunction with existing input features into the learned controller. This approach requires end-to-end retraining of the controller. While the originally proposed learning algorithm can still be used, the model architecture may need to be modified to accommodate the richer embedding inputs. The second introduces a lightweight neural adaptor that takes as input the embeddings and control decisions from the existing controller and refines the controller output. This reduces retraining overhead but may produce suboptimal control decisions compared to the first approach.

In both cases, UNUM's future state predictions can serve as *guardrails* (Figure 1): If predicted states diverge significantly from observed ground truth, we can fall back to existing controllers that operate without these embeddings. Beyond learned controllers, we also show that incorporating UNUM's predictions can improve the performance of rule-based controllers that rely on network state estimation, such as BBR.

Implementation: We design and implement key building blocks to run UNUM in user-space today. These are available with our prototype and include an eBPF-based collector, a store for network features and embeddings, and an embedding/future state estimate cache. The blocks' asynchronous operation enables network control tasks to opportunistically use available embeddings. We show how to make careful embedder configuration choices that lower control overhead while improving end-to-end control performance.

Evaluation: We thoroughly evaluate the embedding design space, finding that RTT-scale feature aggregation improves control, classification outperforms regression for next-token prediction, and bucketization strategies significantly impact embedding and control quality. Second, we show that UNUM embeddings enhance learned congestion control and adaptive bitrate selection across real-world trace emulations. Integrating UNUM improves Orca's test reward by 13.64% and Pensieve's test reward by 45.48%. When Guardrails are enabled, Orca's improvement further rises to 19.65%. Finally, we show that UNUM introduces a modest decision delay of 2.68 ms, which remains negligible relative to the coarse-grained control intervals of the controllers we integrate with.

2 On State Estimation and Network Control

We use two representative network control tasks – congestion control and adaptive bitrate selection – to show the performance impact of their state estimation's inability to capture latent factors and accurately estimate current state.

2.1 Latent Factors

Congestion Control (CC): CC regulates a network flow's sending rate to match the available bandwidth, maximizing utilization while minimizing delay and packet loss. Traditional CC algorithms (Cubic [23] and BBR [9]) rely on instantaneous measurements of packet loss, delay, and throughput to estimate the network state. These estimates are fed into manually designed heuristics to adjust the congestion window (cwnd). Recent advances in learning-based CC outperform traditional methods [3, 14, 25, 28, 54, 56, 57] by replacing manual heuristics with black-box functions that learn more complex mappings from state to actions. However, state estimation remains rudimentary, relying on runtime features and coarse-grained statistics aggregated over short time windows.

Consider Orca [3], a hybrid congestion control scheme that combines Deep Reinforcement Learning (DRL) with a traditional algorithm (Cubic, in Orca's case). The DRL agent periodically suggests adjustments to the cwnd, while the traditional algorithm handles fine timescale adjustments, which are periodically "modulated" by the DRL recommendations.

Orca's monitoring block captures a mix of instantaneous signals (e.g., current cwnd and RTT) and averaged statistics (e.g., delivery rate, loss rate, delay, sRTT), which it provides as input to its DRL agent. However, this limited feature set overlooks critical latent factors, such as path property variations (e.g., capacity changes on wireless links [21,32]), background traffic patterns sharing the bottleneck, and application demand

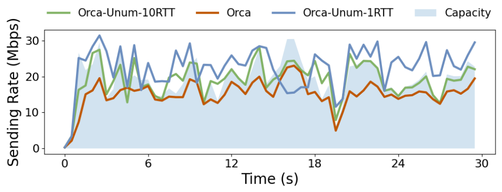


Figure 2: Sending Rate of Orca and Orca augmented with UNUM embeddings using 1RTT and 10RTT context lengths over a real-world cellular trace.

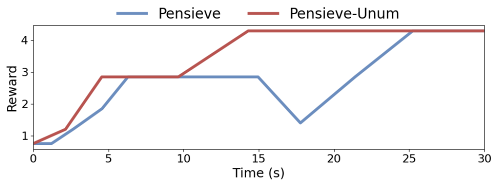


Figure 3: QoE reward (a linear combination of bitrate, rebuffering time and bitrate change) of Pensieve and Pensieve augmented with UNUM embeddings over a real-world cellular trace.

shifts. These latent factors impact optimal control actions and thus should ideally inform CC decisions. Without accounting for such latents, Orca's DRL agent struggles to generalize across diverse network environments, a limitation shared by other learned controllers [14, 28, 52].

In Figure 2, we present the results of an experiment over a real-world cellular trace where using a lacking feature set (as is the case with Orca) leads to consistently conservative recommendations, resulting in bandwidth under-utilization. In contrast, using state embeddings that are derived from primitive features recorded over a window of 10 past RTT's to drive the suggestions made by Orca's monitoring block, we observe network saturation more often. This leads to an *11%* improvement in average utilization. By implicitly modeling these latent factors, the embeddings help better track network capacity and improve control decision quality (§2.2).

Adaptive Bitrate Selection (ABR): To provide seamless viewing experience amid fluctuating and unpredictable network conditions, content providers rely on ABR algorithms. Traditional algorithms estimate delivery rate (throughput) and monitor client-side metrics in real time, and employ manual heuristics to dynamically adjust video quality. State-of-the-art controllers replace manual heuristics with learning-based ones. For example, Pensieve [35] uses A3C [38], a reinforcement learning algorithm that trains a neural network policy to map network observations to informed bitrate decisions.

Like most manual, heuristic-based video streaming policies, Pensieve mainly relies on client video player measurements such as buffer occupancy or average delivery rate sampled at the granularity of video segments - often on the order of several seconds. These segment-level metrics overlook transient network dynamics, such as congestion bursts, and only provide coarse insight into the network state and their impact on delivery rate.

To validate this claim, we conduct an experiment comparing the standard Penseive model against a modified Penseive (Penseive-Unum) that is trained using the same actor-critic training framework as the former, but with the addition of learned embeddings into the input. Both these models are trained to maximize the Quality of Experience (QoE) by way of a surrogate reward: a linear combination of bitrate, rebuffering time and bitrate change. In Figure 3, on a real-world cellular trace, we see that the additional network state emebedding inputs in Penseive Unum improves the QoE metric by 40.3% on average in comparison to standard Penseive.

2.2 Importance of Current State Estimate

Using a simple model of network control, we now argue that a rich understanding of the current state improves control performance. We then present empirical substantiation. Consider network congestion control. A stylized fluid model[1] for "additive increase multiplicative decrease" (AIMD) control that has been widely used for analyzing Internet stability is:

$$\dot{x}(t) = \kappa(w - x(t-d)p(x(t-d)) \tag{1}$$

where $x(\cdot)$ is the transmission rate, $p(\cdot)$ is the congestion indicator at the bottleneck router, and d is the round-trip delay. In words, the source adapts its transmission rate $x(t)$ based on the congestion level $x(\cdot)p(x(\cdot))$ at the router d time units in the past. Using linearized analysis, local stability of the controller can be studied to understand the interplay between network delay d and how aggressively the controller should react to congestion (parameterized by κ). After linearization and reparameterization, we obtain [29]:

$$\dot{y}(t) = -\alpha y(t-d) \tag{2}$$

where $y(t) = x(t) - x^*$ is the rate perturbation centered about the steady-state transmission rate x^* (solution to the fixed point equation: $w = x^* p(x^*)$), and α is a "gain" constant that depends on the controller gain and the "steady-state" congestion level at the router. The key result in [29] is that if the gain-delay product $(\alpha \times d) > \frac{\pi}{2}$, then the controller is unstable, meaning that the injected rate will not converge (to the ideal x^*) and instead will either oscillate or diverge. Suppose instead that we can better predict the *current* congestion level (network state) at the router, e.g., by extracting information from the latent patterns in the time-series of features' history.

In this case, a plausible linearized model for control is:

$$\dot{y}(t) = -\alpha(y(t-d) + by(t)) \tag{3}$$

Here, b captures the relative strength of the current congestion level (network state) estimate ($b = 0$ corresponds to the

[1]The model described here is for a single flow and with a single bottleneck router with delay d. This model can be easily generalized to multiple flows and links, see [29] for details.

standard linearized fluid model). This delay-differential equation has been studied [19], and it has been shown that if $d > \frac{1}{\sqrt{1-b^2}} \cot^{-1}\left(\frac{-b}{\sqrt{1-b^2}}\right)$, the network will be unstable (we have normalized $\alpha = 1$).

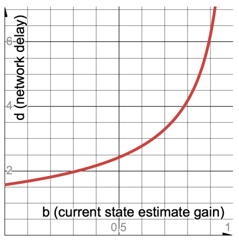


Figure 4: Delay vs Gain

Figure 4 plots the maximum allowable RTT (d) against the state estimate gain b (over the range $[0,1]$). The key observation is that the strength of the estimate of the current network state (value of b) improves the stability tolerance of the controller with respect to path RTTs; for this stylized model, this shows that learning the current state has the benefit of enabling the controller to stabilize and converge-toward the ideal rate x^*. Empirically, our results in Figure 2 present similar trends in a real-world trace: Compared to the 1 RTT embedding, the richer 10 RTT embedding, which incorporates a longer temporal context, enhances performance by capturing long-term network dynamics more effectively. This unlocks stronger predictive power (see Figure 7), enabling the controller to make more informed congestion window adjustments and better adapt to evolving network conditions.

3 Unified Network State Embedding

A well-known principle in control theory is that of “separation” [6]. Informally, it states that an optimal feedback controller for a partially observed stochastic system can be decomposed into two distinct components: a state observer and a deterministic controller. In the classic setting of linear systems with Gaussian noise, for instance, the unique optimal solution is to combine a Kalman filter (for state estimation) with a linear-quadratic regulator (control). This emphasizes the role of precise state estimation in robust and stable control.

Inspired by this principle, we propose to make state estimation a standalone first-class entity. Instead of having each controller rely solely on baked custom, and often ad hoc, state estimation logic intertwined in its design, we introduce UNUM, a unified network state embedding framework. As shown in Figure 1, UNUM deploys an independent embedder module to learn a rich latent representation of the network’s underlying environments that can leverage diverse cross-task training datasets spanning a range of network environments. The embeddings can be used by multiple controllers. UNUM embeddings decouples the complexity of state estimation from control logic by avoiding complex in-controller feature engineering for state estimation, enabling controller developers to focus on high-level policy goals. In §4, we show how existing controllers can leverage the power of this entity together with their own modeling of the mapping from state to actions toward better control decisions.

3.1 Design Space for the UNUM Embedding

To best serve various network control tasks, we need to make unique design choices for the embedding. In particular, we must use design patterns different from traditional practices in both (learned or heuristically designed) network controls and systems for generative AI. We discuss these next.

3.1.1 Windowed Next-Token Prediction

Network state estimation is fundamentally a sequence modeling task, where each state depends on prior events—transmissions, queuing, bursts, and drops—reflected in temporal features like RTTs, delivery rates, and loss. Effective controllers need representations that capture how such sequences evolve and how past events influence future conditions. In NLP, common "pretraining" objectives for generative models include masked language modeling [11, 13, 34] for bidirectional encoding, and next-token prediction [8, 12] for autoregressive tasks. However, neither objective directly suits the generation of network state embeddings, since network statistics evolve causally with respect to historical events and affect future observations spanning *multiple* RTTs.

UNUM *approach.* As discussed in §2.2, an accurate estimate of the network state (or indeed a representation that is predictive of future states [22, 24]) can positively impact decision making. This observation led us to opt for a next-token prediction style objective for embedded training rather than masked-language modeling (that only predicts over the historical window without making forward-looking decisions). Crucially, to better align with the causal nature of network state evolution, UNUM modifies standard next token prediction into "windowed next-token prediction" and computes a loss function over a *window of future predictions* rather than just one single step. We discuss the loss function in §3.1.5.

3.1.2 Transformer-based Model Architecture

Traditional ML models, like MLPs [45], CNNs [31] and LSTMs [27] are commonly used in time-series prediction [30, 33, 46] but each has limitations. MLPs treat inputs as independent features and are agnostic to temporal relationships in the features. They fail to capture how past network states relate to each other and influence the future. While CNNs and LSTMs identify more complex patterns, they struggle with long-term dependencies [5], which may be crucial to understand long-term correlations in network statistics.

UNUM *approach.* UNUM trains an encoder-decoder transformer with causal attention to map token sequences into a latent network state via a windowed next-token prediction task (§3.1.1). Transformers are well-suited for sequential data due to their scalability, ability to handle variable-length inputs, and strong generalization across diverse tasks (akin to what we require of our controllers) when trained on broad datasets. UNUM leverages self-attention to capture input-history dependencies to improve state estimation. The output of the encoder forms the UNUM embedding, encoding the full observation history using causal attention that is trained to be predictive

Table 1: Characteristics of the Data Collection Environments

Parameter	Range / Value
Application	Fixed-rate data transfer; Adaptive video streaming with BBA and RobustMPC
Congestion Control	Reno, Pure CUBIC, Vegas, BBR, CDG, Hybla, HighSpeed, Illinois, Westwood, Yeah, HTCP, BIC, Veno
Bandwidth	6 Mbps – 192 Mbps (constant or step); Real-world traces [17,44,56]
Base RTT	10 ms – 160 ms
Queue Size	$\frac{1}{2}\times$ BDP – $16\times$ BDP

Table 2: State Embedder Input Features

Feature	Description
Base RTT	Minimum RTT observed over the connection path.
sRTT	Smoothed RTT, updated per RTT interval.
Avg Tput	Average throughput measured over each RTT interval.
sRTT Var	Variation in smoothed RTT over one RTT window.
Loss Rate	Ratio of lost packets to total packets sent within one RTT.
cwnd Rate	Rate of bytes change in the congestion window size per RTT.

of future network behavior. We detail the decoder's role and its integration with controllers in §4.2.

3.1.3 Rich Cross-Task Training Data Collection

Issue. It is common for learned controllers to form task-specific datasets to optimize their singular objectives rather than system-wide performance. These are gathered by only varying parameters that are relevant to the particular control task and keeping all others static. For example, adaptive bitrate controllers are often trained on datasets where the congestion control algorithm remains fixed; thus, the model never learns to adapt to the use of different congestion control strategies.

UNUM *approach.* To train a general-purpose state embedding that supports multiple control tasks, we build a cross-task, cross-policy dataset that captures a wide spectrum of real-world network control deployment variability (Table 1). We discuss how we build this dataset in detail in Appendix 8.1.

Together, these form the dataset for all UNUM embedder training and evaluation results presented in the following sections. *Our framework also allows developers to readily expand the training dataset with, e.g., measurements over additional applications or network conditions.*

Issue. Existing controllers have performed complex in-controller feature engineering for state estimation. We initially experimented with the 69 feature set from Sage [57], and observed negligible improvements in state prediction or end-to-end control performance while incurring substantially higher training and inference overhead (we require a larger training dataset and larger transformer models to learn the additional feature correlations, and longer training time due to the increased input dimensionality).

UNUM *approach.* UNUM uses six network layer statistics, listed in Table 2, to serve as input features for the UNUM embedder. This concise set captures key dimensions of observable network statistics — latency, throughput, loss rates, and congestion dynamics — and we've found it to be sufficient for the controls studied in this paper.

3.1.4 RTT-scale Statistic Aggregation

Issue. As illustrated in §2, the majority of state estimates used to instruct control actions today are either instantaneous or a series of statistics collected at fixed, fine time-scales (e.g., every 10ms). The former does not provide enough temporal context, leading to suboptimal decision making, while the latter suffers from large input sizes when scaling to network configurations with large round trip delays.

UNUM *approach.* UNUM processes network statistics into series of tokens, where each token is a 6-dimensional vector of network feature values (Table 2) aggregated over one Round Trip Time (RTT) period, as control actions happen at RTT scales. This addresses both issues by adapting the amount of wall-clock time spanned by the tokens based on the observed RTT of the network flow.

3.1.5 Feature Discretization and Classification

Issue. Existing works use a time-series representation for input features with floating point covariates [3, 35, 54, 57]. However, in practice, network control decisions often hinge on threshold-based triggers (e.g., crossing a queueing delay threshold prompts a rate drop). Fine-grained differences of a few milliseconds or Mbps rarely lead to a significant algorithmic change in the control action, such as going from an increase to a decrease.

UNUM *approach.* In line with this observation, we discretize each input feature over its observed range into semantically meaningful buckets, enabling the model to focus on significant shifts in feature values. This reduces noise and simplifies the learning problem by lowering the number of distinct input patterns. An exception is the base RTT feature, a (normalized) constant per flow.

A challenge with discrete input spaces is selecting effective bucket boundaries for each feature. We use Quantile-based bucketization with 50 buckets as the default discretization method henceforth for all experiments in this paper, unless stated otherwise. We show the detailed bucketization strategy introduction and comparison results in Appendix 8.2.

In addition, the feature discretization naturally transforms the prediction task from a regression problem into a *classification problem*, where the model predicts the discrete class corresponding to each feature. This further simplifies learning and aligns well with network control needs as observed above.

Issue. Employing the classification loss over entirety of the vocabulary in our case is impractical. This is due to the vocabulary size scaling combinatorially in the number of buckets for each feature. For example, with the quantile-based discretization, using 50 buckets per feature yields over 4 million

Table 3: Hyperparameter Space for Transformer Evaluation

Param	Range / Value	Param	Range / Value
Encoder Layers	2–16	Num of Heads	2–8
Decoder Layers	2–16	Dropout	0.1
Embedding Size	16–64	Epochs	1000
FF Dimension	32–256	Batch Size	2048
Learning Rate	5×10^{-5}–1×10^{-4}		

unique tokens. Not only does this present a challenging classification problem due to the sparsity of feedback, but also severely bloats the overall model size (the output dimension of the classification head is the size of the vocabulary).

UNUM *approach.* UNUM thus employs a multi-head classifier where each feature is assigned a dedicated classification head that predicts its corresponding bucket index. The number of classes per head is exactly the number of buckets of the associated feature. The model's overall loss is computed as the sum of the cross-entropy losses across all feature heads. This modification improves scalability: the size of the model now being dependent on the granularity of individual feature discretizations rather than the overall vocabulary size.

3.2 UNUM Embedding Evaluation

We now empirically study the effectiveness and trade-offs of the above UNUM embedding design choices, focusing on the embeddings' ability to accurately predict future network states in CC [3] and ABR [35]. We believe that our design space analysis above–and the empirical study to follow–together provide a principled framework for building and analyzing global embeddings for various system settings beyond the endhost network control problems that this paper focuses on.

We answer several key questions in turn below.

Does RTT-based aggregation yield more effective embeddings than fixed time-interval aggregation?

Setup. We compare the future bucket prediction accuracy of our RTT-aggregated tokens against fixed 10ms interval aggregation tokens (10ms is the aggregation interval used by Orca). Both are trained with the same total sample number, and take in 10 history tokens to predict 10 future tokens. As before—*and henceforth*—we perform a sweep of transformer hyperparameters in Table 3 and report results for the best performing choice.

Results. RTT-based aggregation presents a challenge compared to fixed time-interval aggregation in terms of next token prediction learning over environments with various RTTs because tokens correspond to different absolute durations depending on the RTT. This requires the transformer model to learn and adapt to shifts in RTT scales across different network flows. Despite this challenge, Figure 5 shows that the transformer achieves comparable token prediction accuracy for both RTT-based and time-based aggregation, demonstrating the model's ability to generalize across RTT variations. However, the key distinction lies in how these embeddings impact downstream network control. Figure 12c highlights that RTT-based aggregation leads to higher utilization in downstream network controllers compared to time-based, suggesting that RTT-aligned embeddings provide a more meaningful state representation for control tasks. We select *RTT aggregation as our default choice for experiments henceforth.*

Is classification a more suitable formulation than regression for state prediction tasks?

Setup. Using the same dataset, we train and evaluate the transformer on regression (predicting future token window values) and classification (predicting token classes). For the regression-based transformer, we define the bucket index distance for each feature as the distance between its predicted and actual value buckets.

Results. Figure 6 shows that the transformer can do classification much more accurately than regression. With classification, 90% of tokens are correctly assigned to their true or an adjacent bucket. In contrast, with regression, fewer than 10% of predicted values fall within one bucket of the ground truth.

Does longer context improve the embeddings?

Setup. We vary the input context length (1, 5, 10), i.e. how many history tokens are fed into the transformer, and train different models to predict the future tokens with the same prediction length. We only compare the commonly predicted first future token's bucket prediction accuracy.

Results. Figure 7 shows that increasing the history context leads to higher prediction accuracy. Models trained with longer input histories consistently achieve smaller prediction errors and tighter CDFs. Longer history windows provide the model with more temporal information about how network state evolves, enabling it to better infer trends, such as congestion buildup, which could be especially useful in complex, highly variable bandwidth environments.

How does the transformer architecture perform compared to baseline models?

Setup. We compare the future token prediction accuracy of our transformer (we show both embedding sizes of 16 and 64) against commonly used models for time-series prediction, including: (1) Multi-Layer Perceptron (MLP) (34 hidden layers); (2) Convolutional Neural Network (CNN) (256 channels); (3) Long Short-Term Memory (LSTM) (128 hidden dimensions). To ensure a fair comparison, all models are designed with approximately 10 million parameters and are trained for 1000 epochs. All models use RTT-based tokens with Quantile-50 bucketization.

Results. As shown in Figure 8, transformers achieve significantly higher accuracy than other models, effectively predicting a high percentage of future states across diverse network traces. The 16-embedding transformer correctly predicts 60% of future network state buckets within a 1-bucket range, while the 64-embedding transformer improves this to 70%.

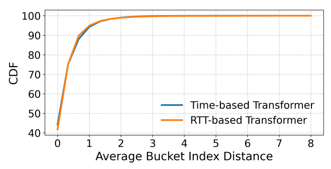


Figure 5: CDF of average bucket prediction error with RTT-based and time-based models.

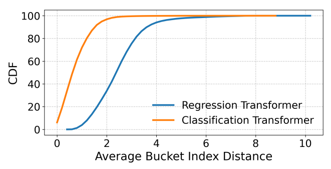


Figure 6: CDF of average bucket prediction error with regression and classification.

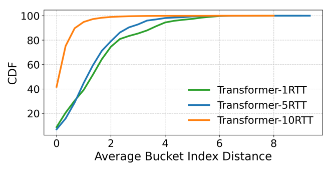


Figure 7: CDF of average bucket prediction error of various context length.

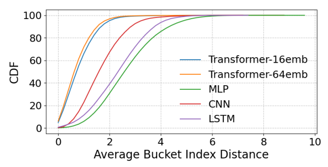
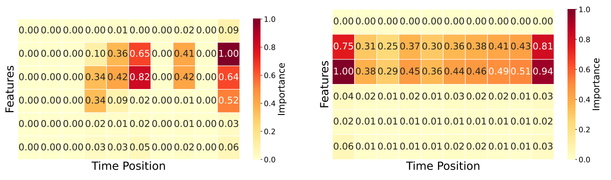


Figure 8: CDF of prediction error comparison for different model architectures.



(a) Test Sequence 1

(b) Test Sequence 2

Figure 9: Feature importance over time (right indicates more recent timesteps). From top to bottom, features are: Base RTT, sRTT, Avg Tput, sRTT Var, Loss Rate, cwnd Rate.

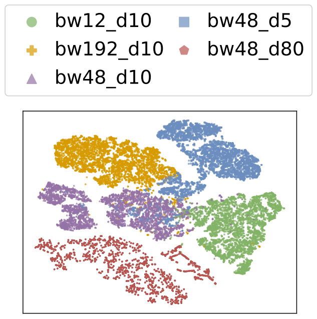


Figure 10: t-SNE visualization of UNUM embeddings for 5 environments.

Despite their comparable total parameter sizes, the encoder parameter sizes differ significantly: the 16-embedding transformer has 16k encoder parameters, while the 64-embedding transformer has 400k encoder parameters. This distinction introduces an interesting tradeoff - the larger-embedding models improve predictive accuracy but introduce higher overheads, which we explore further in §5.4.

What is the contribution of each feature and its time scale?

Setup. We analyze the transformer's feature importance patterns by visualizing attention-weighted feature contributions from test datasets. Specifically, for each test sequence (6 features and a 10-RTT context), we compute the features' importance by weighting them over the encoder-decoder cross-attention weights. Figure 9a and Figure 9b show two representative heatmaps from randomly selected sequences, where the latter corresponds to a video streaming trace.

Results. While in general, sRTT (row 2) and average throughput (row 3) stand out as strong indicators of network state, we observe that many features, and not just the most recent ones, contribute non-trivially in at least one of the samples. Feature importance also varies across sequences, as the network conditions change. For the video-streaming trace (Figure 9b), both near- and long-term context appear to matter reflecting the fact that video delivery spans a long time duration, and long-term trends can directly affect playback smoothness.

Do UNUM embeddings encode meaningful distinctions across network environments?

Setup. We visualize the embeddings from five randomly selected network environments - each with distinct fixed bandwidth(bw) and RTT(d) values - using t-distributed Stochastic Neighbor Embedding (t-SNE) [50]. This method projects high-dimensional embedding data into two dimensions while preserving local neighborhood structure, allowing us to assess similarity in the embedding space.

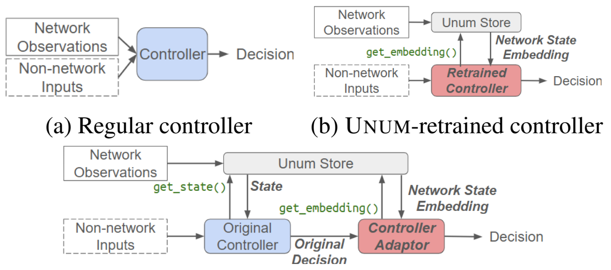
Results. Figure 10 shows that embeddings from the same environment naturally cluster together, while clusters corresponding to different environments remain distanced. Moreover, environments that have closer setups (e.g., share the same RTT or same bandwidth) form partially overlapping clusters (purple overlaps with green and blue). This indicates that UNUM embeddings effectively encode latent environment characteristics into its representation space.

4 UNUM Integration with Network Control

This section focuses on techniques for integrating UNUM with network controls. We start with an operational view of UNUM and its key building blocks that enable integration.

4.1 An Operational View of UNUM

As shown in Figure 1, all components of UNUM operate in user space to ensure portability and ease of deployment. The key components work as follows: At runtime, a background *Collector* monitors network events and records UNUM feature-relevant TCP statistics into the *Store*. The *Tokenizer* processes the collected statistics at the granularity of each RTT, generating a new token that captures the most recent network state. Unlike learned tokenizers in NLP [48], the Tokenizer is a hand-engineered module that performs aggregation and discretization based on the base RTT and predifined bucket boundaries rather than text patterns. The generated token is passed to the *Embedder*, which produces an updated embedding of the current network state. The resulting embedding is written back to the Store for use by controllers. Controllers integrated with UNUM query the Store when necessary to retrieve the most recent embedding, and use this information



(a) Regular controller (b) UNUM-retrained controller

(c) Controller with UNUM adaptor

Figure 11: UNUM controller integration: (a) working of a regular controller, (b) controller integration via retraining with UNUM embeddings, (c) controller integration with UNUM adaptor.

in their decision-making pipelines.

UNUM can optionally deploy a *Decoder* that runs as a background process, fetching the latest embeddings and generating predictions for future token buckets (written to the Store). The *Guardrail*—another optional UNUM component—uses historical Decoder predictions along with the latest tokens from the Store to guide decision-making. It supports two key actions: (a) determining whether predictions are consistently and sufficiently inaccurate compared to the ground truth reflected in the tokens; (b) if so, disabling the UNUM-integrated controller and switching control to a predefined "backup" controller (e.g., the original non-UNUM version). Additionally, (a) can be used to trigger *retraining* of the Embedder. Note that custom policies can be defined for (a) and (b).

The Collector needs to efficiently gather a few main types of data: *(i)* recent network observations (e.g., throughput, RTT, queue occupancy) that must be frequently sampled from within the kernel and tokenized, *(ii)* recently computed embeddings that are stored but are to be used in real-time (soon after they are computed) by controllers, *(iii)* predicted tokens (optional). The data collection must be low-overhead but also occur at a sufficient frequency. To this end, our Collector relies on the widely supported eBPF [1], which is lightweight and avoids kernel changes. Thanks to eBPF, the UNUM collector only takes 25 LoC in our prototype (to add kprobes for Kernel TCP struct tracing). The Collector can be easily extended to measure any kernel structure.

The Store provides a structured interface for updating/accessing the latest network observations collected by the collector and the corresponding embeddings. By decoupling data collection from embedding computation, the Store enables asynchronous operation and helps maintain high performance. Inspired by KernMLOps [2], we use KFSTORE to manage all intermediate UNUM data as eBPF maps, since these data are array-like and can be easily serialized. As a result, all UNUM data benefit from minimal access latency, as they are registered to be accessible from both kernel and user space. Accessing data from KFSTORE requires only a pointer dereference.

The Tokenizer and Embedder operate as outlined in §3.

4.2 Controller Integration

We aim to integrate learned network controllers into UNUM, which have outperformed heuristic approaches due to their proven adaptability. However, existing controllers rely on custom feature pipelines and model tuning. Thus, our goal here is to seamlessly incorporate UNUM embeddings into the controllers to enhance them with minimal effort.

This raises two challenges: *1. Training cost:* UNUM embeddings require packet-level features, which stylized simulations (e.g., Pensieve's, which models video delivery time by abstracting network capacity) often omit. Meanwhile, training with high-fidelity emulation or real prototypes is time-intensive and often impractical. *2. Model adjustments:* UNUM generates compact but high-dimensional embeddings that may require significant changes to existing architectures for integration.

We explain our solutions in the context of controllers for CC (Orca [3]) and ABR (Penseive [35]).

Controller Training. To enable the use of UNUM's state representation, the learned controllers (Figure 11a) require retraining or fine-tuning to fully exploit the additional network state representaion. We introduce two approaches to make this possible without rewriting the controller's entire logic:

Full model retrain (Figure 11b). The simplest approach is to augment the controller's input space with the UNUM embedding and retrain the controller end-to-end. This allows the controller's neural network to fully leverage the embedding by adapting all parameters for seamless integration.

However, retraining from scratch can be computationally expensive and requires access to the original training pipeline and hyperparameter tuning. Nevertheless, we train such a model for the Orca learned congestion controller, which we call Orca-UNUM-Retrained model. We increased the hidden layer size from 256 in original Orca to 512 to accommodate the expanded input dimensionality.

The full-retraining approach is infeasible for Pensieve. First, Penseive's actor network model architecture was carefully designed for its specific input states - it uses 1D convolutional networks (CNNs) for past throughput, past download time and next chunk sizes, while using simple neural networks for other features. Integrating a higher-dimensional embedding into such a tailored model architecture is nontrivial and risks breaking the well-tuned design.

Another problem is the training cost. To add the network embedding with packet-level information in Pensieve, we moved its training from simulation to emulation (details in §5.1). Under emulation, where real videos are transmitted, a single epoch can take more than 5 minutes. Thus, matching the original Pensieve model's total training epochs takes 6.7 months. Given that the additional embedding dimension is larger than the original feature dimension, a well-trained full-retraining model would require an even greater training time.

Full-retraining also results in larger models, which increases inference cost.

Controller action adaptor (Figure 11c): Alternatively, one can freeze the original controller and introduce a lightweight adaptor module that adjusts the existing controller's output with the UNUM embedding. The adaptor is then trained via the same backpropagation feedback mechanism as the original controller training to produce the final control decision.

This simpler approach preserves the original controller's design - only the adaptor's parameters are updated, and is therefore much easier to train. It integrates smoothly with any controller, even if the controller is a black-box model. However, since the base controller is unchanged, its internal weights cannot co-evolve with the adaptor training. If the original latent space is misaligned with UNUM's embedding, the adaptor may only partially bridge the gap.

Ultimately, the choice between full retraining and an adaptor-based approach depends on training and inference resource constraints and performance goals.

Controller Integration. Once UNUM is running in the end host, it exposes a *get_state* and *get_embedding* API through the UNUM Store, allowing any controller to fetch both raw network state and the related embedding on demand. Computing the embedding and managing data adds latency; §5.4 discusses the impact of the delay UNUM introduces.

Despite UNUM's high predictive embedding quality (§3.2), there can be environments where embeddings degrade, such as sudden bandwidth drops. To guard against harmful decisions under these conditions, controllers can optionally enable and use UNUM Guardrails as outlined in §4.1.

5 UNUM End-to-end Evaluation

We evaluate UNUM end-to-end, integrating it with two representative network control tasks: congestion control (CC; Orca) and adaptive bitrate streaming (ABR; Penseive). Despite more recent works in learning-based network control, such as in CC [3, 57] and ABR [35, 49, 55], we use Orca and Pensieve as our primary learned controller baselines due to their reproducible results and widespread research adoption. We believe that our results extend to other controllers.

We explore three questions: (1) How do UNUM-augmented controllers fare relative to baselines? (2) How do various integration strategies and embedder configurations perform? (3) What are the runtime overheads and their controller impact?

5.1 Evaluation Setup

UNUM *embedder configuration.* By default, the UNUM embedder applies RTT-based token aggregation and uses Quantile-50 bucketization for feature discretization. Each embedding is generated based on a history of 10 tokens. The transformer model used has a feedforward dimension of 256, 4 encoder layers, 16-dimensional embeddings, 4 attention heads, and is trained with a learning rate of 1×10^{-4}.

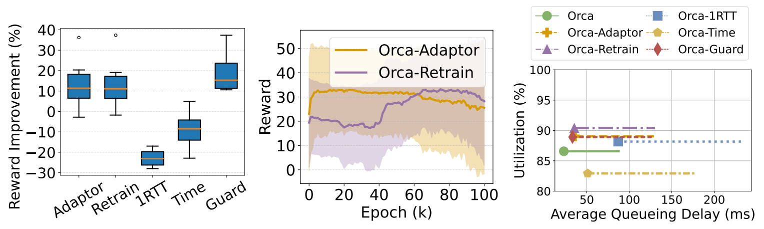
Orca*-UNUM *integration. We replicated the training setup mentioned from [3], and trained both our Orca baseline and Orca-UNUM variants with 320 actors that interact with network environments that are the same as the original Orca, including the bandwidth range of *6 Mbps-192 Mbps*, delay range of *4ms-400 ms* and queue size of *3kB - 96MB*. We train all models with 100k epochs. We train Orca-UNUM variants including various embedder-based full retrained controller and adaptor-based controller. Unless otherwise specified, Orca-UNUM refers to the Orca-UNUM-Retrained, where the full model is retrained using the UNUM embedding with 16-dimension embeddings. We use Orca's original reward function that normalizes throughput by a penalty function based on delay and packet loss, motivating agents to maintain high throughput under controlled latency and reliability. In addition to reporting test reward, we also evaluate end-to-end CC metrics, including bandwidth utilization and average and 95%-ile queueing delay.

Pensieve*-UNUM *integration. Pensieve was originally trained in a pure simulation environment, where chunk download times are computed using a simple bandwidth-based formula. This abstraction omits packet-level network dynamics. To integrate with UNUM and examine how much ABR can benefit from understanding more complex network dynamics, we adapt Pensieve's simulation-based training loop to interact with an emulated network environment, sending real video traffic during training. We base our Pensieve training framework on Genet's reimplementation [54], which changes the model's action space to select using a larger or smaller adjacent bitrate; we use their largest training configuration (RL3), use their off-the-shelf Pensieve model, and train the Pensieve-UNUM adaptors on the same dataset for a direct comparison.

Our Pensieve-UNUM uses an action adaptor that takes Pensieve's bitrate selection and refines them using the UNUM embedding, implemented as a lightweight three-layer fully connected MLP. This approach allows our model to reduce the significant training time required for a full model, as the original Pensieve model has already learned enough correlations between a good bitrate and the chunk-level states. We don't consider Penseive-Retrained for reasons mentioned in §4.2.

We train Pensieve using its original reward function—defined as video quality minus penalties for rebuffering and quality fluctuations—favoring high throughput, low stall times, and smooth playback. We also report the bitrate and rebuffering time.

Hyperparameters. Since UNUM integration changes the input state dimension of the controller models, we retune hyperparameters (hidden layer sizes and learning rates) for each trained controller variant. For full retrained models, we tune all key hyperparameters, while for adaptor-based integrations,

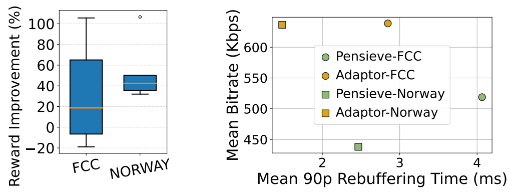


(a) Test reward improvement of Orca-UNUM Variants over Orca.

(b) Training reward curves of Orca-Retrain and Orca-Adaptor.

(c) Orca variants tput util, avg. delay (icons) and 95% tail delay (line ends).

Figure 12: Orca end-to-end results.



(a) Test reward improvement of Pensieve-Adaptor over Pensieve.

(b) Pensieve variants mean bitrate and mean 90p rebuffering time.

Figure 13: Pensieve end-to-end results.

we tune only the adaptor layers, leaving the original controller parameters unchanged. We report the test results for the best-performing configuration based on validation set reward performance.

Test environments We run all the tests on CloudLab Clemson c6420 nodes, each with 64 CPUs and 377G memory. All tests use the real-world emulation setups described in the Orca and Genet papers [3, 54]. For Orca evaluations, we send fixed-rate query traffic over Pantheon traces using Mahimahi with the same delay range and queue size range in the training setup. We repeat each experiment 10 times, and report the average test reward and end-to-end metric values. For Pensieve evaluations, we send pre-recorded videos over FCC broadband measurements and Norway cellular traces. We stick to Pensieve and Genet's emulation test environment setup with $\sim$80ms RTTs and ABR buffer threshold of 60s.

5.2 UNUM Design Space Evaluation

Embedder choices and controller interaction impact.

Remark 1: *Embedding models with richer context and higher quality improve controller performance.* We evaluate controller performance when fully retrained with the following configurations: (1) our default Embedder using RTT-based aggregation with a context length of 10 RTTs (Retrain); (2) an Embedder with a context length of 1 RTT (1RTT); and (3) an Embedder with a context length of 10 tokens, where each token aggregates measurements over 10ms (Time), which is on the same scale as Orca's original input.

Embeddings that exhibit stronger predictive capabilities (see §3.2)—in particular, Retrain outperforming both 1RTT and Time—also lead to better end-to-end performance when integrated into controllers (Figure 12a). Since both the time-based and 1RTT-based Embedders use significantly shorter history windows (especially when the base RTT exceeds 10ms for the time-based version), the controller using our default Embedder (Retrain) achieves superior test rewards, outperforming the other two by *36.49%* and *22.57%*, respectively.

Detailed performance results are shown in Figure 12c: Orca-1RTT suffers from high queueing delays, while Orca-Time exhibits low throughput utilization.

Remark 2: *Fully retrained controller models achieve greater performance gains, while controller action adaptors are easier and faster to train.* We train both the fully retrained controller model (Retrain) and the action adaptor (Adaptor) for Orca using our default Embedder. Orca-Retrain achieves a *13.64%* improvement in test reward over the baseline Orca, while Orca-Adaptor achieves a *13.60%* improvement. However, as shown in Figure 12b, Orca-Adaptor converges in just 2.8K epochs, significantly faster than the 65K epochs required for Orca-Retrain. This makes adaptors well-suited for iterative controller development; their smaller size also enables deployment on edge devices with limited resources.

Controller performance improvement.

Remark: Our controllers integrated with embeddings consistently improve test rewards across both Orca and Pensieve. Orca-Retrain achieves an average test reward improvement of *13.64%* over the baseline Orca on Pantheon real-world trace emulations (Figure 12a). It delivers higher average throughput utilization, albeit with a slightly increased average queueing delay (Figure 12). We show the Orca variants performance on two representative traces in Appendix 8.3.

For Pensieve, we evaluate Pensieve-Adaptor, which incorporates UNUM embeddings without requiring full model retraining. Despite this limitation, Adaptor achieves test reward improvements of 33.04% over Pensieve on FCC traces, and 53.34% on Norway traces (Figure 13a). They improve both the mean bitrate and mean 90p rebuffering time (Figure 13b).

Table 4: Test reward improvement of Pensieve-Raw over Pensieve. Pensieve-UNUM (1.6M params, CW10) has +40.84% improvement over Penseive.

Parameters	**CW 5**	**CW 10**
1.6M	-9941.95%	-9940.92%
6.4M	+62.43%	+60.36%

Deep Dive on Pensieve-UNUM Performance Gains

Pensieve-UNUM augments the decision of original Pensieve using compact embeddings computed from raw network observations. These embeddings are computed at fine timescales and appended to output of the original Pensieve controller network which uses video chunk-level average

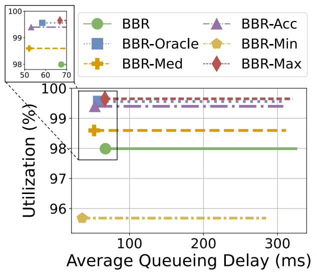


Figure 14: BBR variations: throughput utilization, average queueing delay (icons), and 95% tail delay (line ends).

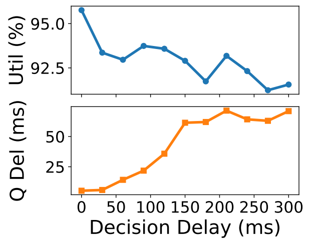
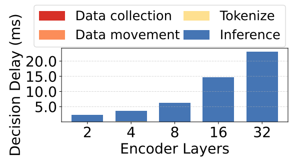
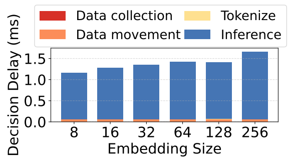


Figure 15: Impact of decision delay on throughput utilization and queueing delay.



(a) Decision delay vs. encoder layers.



(b) Decision delay vs. embedding size.

Figure 16: Sensitivity of Orca's decision delays to encoder depth and embedding size.

statistics as state to adapt the bit rate. To isolate the contribution of the embeddings within the Pensieve adaptor architecture towards bit rate selection, we train an alternative adaptor (Pensieve-Raw) that uses the same adaptor architecture but replaces the embeddings with raw network statistics (i.e., the input features to the UNUM encoder). We explored a wide range of context window lengths and parameter sizes for the alternate model, most of which fail to outperform Pensieve. We highlight a subset of configuration results for this adaptor in Table 4.

Remark: UNUM *embeddings are compute-efficient.* When the number of trainable parameters in Pensieve-Raw adaptor matches those in Pensieve-UNUM's adaptor *plus* UNUM Embedder, Pensieve-Raw consistently underperforms even the original Pensieve.These models display poor generalization to unseen environments and often choose to overshoot bitrate choices, resulting in bloated rebuffering times (up to 20ms). This observation emphasizes the role of the embedding computed using the transformer architecture in the UNUM Embedder towards efficient and robust decision making. However we do observe that Pensieve-Raw improves over Pensieve-UNUM when provided with 4× the number of parameters. This would suggest that UNUM could also benefit from larger adaptor and/or encoder networks at the cost of additional compute overheads.

Guardrails.

Remark: *Guardrails significantly improve the tail of performance gains.* We evaluate the impact of using Guardrails, running Orca-Retrain with Guardrails enabled (Guard). The Guardrail trigger condition as follows: trigger if the average bucket index distance between predicted and later-observed is larger than 5 for more than 10 base RTTs. Guardrails help the controller fallback to the original Orca model whenever UNUM Decoder predictions become inaccurate. We show an example where the guradrail is triggered in Appendix 8.3.

Guard improves test reward by *19.65%* and significantly reduces degradations from the Retrain model (Figure 12a). It cannot completely avoid degradation for integrated controllers because worse controller decisions could be caused by noise rather than the embedding lacking predictive ability. Also, the fallback is only triggered after a defined trigger window (10 RTTs in our experiments), which can neglect short performance degradations and also leave up to 10 RTTs of bad performance before the fallback is activated.

5.3 Case Study: BBR with UNUM.

We evaluate how rule-based controllers can benefit from UNUM embeddings through a case study on BBR [9], a congestion controller that regulates inflight data to match the estimated bandwidth-delay product (BDP).

There are two possible ways to integrate UNUM with BBR: (1) directly replace BBR's bandwidth and delay estimates with UNUM predictions, or (2) learn a downstream adaptor that modifies BBR's output decisions. We adopt the first approach because it preserves BBR's original rule-based control logic, whereas a downstream adaptor could alter the controller's decision structure and reduce interpretability.

BBR sets its sending rate based on estimated bandwidth and RTT. If these estimates were perfect, BBR would naturally converge to the correct BDP. Therefore, improving the quality of these input estimates via UNUM predictions enhances the controller while preserving its original control logic and intuition.

We compare six BBR variants: (1) **BBR**, the original design with in-protocol bandwidth and RTT estimations; (2) **BBR-UNUM**, where bandwidth estimates are replaced with UNUM predictions, with three variants using the *min*, *median*, or *max* value of the predicted bucket (BBR-Min, BBR-Med, BBR-Max); (3) **BBR-Acc**, which assumes error-free UNUM bucket prediction; and (4) **BBR-Oracle**, with perfect bandwidth and RTT knowledge from the trace.

Remark: *BBR can benefit from* UNUM *predictions.*

On Pantheon traces, BBR-UNUM improves utilization to 98.6% and reduces average queueing delay from 67.5ms to 52.2ms (a 22.7% reduction) compared to standard BBR (Figure 14). The remaining gap to oracle performance arises from three factors: (i) UNUM outputs bucket indices, introducing quantization error when converting to values. Using the upper bucket boundary (BBR-Max) overestimates bandwidth, boosting throughput but increasing delay, while the lower boundary (BBR-Min) underestimates and underutilizes the link; the median (BBR-Median) strikes a balance. (ii) Predictions are imperfect, as indicated by the higher performance of BBR-Acc. (iii) Predictions incur delay, analyzed in Section 5.4.

5.4 Decision Delay Analysis

As we showed in §5.2, high-quality embeddings improve controller performance. However, they also introduce higher decision latency - particularly when the embedder runs on CPU. In this section, we quantify the decision delay and show its impact on controller end-to-end performance. All models in the experiments in this Section run on CPUs.

We benchmark the controller critical path decision delay introduced by UNUM, breaking it down into four components: (1) **Data collection**: Time measured between when a network event occurs and when it is logged by the eBPF-based data collector; (2) **Data movement**: Time to transfer raw event logs, tokens and embeddings to/from the UNUM datastore; (3) **Tokenize**: Time to perform RTT-based aggregation and feature discretization on collected events; (4) **Inference**: Time for the embedder model to process the latest token history and generate an updated embedding.

Remark: UNUM *decision delay is domaninated by Embedder inference latency. However, our default Embedder configuration brings negligible performance drop.*

Decision delay breakdown. We evaluate the impact of key embedder model hyperparameters on decision delay. Among them, only the number of encoder layers and embedding size have an observable influence. Figure 16a and Figure 16b show that the dominant contributor to decision delay is the inference time. Inference delay ranges from 2.2 ms to 23 ms as the number of encoder layers increases from 2 to 32, and from 2.2 ms to 3.2 ms as the embedding size increases from 8 to 256. For the default UNUM embedder configuration (4 encoder layers, 16 embedding size), the average decision delay is 2.68 ms (data collection: 38.95 ns, data movement: 0.12 ms, tokenization: 0.01 ms, inference: 2.44 ms). Since both Orca and Pensieve operate at coarse-grained timescales for control decisions, such a small added delay is acceptable.

Impact on Controller Performance. We evaluate the impact of UNUM's decision delay on Orca by injecting artificial delays into its test pipeline in a real-world intracontinental deployment (client on an Azure node in Washington, server on a Cloudlab machine in Utah). These delays model latencies from larger Embedders, extra feature processing, or limited CPU. Figure 15 shows that as decision delay grows, Orca suffers lower utilization and higher queuing delays. Under our default settings, however, delays are small, preserving 96% of the performance relative to the no-delay ideal.

Decoder prediction delay. We separately evaluate the decoder's prediction delay, which impacts guardrail decisions. With our default configuration of 4 decoder layers, it takes an average of 4.91ms to get the future prediction from embedding, which brings minimal performance impact.

6 Related work

A rich body of work has applied ML to systems problems, yielding gains in resource management [43], performance modeling [16, 37], anomaly detection [15, 47], and verification/test generation [10, 41]. More recently, researchers have begun adapting large language models (LLMs) [26, 53] to system tasks, leveraging their reasoning and pretraining scale. While these approaches improve decision quality by mapping observed states to control actions, they remain constrained by naive state representations, as argued in §2.

Recent work has applied transformer architectures to represent system state. Osprey [39] trains a transformer on DBMS logs for query latency prediction, but its embedding is task-specific, co-trained with the prediction head and a query-plan model, and thus unsuitable for multiple downstream tasks. Zoom2net [20] uses transformers to impute fine-grained network data from coarse samples, achieving impressive high accuracy but targeting offline analysis, making it impractical for real-time control. In contrast, UNUM provides a unified, general-purpose network state embedding tailored for real-time control. It produces predictive embeddings at RTT granularity, supporting diverse tasks such as congestion control (CC) and adaptive bitrate (ABR).

Recent work has proposed guardrails and monitoring frameworks for ML-based networked systems (e.g., [36, 42]). Our approach is complementary, advocating prediction–ground truth comparisons to directly drive corrective actions.

7 Conclusion

Our work introduces UNUM, a framework for unified network state embeddings. Drawing inspiration from NLP, UNUM leverages diverse datasets and a transformer-based pretraining approach to build rich state representations from historical features. These embeddings enable network controllers to perform robustly across a wide range of environments and application mixes. UNUM is designed for seamless integration, allowing controllers to easily incorporate its embeddings. Our experiments on both real-world and synthetic traces demonstrate not only how to configure effective embedders but also the significant performance gains they bring to control tasks.

Acknowledgement

We would like to thank our shepherd, Mohammad Alizadeh, and the anonymous NSDI reviewers for their invaluable feedback. This work is part of the Learning Directed Operating System (LDOS) NSF expedition. This work was supported in part by the National Science Foundation under Grants CCF-2326576, CNS-2107037, CNS-2112471, ONR Grant N000142412779, Cisco and the Machine Learning Lab (MLL) at UT Austin.

References

[1] ebpf - https://ebpf.io/, 2024. Accessed: March 18, 2025.

[2] Kernmlops - https://github.com/ldos-project/kernmlops, 2025. Accessed: April 25, 2025.

[3] Siavash Abbasloo, Chen-Yu Luo, and Danny HK Tsang. Classic meets modern: A pragmatic learning-based congestion control for the internet. *IEEE/ACM Transactions on Networking*, 28(5):2003–2016, 2020.

[4] Soheil Abbasloo. Internet congestion control benchmarking. *arXiv preprint arXiv:2307.10054*, 2023.

[5] Safwan Mahmood Al-Selwi, Mohd Fadzil Hassan, Said Jadid Abdulkadir, Amgad Muneer, et al. Lstm inefficiency in long-term dependencies regression problems. *Journal of Advanced Research in Applied Sciences and Engineering Technology*, 30(3):16–31, 2023.

[6] Karl J Astrom. *Introduction to Stochastic Control Theory*. Elsevier, 1971.

[7] Balakrishna Balasingam, Yi Wang, Chi-yuan Lee, et al. Application-aware optimization of 5g network slices using reinforcement learning. In *Proceedings of the IEEE International Conference on Communications (ICC)*, 2024.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[10] Xinyuan Chen and Laurent Vanbever. Ml-driven network verification. *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021.

[11] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019.

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

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[14] Yanhua Dong, Aditya Narayan, Sachin Katti, and Scott Shenker. Pcc vivace: Online-learning congestion control. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 343–356, 2018.

[15] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[16] Jason Ernst, Martin Goldsteins, and Aniruddha Gokhale. Neural networks for modeling system performance. *Proceedings of the 15th International Conference on Artificial Neural Networks*, 2006.

[17] Federal Communications Commission. Measuring broadband america, 2024. Accessed: 2025-04-20.

[18] David Freedman and Persi Diaconis. On the histogram as a density estimator: L 2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 57(4):453–476, 1981.

[19] H.I. Freedman and Yang Kuang. Stability switches in linear scalar neutral delay equations. *Funkcialaj Ekvacioj. Serio Internacia*, 34, 1991.

[20] Fengchen Gong, Divya Raghunathan, Aarti Gupta, and Maria Apostolaki. Zoom2net: Constrained network telemetry imputation. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 764–777, 2024.

[21] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[22] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[23] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[24] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[25] Bo He, Jingyu Wang, Qi Qi, Haifeng Sun, Jianxin Liao, Chunning Du, Xiang Yang, and Zhu Han. Deepcc: Multi-agent deep reinforcement learning congestion control for multi-path tcp based on self-attention. *IEEE Transactions on Network and Service Management*, 18(4):4770–4788, 2021.

[26] Zhiyuan He, Aashish Gottipati, Lili Qiu, Xufang Luo, Kenuo Xu, Yuqing Yang, and Francis Y Yan. Designing network algorithms via large language models. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, pages 205–212, 2024.

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

[28] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

[29] R. Johari and D.K.H. Tan. End-to-end congestion control for the internet: delays and stability. *IEEE/ACM Transactions on Networking*, 9(6):818–832, 2001.

[30] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks, 2018.

[31] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[32] Jinyang Li, Charles Blake, Douglas SJ De Couto, Hu Imm Lee, and Robert Morris. Capacity of ad hoc wireless networks. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 61–69, 2001.

[33] Bryan Lim, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting, 2020.

[34] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[35] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the ACM SIGCOMM Conference*, pages 197–210. ACM, 2017.

[36] Hongzi Mao, Malte Schwarzkopf, Hao He, and Mohammad Alizadeh. Towards safe online reinforcement learning in computer systems. In *NeurIPS Machine Learning for Systems Workshop*. Curran Associates New York, NY, 2019.

[37] Lingpu Meng, Xuechao Wang, Zihao Yu, and Jiahai Yang. Deep learning for system performance modeling: A survey. *ACM Computing Surveys*, 53(4), 2020.

[38] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PmLR, 2016.

[39] Parimarjan Negi, Ziniu Wu, Arash Nasr-Esfahany, Harsha Sharma, Mohammad Alizadeh, Tim Kraska, and Sam Madden. Os pre-trained transformer: Predicting query latencies across changing system contexts, 2024.

[40] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: accurate {Record-and-Replay} for {HTTP}. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, 2015.

[41] Arun Kumar Padhi, Rahul Sharma, and Todd D. Millstein. Automatically learning abstractions for program verification. *ACM SIGPLAN Notices*, 51(6), 2016.

[42] Sagar Patel, Dongsu Han, Nina Narodystka, and Sangeetha Abdu Jyothi. Toward trustworthy learning-enabled systems with concept-based explanations. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, HotNets '24, page 60–67, New York, NY, USA, 2024. Association for Computing Machinery.

[43] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the 3rd ACM Symposium on Cloud Computing (SoCC)*, 2011.

[44] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3g networks: Analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 114–118, 2013.

[45] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[46] David Salinas, Valentin Flunkert, and Jan Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks, 2019.

[47] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, and Todd Phillips. Machine learning for computer system security. *arXiv preprint arXiv:1811.00241*, 2018.

[48] Yusuxke Shibata, Takuya Kida, Shuichi Fukamachi, Masayuki Takeda, Ayumi Shinohara, Takeshi Shinohara, and Setsuo Arikawa. Byte pair encoding: A text compression scheme that accelerates pattern matching. 1999.

[49] Farzad Tashtarian, Abdelhak Bentaleb, Hadi Amirpour, Sergey Gorinsky, Junchen Jiang, Hermann Hellwagner, and Christian Timmerer. {ARTEMIS}: Adaptive bitrate ladder optimization for live video streaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 591–611, 2024.

[50] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[52] Wenting Wei, Huaxi Gu, and Baochun Li. Congestion control: A renaissance with machine learning. *IEEE network*, 35(4):262–269, 2021.

[53] Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. Netllm: Adapting large language models for networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 661–678, 2024.

[54] Zhengxu Xia, Yajie Zhou, Francis Y Yan, and Junchen Jiang. Genet: automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 397–413, 2022.

[55] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 495–511, 2020.

[56] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, 2018.

[57] Chen-Yu Yen, Soheil Abbasloo, and H Jonathan Chao. Computers can learn from the heuristic designs and master internet congestion control. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 255–274, 2023.

8 Appendix

8.1 Cross-Task Training Data Collection Details

We collect data using a "cross product" of: (1) two application types: fixed-rate data transfers and adaptive video streaming (with various ABR algorithms); (2) all Linux CC algorithms (options for both the above summarized in Table 1). We run experiments across a wide range of network conditions, emulating a single bottleneck link between client and server, and varying the bandwidth, base RTT, and queue size using Mahimahi [40]. By sweeping through this space of parameters and configurations, we ensure the embedding is trained on rich interactions between applications, protocols, and environments likely to occur in real systems. We also include real-world, publicly available datasets (details below).

Our data collection yields over 35M unique samples, which we split 80%–20% into training and test sets. We organize the test set into four categories based on the bandwidth trace source and workload type:

- *a) CC synthetic* with constant size queries over synthetic bandwidth patterns such as constant or step changes [4], using various CC algorithms.
- *b) CC real-world* with queries over real-world Cellular and Ethernet bandwidth measurements from Pantheon [56].
- *c) ABR synthetic* by utilizing Pensieve's synthetic training trace generator [35].
- *d) ABR real-world* with video streaming traces from real-world sources such as FCC [17] and Norway [44].

8.2 Bucketization Choice Evaluation

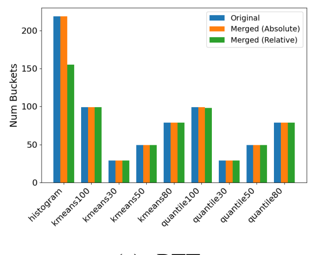
We explore three data-driven bucketization methods:

1. *quantile-based:* this partitions features based on constantly-spaced percentiles;
2. *histogram-based:* this uses the Freedman-Diaconis rule [18] to set bin widths based on inter-quartile range;
3. *clustering-based:* this applies K-Means and uses midpoints between cluster centers as boundaries.

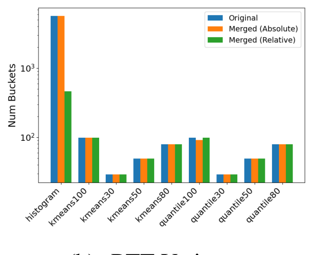
For quantile and clustering methods, we evaluate different bin counts and clustering algorithms. In all cases, coarser buckets reduce vocabulary size, while finer buckets may capture more detail.

8.2.1 Bucket Number Reduction with 1% Precision

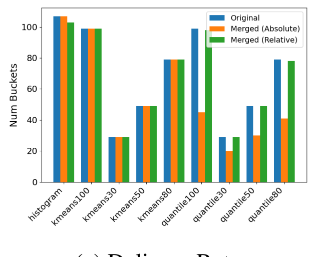
Setup. The bucketization methods can create a lot of buckets, i.e. a large vocabulary size. With each method samples 10% of the training data to define bucket boundaries, we consider merging adjacent ones if their relative difference is under 1%. We compare this merging heuristic using absolute vs. relative difference.



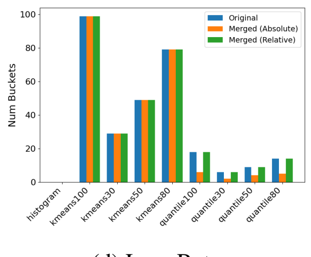
(a) sRTT



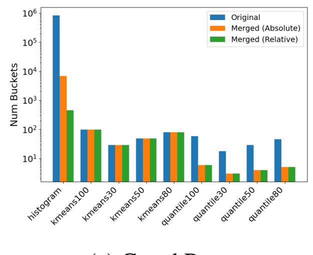
(b) sRTT Variance



(c) Delivery Rate

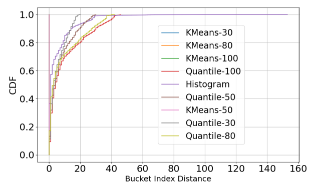


(d) Loss Rate

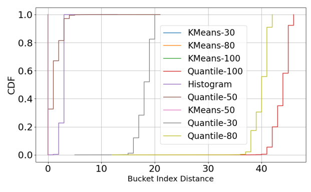


(e) Cwnd Rate

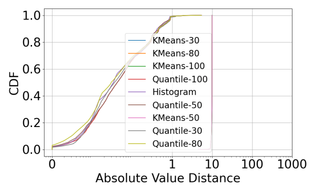
Figure 17: Precision-based bucket merging's impact on bucket numbers for different bucketization methods.



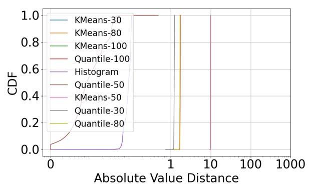
(a) CC Real-world (Index).



(b) ABR Real-world (Index).



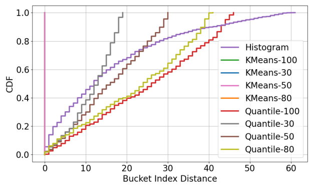
(c) CC Real-world (Value).



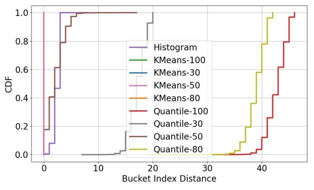
(d) ABR Real-world (Value).

Figure 18: Comparison of CDF distances for delivery rate prediction on Real-world trace. Top: bucket index distance. Bottom: value distance.

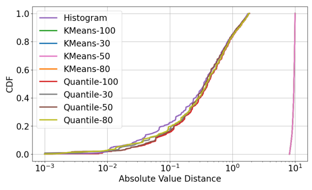
Results. These are shown in Figure 17. We choose the relative difference as the merging heuristic because it effectively reduces the huge bucket number of histogram, while preserving fine-grained separation for small values.



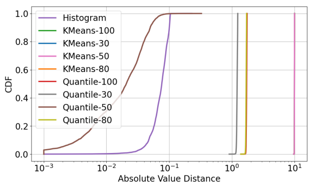
(a) CC Synthetic (Index).



(b) ABR Synthetic (Index).



(c) CC Synthetic (Value).



(d) ABR Synthetic (Value).

Figure 19: Comparison of CDF distances for delivery rate prediction on sythetic trace. Top row: bucket index distance. Bottom row: value distance.

8.2.2 Bucketization Impact on Embedder Predition Accuracy

Setup. We use the data in §3.1.3 to evaluate the three bucketization strategies from §3.1.5—Quantile, Histogram, and KMeans—using multiple bucket counts for Quantile and KMeans.

For each generated boundary set, we discretize the training and testing set by giving each feature in each token a bucket index.

For each potential bucketization, we evaluate transformer models with a range of hyperparameters (details provided in Table 3) and report results for the best performing hyperparameter choice based on next-token prediction accuracy.

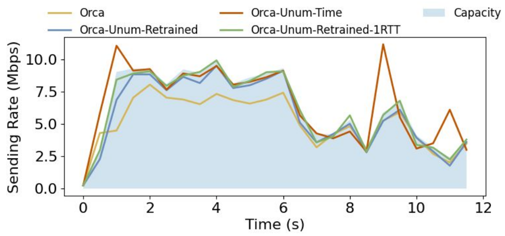
Results. Figure 18 (a)-(b) reports the absolute difference between the predicted and true bucket indices. Finer-grained bucketization (e.g. quantile with 80-100) often leads to high bucket index error predictions.

However, small bucket index distances do not necessarily imply accurate predictions. If the buckets themselves are too coarse, then even a close index may correspond to a large deviation in actual value.

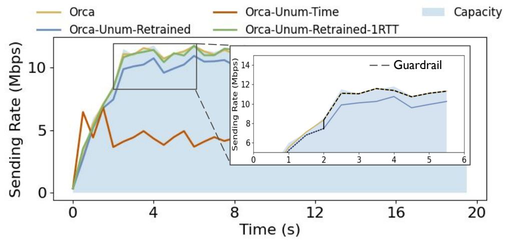
To address this, Figures 18 (c)-(d) measure the value distance—the absolute difference between the true value and the midpoint of the predicted bucket. Interestingly, KMeans with 50 buckets has perfect bucket index distance predictions, but very high value distances. This arises because KMeans-50 forms clusters influenced by extreme outliers, pushing most reasonable values into a single dense cluster. The model can easily predict this majority cluster, appearing accurate in terms of index distance, but the predicted values are far from meaningful due to the poor representativeness of the clusters.

In Figure 19, we show the same for synthetic traces; our observations remain unchanged quantitatively.

Adding to the above, Figure 21 shows the results for



(a) Zoom of a good-performing environment.



(b) Zoom of a bad-performing environment.

Figure 20: Case study of Orca-Retrain results.

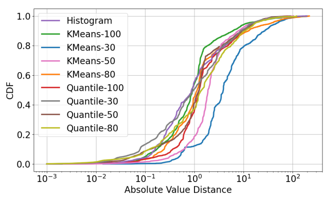
smoothed RTT (sRTT), Figure 22 for sRTT variance, Figrue 23 for loss rate, and Figure 24 for the congestion window; our observations remain unchanged quantitatively.

Based on these results, we use *Quantile-based bucketization with 50 buckets as the default discretization method henceforth* for all experiments in this paper, unless stated otherwise. We'll just look at the average bucket index distance across all features for the whole test dataset in the rest of this section as the bucketization method is fixed.

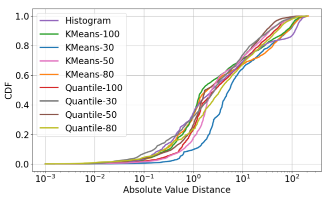
8.3 Orca Case Study

To better understand the conditions under which embed- dings are beneficial, we examine two representative traces. As shown in Figure 20a, when the predictive embedding accurately estimates the available bandwidth, Orca-UNUM- Retrained selects sending rates that closely match the link's capacity, resulting in higher rewards. In contrast, when the embedding fails to anticipate a rise in bandwidth—as seen in Figure 20b—Orca-UNUM-Retrained underutilizes the available capacity, leading to a lower reward.

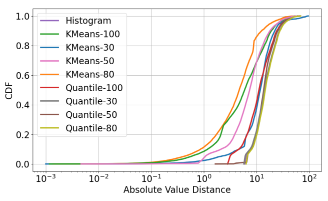
For example, in Figure 20b, the Guardrail mechanism detects sustained poor prediction accuracy between 1.8s to 2.0s (corresponding to the predefined Guardrail trigger window threshold of 10 consecutive RTTs). Once triggered, the Guardrail disables the Embedder-integrated controller and reverts to the original Orca controller, allowing it to quickly recover and match the appropriate sending rate.



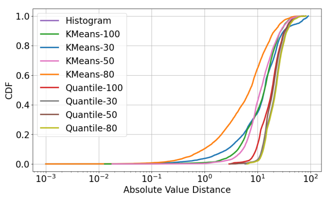
(a) CC Synthetic.



(b) CC Real-world.

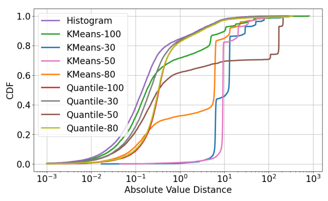


(c) ABR Synthetic.

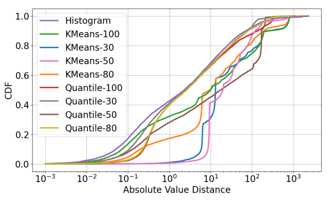


(d) ABR Real-world.

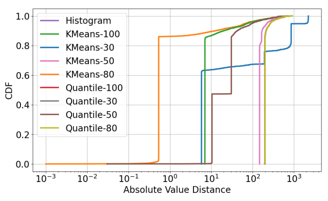
Figure 21: CDF of the average distance between the predicted bucket midpoint and the true **Smoothed RTT (sRTT)** across different test environments.



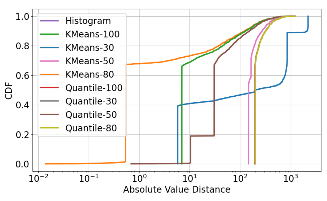
(a) CC Synthetic.



(b) CC Real-world.

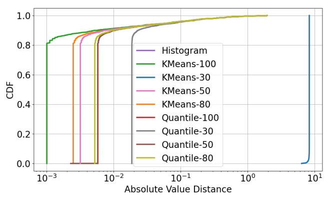


(c) ABR Synthetic.

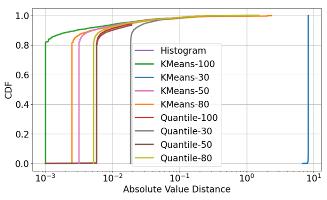


(d) ABR Real-world.

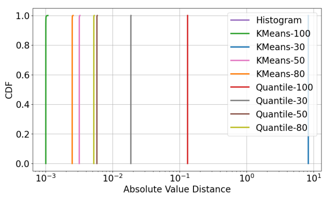
Figure 22: CDF of the average distance between the predicted bucket midpoint and the true **sRTT Variance** across different test environments.



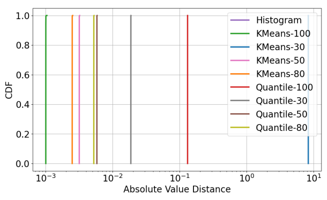
(a) CC Synthetic.



(b) CC Real-world.

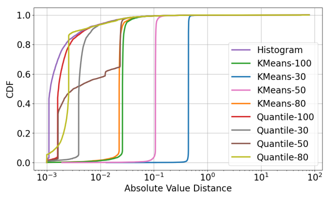


(c) ABR Synthetic.

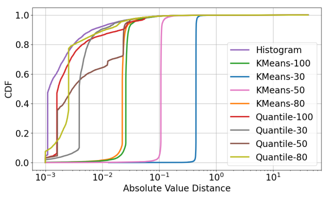


(d) ABR Real-world.

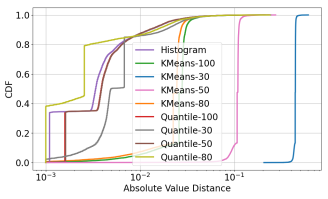
Figure 23: CDF of the average distance between the predicted bucket midpoint and the true **Loss Rate** across different test environments.



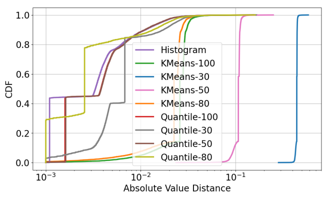
(a) CC Synthetic.



(b) CC Real-world.



(c) ABR Synthetic.



(d) ABR Real-world.

Figure 24: CDF of the average distance between the predicted bucket midpoint and the true **CWND Rate** across different test environments.