IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 December 2025, accepted 24 January 2026, date of publication 28 January 2026, date of current version 4 February 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3658877

RESEARCH ARTICLE

An In-Context Foundation Model for Predictive Process Monitoring on Event Logs

ALESSANDRO BERTI[1] **AND WIL M. P. VAN DER AALST**[1,2]**, (Fellow, IEEE)**
[1]Process and Data Science (PADS), RWTH Aachen University, 52074 Aachen, Germany
[2]Celonis SE, 80331 Munich, Germany

Corresponding author: Alessandro Berti (a.berti@pads.rwth-aachen.de)

This work was supported in part by the JUPITER AI Factory (JAIF) Project, funded via the EuroHPC Joint Undertaking (JU) under Grant ID 101250682 (https://cordis.europa.eu/project/id/101250682); and in part by the German Federal Government and the German States of North Rhine-Westphalia and Hesse, coordinated by Forschungszentrum Jülich and involving consortium partners, including RWTH Aachen University, Fraunhofer, and the Hessian AI Ecosystem.

ABSTRACT Event logs record the execution of business processes as sequences of timestamped events. Most predictive process monitoring methods still learn a separate model per log: when the process, the activity vocabulary, or the time scale changes, the model must be retrained and revalidated. This paper takes a different route and argues for a foundation model for process mining: a single reusable model trained across heterogeneous event logs, designed to generalize to new logs and to be adapted with only a small set of in-log examples. To the best of our knowledge, we present the first event-log-native foundation model specifically tailored to process mining. The model consumes prefixes of cases directly (as ordered event sequences with timestamps and attributes), produces a compact representation of the running case, and supports two core monitoring tasks: next-activity prediction and remaining-time estimation, without any per-log parameter updates. At use time, it adapts in context from a support set sampled from the target log, which aligns the model to the local activity set and temporal scale; string attributes can optionally provide additional semantic hints. We study both balanced few-shot contexts and retrieval-based contexts built by nearest-neighbor selection with same-case masking, and we report extensive ablations over context size and design choices. Results on held-out logs and a real-life case study show that a single pretrained model can transfer to unseen event logs and improve with a handful of contextual examples. Overall, this work is a first step toward general-purpose, reusable foundation models for process mining that can be trained once and applied broadly across organizations and processes.

INDEX TERMS Foundation models, process mining, predictive process monitoring, in-context learning.

I. INTRODUCTION

Process mining studies event data recorded by information systems to discover, monitor, and improve real processes [1]. An event log captures process executions as dcases (e.g., orders, tickets, applications), each represented by a time-ordered sequence of events with activity labels, timestamps, and optional attributes. This sequential, time-stamped structure is central: it encodes control-flow (what happened and in which order) and performance (how long things took). Figure 1 sketches this setting: a single event-log-native foundation model is pretrained across heterogeneous logs and, at deployment on an unseen log, adapts purely in context from a small set of labeled support prefixes to predict the next activity and the remaining time.

A key application area is predictive process monitoring: using partial case histories to anticipate what will happen next in a running case. Typical tasks include predicting the next activity and estimating the remaining cycle time [2], [3]. Most approaches address these tasks by training a supervised model on one historical log and then deploying it on the same process setting [2], [4]. In practice, however, organizations face many event logs and frequent change: activity vocabularies differ across systems, naming conventions vary,

The associate editor coordinating the review of this manuscript and approving it for publication was Wanqing Zhao.

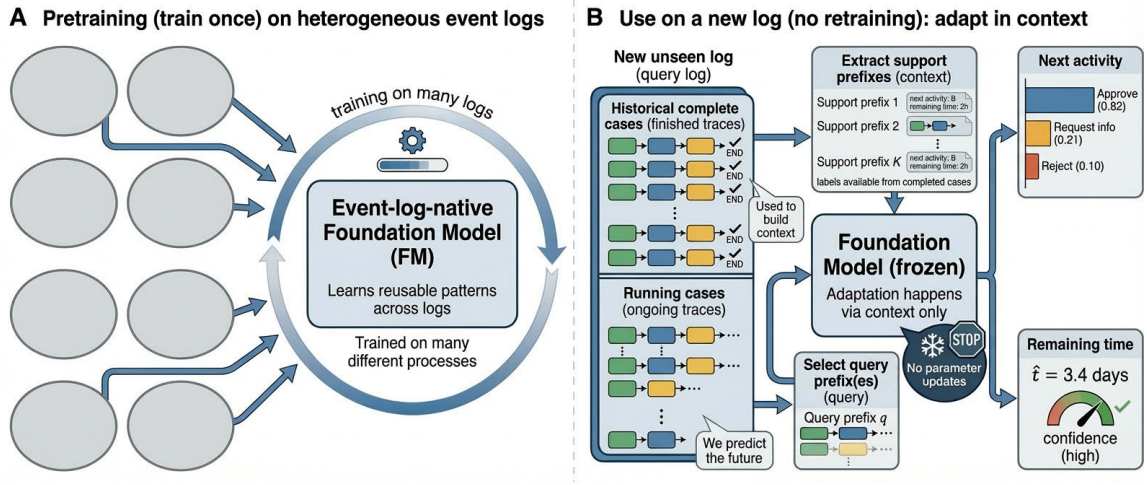


FIGURE 1. Event-log-native foundation model with in-context adaptation to unseen logs. The foundation model (FM) is pretrained once on many heterogeneous event logs. At deployment on a new unseen log, we do not retrain the FM: complete historical cases from the target log are used to extract labeled *support prefixes* (context), while prefixes from running cases serve as *query prefixes*. Given the support and query prefixes, the frozen FM predicts the next activity and the remaining time, adapting to the new log through the provided context rather than through parameter updates.

and temporal regimes shift. As a result, per-log models often require retraining, retuning, and renewed validation whenever the context changes [2], [3], [4].

This paper proposes a foundation-model perspective for predictive process monitoring. In this work, foundation model does not mean a specific kind of neural architecture or simply a ''deep'' model. Rather, it refers to a single reusable model trained across many datasets in the same domain, with the explicit goal of (i) capturing reusable regularities and (ii) transferring to unseen datasets with little or no additional training [5]. In other domains, foundation models have enabled strong transfer across datasets (e.g., language models; and more recently models for time series and tables) [5], [6], [7], [8], [9]. Yet process mining has a distinct input structure: event logs are sequences of timestamped events, not plain tables, and they require models that respect ordering, timing, and local activity vocabularies.

To the best of our knowledge, we introduce the first process-mining-specific, event-log-native foundation model. Event-log-native means that the model consumes event sequences directly (case prefixes as ordered events with timestamps and attributes), instead of relying on log-specific feature engineering or flattening prefixes into fixed encodings that implicitly assume a stable global activity set. The central claim is that a single pretrained model can generalize across event logs: when given a small set of labeled examples from a new target log, it can adapt to that log's local activity space and temporal scale without retraining or updating parameters.

Concretely, the proposed model learns a general way to represent (i) individual events using simple time-derived signals and optional string-based cues, and (ii) whole prefixes by summarizing the event sequence into a fixed-dimensional ''state'' vector. Predictions for a target prefix are then made in context by comparing it to a small support set drawn from the same target log. This makes the adaptation mechanism explicit and lightweight: the support set supplies the local meaning of activity labels and anchors time-related predictions to the target log. We study two practical ways to build such support sets: controlled, balanced few-shot contexts and retrieval-based contexts built from nearest neighbors (while masking examples from the same case to avoid information leakage).

Overall, this work should be read as a first step toward broader foundation models for process mining: models that are trained once on diverse event logs and can be reused across new processes, organizations, and tasks, reducing the need for repeated per-log training cycles and enabling more plug-and-play process analytics.

To make the scope precise, we investigate the following research questions:

RQ1. Cross-log generalization. To what extent can a single event-log-native foundation model, trained across heterogeneous logs, transfer to unseen logs using only a small in-log context and no parameter updates?

RQ2. How to form the context. How do balanced few-shot support sets compare to retrieval-based support sets in terms of accuracy, data efficiency, and robustness to label imbalance?

RQ3. Design choices. Which components most affect transfer (event representation, prefix summarization, context size, and optional use of string semantics)?

RQ4. Reliability. Do the model's built-in confidence signals correlate with errors and help diagnose when predictions are well supported by the available context?

RQ5. Baselines under the same data budget. How does in-context transfer compare to strong baselines that fit a small model directly on the same support examples from the target log?

The rest of the paper is organized as follows: Section II reviews related work on predictive process monitoring and foundation models; Section III introduces notation for event logs, prefixes, and predictive tasks; Section IV describes the proposed event-log-native foundation model and its in-context adaptation mechanism; Section V summarizes the implementation; Section VI reports experimental results and ablations; Section VII discusses alternative design options; and Section VIII concludes and outlines future directions.

II. RELATED WORK

Research on predictive process monitoring and on foundation models spans several directions. We group related work into three areas and compare them with our approach.

A. PREDICTIVE PROCESS MONITORING

Early work focused on supervised learning models trained per event log to predict the next activity, the remaining time, or other process outcomes [2]. Deep learning approaches then gained traction. Recurrent neural networks and LSTMs were used to predict next activities, timestamps, or trace suffixes [4], [10], [11]. More recent work investigated transformer-based architectures for next-activity prediction [12]. Surveys provide broad overviews of machine learning in business process management [13].

These methods are trained for a specific process or dataset. When applied to a new log, they require retraining and assume a fixed activity vocabulary and stable data distribution [2]. They also do not provide mechanisms for in-context adaptation.

Our approach differs by training a single foundation model across heterogeneous logs and adapting it to a new log through in-context examples at inference time, without parameter updates. It operates directly on event sequences rather than static prefix encodings.

B. FOUNDATION MODELS FOR TABULAR AND TIME-SERIES DATA

Foundation models have been developed for time-series forecasting and tabular prediction. Time-series models such as TimesFM and TimeGPT-1 demonstrate strong zero-shot or few-shot forecasting performance across diverse datasets [6], [14]. For tabular data, TabPFN treats classification as amortized Bayesian inference and achieves strong accuracy with minimal configuration [7]. SAP-RPT-1-OSS and ConTextTab extend in-context learning to relational tabular data and incorporate semantic embeddings [8], [9].

However, these models are not designed for event logs. They operate on row-based structures and thus cannot directly exploit the sequential dependencies, control-flow patterns, and timestamp relationships inherent to process executions.

Our work introduces a foundation model specifically for event-log sequences. Rather than flattening prefixes into tabular rows, the model encodes the sequence of events, enabling richer extraction of control-flow and temporal cues.

C. FEW-SHOT AND METRIC-BASED META-LEARNING

Prototypical networks [15] represent classes through prototypes in an embedding space and perform prediction via distance comparisons. They have been influential in few-shot learning due to their simplicity and effectiveness.

Our method integrates prototypical classification and regression heads with a transformer-based prefix encoder trained on event logs. At inference time, a small support set from the target log defines the context for in-situ adaptation, combining ideas from meta-learning with process-mining-specific sequential modeling.

III. PRELIMINARIES

The following preliminaries formalize event data structures and core prediction problems.

A. EVENT LOGS, CASES, TRACES, AND ATTRIBUTES

We work with event data produced by information systems. Intuitively, a case records one execution of a process; the history of a case is a time-ordered list of events; and an event log is a finite collection of such case histories. This subsection formalizes these objects and fixes the notation used throughout the paper. Let Σ be an alphabet of strings and let $A \subseteq \Sigma$ denote the (finite) set of activity labels present in the current log. We write E for the universe of events. Each event $e \in E$ comes with projections for the mandatory fields $\pi_{\text{act}} : E \to A$ and $\pi_{\text{time}} : E \to \mathbb{R}$, mapping e to its activity label and its timestamp, respectively. Optional attributes are represented as partial maps indexed by attribute names. Let $N_{\mathbb{R}}$ and N_{Σ} be finite sets of names for numeric and string attributes. For every $e \in E$ we use $\pi^{\mathbb{R}}(e) : N_{\mathbb{R}} \rightharpoonup \mathbb{R}$ and $\pi^{\Sigma}(e) : N_{\Sigma} \rightharpoonup \Sigma$ to denote the (possibly undefined) value of each named numeric or string attribute on e. When an attribute is missing, its projection is undefined; in later components this is handled explicitly but no special convention is needed here. A trace (or case history) is a finite sequence of events ordered by time, $\tau = (e_1, \ldots, e_T)$ with $T \in \mathbb{N}_{\geq 1}$, such that timestamps are strictly increasing within the sequence: $\pi_{\text{time}}(e_i) < \pi_{\text{time}}(e_{i+1})$ for all $i \in \{1, \ldots, T-1\}$. If the source system produces equal timestamps for consecutive events, we assume a stable tie-break that induces a strict order; none of the results depends on how ties are broken. We write $|\tau| = T$ for the length of a trace and $\text{last}(\tau) = e_T$ for its last event. An event log L is a finite set of traces, $L \subseteq \bigcup_{T \geq 1} E^T$, where each trace in L corresponds to one case. By construction, the activity set A is exactly the set of activity labels observed in L, i.e.,

$$A := \{\pi_{\text{act}}(e) : e \in \tau,\ \tau \in L\}.$$

Unless stated otherwise we do not compare timestamps across different traces; time is only ordered within a case.

B. PREFIXES OF TRACES

Predictions in this work are conditioned on what has happened so far in a running case. Given a trace $\tau = (e_1, \ldots, e_T)$ in time order, the prefix of length m,

with $1 \le m \le T$, is the initial segment $\tau_{\le m} = (e_1, \dots, e_m)$. We use $|\tau_{\le m}| = m$ for its length and keep the time order inherited from τ. The case is non-complete at position m when $m < T$; in that situation the next event in the full trace is e_{m+1} and its activity label $\pi_{\text{act}}(e_{m+1})$ becomes the target for next-activity prediction introduced later. The last timestamp observed in the prefix is $\pi_{\text{time}}(e_m)$, which serves as the origin for remaining-time prediction. It is useful to make simple elapsed-time quantities explicit on prefixes. The time since the start of the case at position m is $\Delta_{\text{start}}(e_m) = \pi_{\text{time}}(e_m) - \pi_{\text{time}}(e_1)$, and the time since the previous event is $\Delta_{\text{prev}}(e_m) = 0$ if $m = 1$ and $\pi_{\text{time}}(e_m) - \pi_{\text{time}}(e_{m-1})$ if $m \ge 2$. These quantities depend only on the information contained in the prefix and will be reused as basic numeric features. A prefix is determined entirely by its first m events and their attributes. When we write functions of a prefix, we will often suppress explicit mention of τ and m if the context is unambiguous; for example, we write Δ_{start} and Δ_{prev} for the values at the current last event in the prefix.

C. PREDICTIVE TASKS

We study two standard predictive questions on non-complete prefixes. Let $\tau = (e_1, \dots, e_T)$ be a trace in a log L, let $1 \le m < T$, and write $\tau_{\le m} = (e_1, \dots, e_m)$ for its prefix. The set of admissible inputs for prediction is the collection of all such prefixes, denoted $\text{Pref}^\circ(L)$.

The first task is next-activity prediction. The goal is to determine which activity occurs immediately after the prefix. Formally, the target is the map

$$a^\star \colon \text{Pref}^\circ(L) \longrightarrow A, \qquad a^\star(\tau_{\le m}) := \pi_{\text{act}}(e_{m+1}),$$

where A is the set of activity labels observed in the log (cf. Section III-A). The learning problem later associates to each $\tau_{\le m}$ a probability distribution over A supported on the labels relevant to the episode.

The second task is remaining-time prediction. The aim is to estimate how much time remains from the last event in the prefix to the completion of the case. With $t_m := \pi_{\text{time}}(e_m)$ and $t_T := \pi_{\text{time}}(e_T)$, the target is the nonnegative map

$$r^\star \colon \text{Pref}^\circ(L) \longrightarrow \mathbb{R}_{\ge 0}, \quad r^\star(\tau_{\le m}) := t_T - t_m.$$

To stabilize heavy-tailed durations we may work on the transformed scale $\tilde{r}^\star = \psi\big(r^\star\big)$ with $\psi(x) = \log(1 + x)$ and recover original units by $\psi^{-1}(z) = \exp(z) - 1$ when reporting results.

Both targets depend only on information contained in the prefix together with, for $r^\star$, the end-of-case timestamp used here for definition and evaluation. Predictions are not defined for complete prefixes ($m = T$) and such inputs are excluded from $\text{Pref}^\circ(L)$.

D. VECTOR INTERFACES FOR EVENTS AND PREFIXES

We represent each prefix by a fixed-dimensional embedding and compare embeddings through a similarity function used by the in-context heads.

Let $d_\Phi, d \in \mathbb{N}$ denote the event-vector and prefix-embedding dimensions. An event encoder is a map $\Phi : E \to \mathbb{R}^{d_\Phi}$ and a prefix encoder is a map $\Omega : (\mathbb{R}^{d_\Phi})^* \to \mathbb{R}^d$. For a prefix $\tau_{\le m} = (e_1, \dots, e_m)$ we write

$$y(\tau_{\le m}) := \Omega\big(\Phi(e_1), \dots, \Phi(e_m)\big) \in \mathbb{R}^d.$$

To compare prefixes we use a similarity function $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, where larger values indicate more similar embeddings. All subsequent episode, prototype, and kernel constructions depend on prefixes only through $y(\cdot)$ and $\kappa(\cdot, \cdot)$. Section IV specifies concrete choices for Φ, Ω, and κ.

E. IN-CONTEXT EPISODES AND ADAPTATION

We formalize adaptation to a new log through small, self-contained prediction problems called episodes. Each episode provides a tiny labeled context drawn from the target log and a disjoint set of query prefixes. Crucially, in realistic monitoring, and in many test-time settings, the targets of query prefixes (the next activity and the remaining time) are future quantities and therefore unknown at prediction time. We thus distinguish between (i) the observable query inputs used for inference and (ii) the corresponding targets, which are available only in supervised training or offline evaluation on fully observed traces.

Let $d \in \mathbb{N}$ be the prefix-embedding dimension and write $\mathcal{Y} = \mathbb{R}^d$. Using the interfaces of Section III-D, every prefix $\tau_{\le m}$ is mapped to a vector $y(\tau_{\le m}) \in \mathcal{Y}$. An episode is a pair (S, Q) with

$$S = \{(y_i, t_i)\}_{i=1}^K \quad \text{and} \quad Q = \{y'_j\}_{j=1}^J,$$

where $y_i, y'_j \in \mathcal{Y}$ are embeddings of distinct prefixes from the same target log, and t_i are task-specific targets as defined in Section III-C. For next-activity prediction one has $t_i \in A$, while for remaining-time prediction one has $t_i \in \mathbb{R}_{\ge 0}$ (optionally on the transformed scale ψ). Inputs are disjoint in the sense that the prefixes that yield $\{y_i\}$ and $\{y'_j\}$ share no events.

When supervised signals for the queries are available (during episodic training, or when evaluating on a fully observed log), we denote the corresponding (held-out) query targets by $\{t'_j\}_{j=1}^J$ and may write the labeled query set as

$$Q = \{(y'_j, t'_j)\}_{j=1}^J.$$

However, the predictor never receives t'_j as input: at inference time the model conditions only on S and on the query embeddings $\{y'_j\}$, while t'_j (if known at all) is used solely to compute losses or evaluation metrics.

The support set S is the context of the episode. It defines the information available for adaptation and fixes the local scope of the prediction. In classification we use the local label set

$$A_S := \{t_i : (y_i, t_i) \in S\} \subseteq A,$$

and predictions for a query embedding y' are distributions supported on A_S. In regression the numeric scale is likewise local: the remaining-time estimate for y' is computed relative

to the values $\{t_i\}$ present in S and, if ψ is used, is reported back on the original scale via ψ^{-1}.

By in-context adaptation we mean that the predictor for queries takes the form of a mapping $f(\cdot \mid S)$ whose behavior depends on S but not on any parameter update. Concretely, for classification one obtains a conditional distribution $f_{\text{cls}}(y' \mid S)$ over A_S, and for regression a scalar estimate $f_{\text{reg}}(y' \mid S) \in \mathbb{R}_{\geq 0}$; in both cases S is an explicit argument of the predictor at inference time.

Section IV-F details how we build S in episodic and retrieval modes. An overview of how the shared mixture-of-experts processes an episode for both next-activity classification and remaining-time regression is shown in Figure 2.

IV. APPROACH

This section presents the architecture and learning paradigm of the proposed foundation model. We build on the formal interfaces introduced in the preliminaries and describe how the implemented system instantiates them.

A. ARCHITECTURE OVERVIEW: MIXTURE-OF-EXPERTS FOUNDATION MODEL

The model is a mixture-of-experts (MoE) with E experts. Each expert e implements its own event encoder $\Phi^{(e)}$ and prefix encoder $\Omega^{(e)}$, thus inducing an embedding function $y^{(e)}(\tau_{\leq m}) \in \mathbb{R}^d$ as defined in Section III-D. Given an episode support set S from the target log (formalized in Section III-E), each expert predicts for a query prefix using two nonparametric, in-context heads: (i) a prototype classifier for next-activity prediction and (ii) a kernel regressor for remaining-time prediction (Section IV-D).

Training uses episodes sampled across heterogeneous logs. Each training step routes one full episode to a single randomly chosen expert and updates only that expert. At inference time, all experts process the same query and support set in parallel; their outputs are combined using confidence-aware aggregation (Section IV-G2). Episode construction for training and test-time context is described in Section IV-E and Section IV-F.

We route whole episodes to a single randomly chosen expert during training, updating only that expert; at inference, all experts are evaluated and their predictions are aggregated. Episode construction is detailed in Section IV-E and Section IV-F, while the aggregation rules are given in Section IV-G2. A schematic comparison of training-time single-expert updates versus inference-time all-expert aggregation is provided in Figure 9.

B. EVENT-TO-VECTOR MAPPING Φ

The event-to-vector mapping Φ produces a fixed-size vector for each event, ready for the prefix encoder Ω. The concrete decomposition of Φ into basic temporal/numeric features and an optional string/semantic channel is illustrated in Figure 3. It is open-vocabulary, applying uniformly across processes, and fuses two channels: an optional string channel for attributes like activity names or resources, and a numeric channel for features such as elapsed times.

The string channel embeds strings so that semantically related terms (e.g., ‘‘approve’’, ‘‘authorize’’) map nearby when semantics are enabled, aiding transfer across logs with variant naming. This is optional and can be regularized via episode-level label shuffling during training (Section IV-E3): with probability p_{shuffle}, we apply a random permutation of activity labels within an episode (to all events in support/query prefixes and to the next-activity targets), which breaks any fixed global mapping from activity names to labels and forces support-set-based meaning.

Semantics help when names follow consistent conventions or span languages, clustering related labels pre-context. They are often unnecessary with shuffling, as structure and timing suffice, and can be disabled for noisy or idiosyncratic names, yielding a structure-only mode where strings act as opaque IDs. Predictions then depend solely on event order (via Ω), numeric features, and the support set. Two options exist: a frozen pretrained text encoder for stable cross-log semantics (fusing activity and resource vectors, e.g., by addition), or a learned character/subword encoder robust to typos and styles like ‘‘ApproveInvoice’’ or ‘‘approve_invoice’’.

The numeric channel encodes universal signals like time since case start (Δ_{start}), time since previous event (Δ_{prev}), and cost (if present). We apply the monotone transform ψ (defined in Section III-C) to heavy-tailed durations.

Channels combine by concatenation followed by a small nonlinear projection and normalization, aligning scales for a compact event vector. This supports unseen strings, retains event-level temporal cues, and works in both semantic and structure-only modes. The prefix encoder sees only these vectors; all task-specific adaptation occurs in-context via support sets.

C. PREFIX ENCODER Ω

The prefix encoder turns a variable-length sequence of event vectors into a single vector that summarizes the state of the case so far. The resulting prefix-to-vector mapping $y(\tau_{\leq m}) = \Omega(\Phi(e_1), \ldots, \Phi(e_m))$, including the summary token and attention-style pooling used to obtain a fixed-dimensional prefix embedding, is depicted in Figure 4. It must remember order, focus on the few events that matter for the next decision, and work for short and long prefixes alike. We use an attention-based encoder with a small learned summary token and a pooling mechanism that produces one fixed-size vector per prefix.

1) ORDER ENCODING

The encoder first prepares the sequence so that order is explicit and padding is ignored.

- Learned summary token. A special vector is added at the front of every sequence. It plays the role of a global collector that accumulates information from the whole prefix.

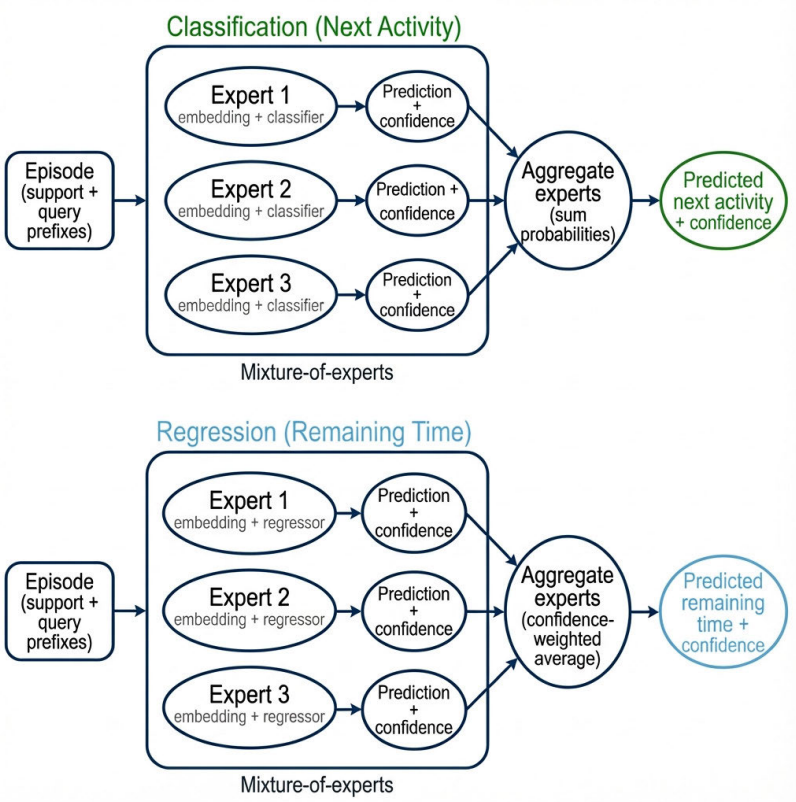


FIGURE 2. Shared mixture-of-experts for classification and regression. Episodes consisting of support and query prefixes are processed by a single mixture-of-experts encoder. For next-activity prediction (top), each expert applies a prototype-based classifier and returns a class prediction with a confidence score; the model aggregates expert outputs by summing their class probabilities and predicts the most probable next activity. For remaining-time prediction (bottom), the same experts use a kernel-style regressor and produce time estimates with confidence scores; these are combined through a confidence-weighted average to obtain the final remaining-time prediction.

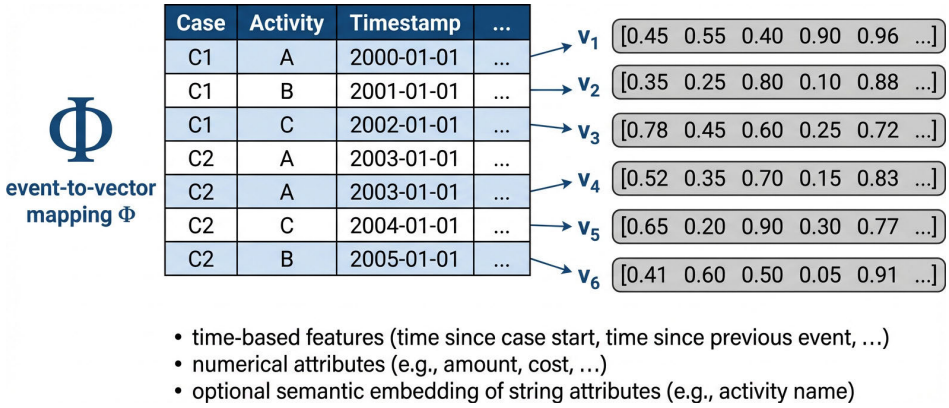


FIGURE 3. Event-to-vector mapping (Φ). Each row of the event log (case, activity, timestamp, and additional attributes) is mapped by Φ to a fixed-dimensional vector $\mathbf{v}_i$. The vector components combine simple temporal features (e.g., time since case start and since the previous event), numeric attributes, and (optionally) semantic embeddings of string attributes such as the activity name.

- Positional signal. Each event vector receives a position-specific signal so the model can tell whether an event happened early or late in the case and how far apart events are.
- Masking of padding. Prefixes have different lengths. We keep a mask that marks padded positions. The attention layers respect this mask so that padding never influences the summary.

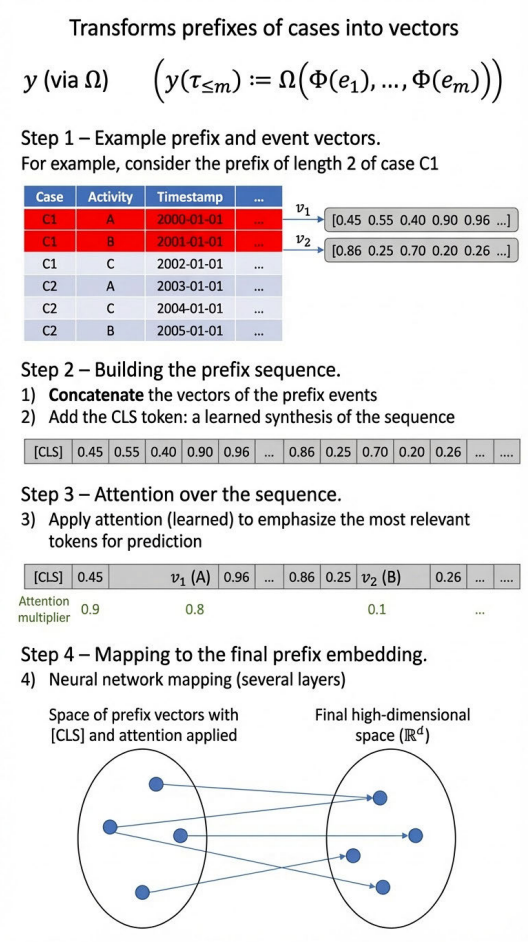


FIGURE 4. Prefix-to-vector mapping (Ω). Given the events of a case prefix $\tau_{\leq m}$, the model first maps each event to an event vector $\Phi(e_i)$ and defines the prefix embedding as $y(\tau_{\leq m}) = \Omega(\Phi(e_1), \ldots, \Phi(e_m))$. The example shows the prefix of length $m = 2$ for case C1: the two event vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ are stacked into a length-2 sequence, a learned summary ([CLS]) token is prepended, and the sequence is processed by transformer layers. The final prefix embedding is obtained by combining the summary token with an attention-pooled vector (as described in Section IV-C2) and projecting to a fixed-dimensional embedding $y(\tau_{\leq m}) \in \mathbb{R}^d$.

With these ingredients the encoder can compare and combine events while preserving their order and spacing.

2) CONTEXT AGGREGATION

After order is made explicit, the sequence passes through a stack of attention layers. Each layer lets an event look at other events and refine its representation based on what came before and after.

To reduce the refined sequence to a single vector, we use a two-part pooling step:

- Attention pooling with a learned query. A small learned query attends to all refined event vectors and returns a pooled vector that emphasizes the most relevant positions for the current prefix.
- Hybrid merge with the summary token. The pooled vector is combined with the learned summary token from the front of the sequence. The merge is done by a simple projection that learns how much to trust the global summary versus the attention pool. A light normalization stabilizes the scale.

The result is one fixed-size prefix embedding. It is order-aware, it highlights informative events, and it is stable across different prefix lengths. The same embedding is then used by the in-context heads for both next-activity and remaining-time prediction, and it supports nearest-neighbor retrieval when building the support set at test time.

D. IN-CONTEXT PREDICTION HEADS

The two nonparametric in-context heads are illustrated in Figure 5 (prototype classifier for next-activity prediction[1]) and Figure 6 (kernel regressor for remaining time[2]).

Each expert predicts from a query prefix embedding y' conditioned on an episode support set S (cf. Section III-E) without any parameter update. The heads depend on S only through support embeddings and targets and a similarity measure in the embedding space.

1) PROTOTYPE-BASED NEXT-ACTIVITY PREDICTION

Let $S = \{(y_i, a_i)\}_{i=1}^K$ be support pairs with activity labels and let A_S be the induced local label set (Section III-E). For each activity $a \in A_S$ with index set $I_a = \{i : a_i = a\}$, form the prototype

$$p_a = \frac{1}{|I_a|} \sum_{i \in I_a} y_i.$$

Given a query embedding y', compute class scores using cosine similarity with temperature $\beta > 0$:

$$s_a = \beta \cdot \cos(y', p_a), \qquad \hat{p}(a \mid y', S) = \frac{e^{s_a}}{\sum_{b \in A_S} e^{s_b}}.$$

The prediction is $\arg\max_{a \in A_S} \hat{p}(a \mid y', S)$.

2) KERNEL-STYLE REMAINING-TIME REGRESSION

Let $S = \{(y_i, t_i)\}_{i=1}^K$ be support pairs with remaining-time targets (possibly on the transformed scale ψ from Section III-C). For query y', define normalized kernel weights

$$w_i \propto \exp\big(-\gamma \, \|y' - y_i\|^2\big), \qquad \sum_{i=1}^K w_i = 1,$$

[1]For clarity, the illustration shows a single embedding space with one prototype-based head. In the implemented model, each expert in the mixture has its own event encoder and prefix encoder, forms its own prototypes from the local support set, and the final next-activity prediction is obtained by aggregating the per-expert class probabilities as described in Section IV-A.

[2]The distances and remaining times in the figure are purely illustrative and drawn on the raw time scale. In the actual approach, each expert operates in its own prefix-embedding space, kernel weights are computed per expert and aggregated, and remaining times are often modeled on a transformed scale $\tilde{r} = \psi(r)$ (e.g., $\psi(x) = \log(1 + x)$) before being mapped back to seconds, as detailed in Section III-C and Section IV-D2.

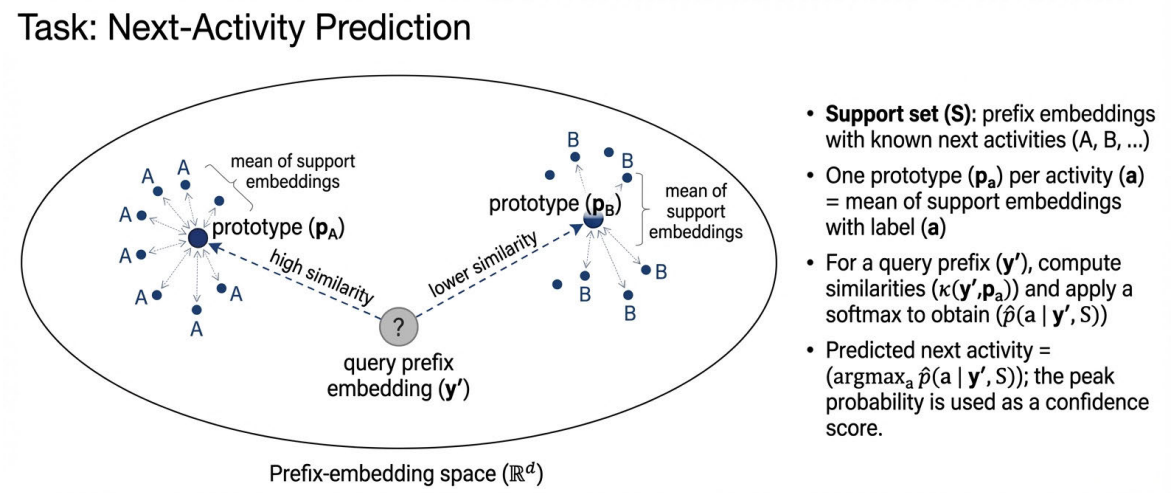


FIGURE 5. Prototype-based head for next-activity prediction. In the prefix-embedding space $\mathbb{R}^d$, the support set S for a new log provides labeled prefix embeddings (here with next activities A and B). For each activity a in the local label set, the model forms a prototype $\mathbf{p}_a$ by averaging the embeddings of its support examples. Given a query prefix embedding $\mathbf{y}'$, similarities $\kappa(\mathbf{y}', \mathbf{p}_a)$ to the prototypes are converted by a softmax into a posterior $\hat{p}(a \mid \mathbf{y}', S)$. The predicted next activity is the label with the highest posterior probability, and the peak probability serves as a confidence score.

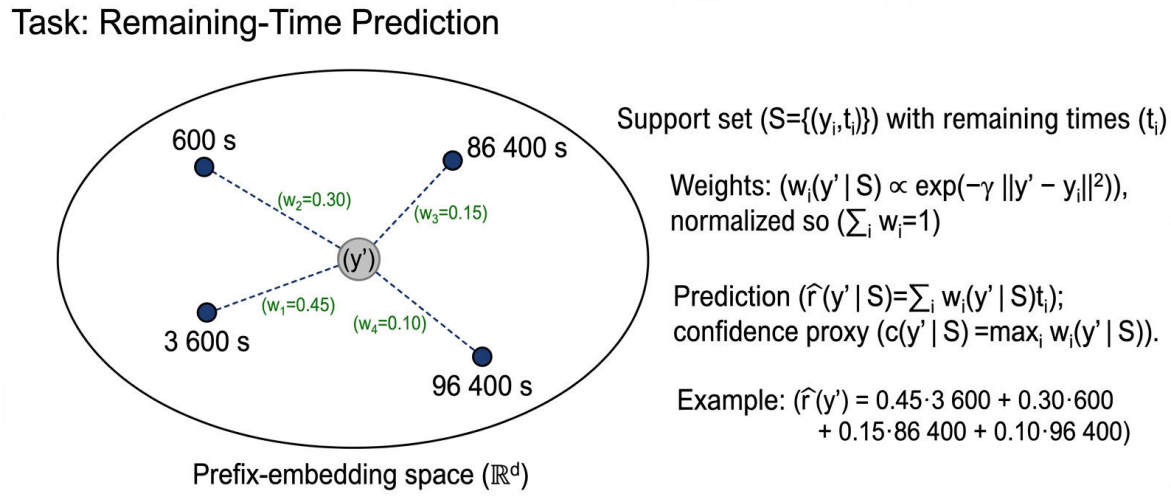


FIGURE 6. Kernel regression head for remaining-time prediction. In the prefix-embedding space $\mathbb{R}^d$, each support prefix $\mathbf{y}_i$ from the target log is paired with its observed remaining time t_i. For a query prefix embedding $\mathbf{y}'$, the model assigns kernel weights $w_i(\mathbf{y}' \mid S)$ that decrease with the squared distance between $\mathbf{y}'$ and $\mathbf{y}_i$ and are normalized to sum to one. The predicted remaining time is the similarity-weighted average $\hat{r}(\mathbf{y}' \mid S) = \sum_i w_i(\mathbf{y}' \mid S)\, t_i$, while the maximum weight $\max_i w_i(\mathbf{y}' \mid S)$ serves as a simple confidence signal.

with γ set by a robust median-distance heuristic to avoid per-log tuning. The prediction is

$$\hat{t}(y' \mid S) = \sum_{i=1}^{K} w_i\, t_i,$$

followed by ψ^{-1} if a transformed target is used.

3) CONFIDENCE SIGNALS

Both heads provide a confidence signal that is reused by the MoE aggregator (Section IV-G2): (i) for classification, the maximum softmax probability $\max_{a \in A_S} \hat{p}(a \mid y', S)$; and (ii) for regression, a kernel-concentration proxy such as $\max_i w_i$ (high when one neighbor dominates, low when weights are diffuse).

E. TRAINING STRATEGIES

We train the model to rely on small, local contexts and to generalize across heterogeneous logs. Training is organized in episodes. Each episode mimics test-time use: a small support set is provided and predictions are made for disjoint queries. At each step, the full episode is routed to a single expert in the mixture. Only that expert is updated. Episodes come in two flavors that target complementary weaknesses. The two episode-construction schemes used throughout training (balanced N-way/K-shot sampling and retrieval-driven

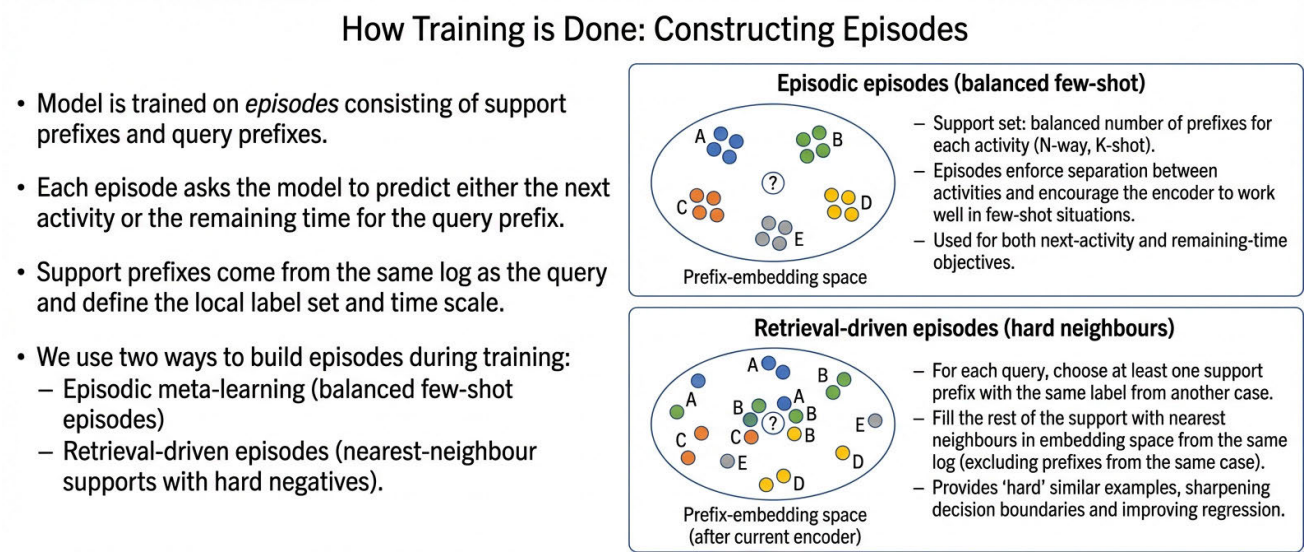


FIGURE 7. Episode construction used for training. The model is trained on episodes composed of support prefixes and query prefixes from event logs, with each episode used for both next-activity and remaining-time objectives. In episodic meta-learning (top right), episodes are built by sampling a balanced number of support prefixes for each activity (N-way, K-shot), encouraging the encoder to separate activities and perform well in true few-shot settings. In retrieval-driven episodes (bottom right), each query is paired with at least one support prefix of the same label and additional nearest neighbours from the same log (excluding prefixes from the same case), producing an imbalanced but locally informative context with many "hard" neighbours. Both strategies expose the shared encoder to diverse episode structures while keeping the prediction heads purely in-context.

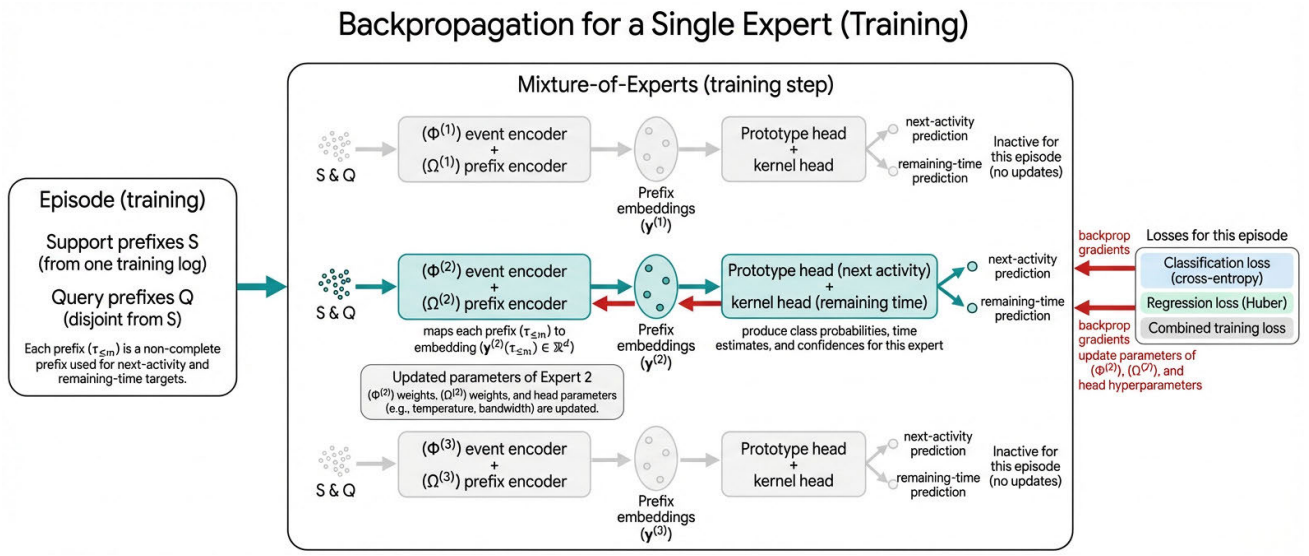


FIGURE 8. Backpropagation for a single expert during training. During training, an episode consisting of support prefixes S and query prefixes Q from one log is routed to a single expert in the mixture-of-experts. Inside this expert, the event encoder $\Phi^{(e)}$ and prefix encoder $\Omega^{(e)}$ map each prefix $\tau_{\leq m}$ to an embedding $\mathbf{y}^{(e)}(\tau_{\leq m}) \in \mathbb{R}^d$. Nonparametric heads (a prototype-based classifier for next-activity prediction and a kernel regressor for remaining-time prediction) use these embeddings to produce predictions, from which a classification loss and a regression loss are computed and combined. Backpropagation flows from the total loss through the heads and embeddings into the parameters of $\Phi^{(e)}$, $\Omega^{(e)}$, and the small head hyperparameters (e.g., the classification temperature), which are updated by gradient descent. The other experts remain inactive and are not updated for this episode.

neighbor selection) are summarized in Figure 7. The gradient flow and parameter updates within a single expert for one routed episode are depicted in Figure 8.

1) EPISODIC META-LEARNING ACROSS LOGS

This strategy teaches the model to learn from few labeled examples and to respect a local label space.

1) Sample an episode configuration. Choose N and K (and a query budget) and select a training log.
2) Form an episode. Draw a small support set and a disjoint query set from that log. For next-activity prediction, select N activities with at least $K+1$ instances each, take K per class for the support, and reserve at least one per class for queries. We additionally enforce that support

and query prefixes come from different cases to avoid leakage.
3) Predict and learn (multi-task). Using the same episode, compute (i) a cross-entropy loss for next-activity prediction with the prototype head and (ii) a Huber loss (a robust squared-error loss with linear tails) for remaining-time prediction with the kernel head, and optimize a weighted sum of the two losses.

Episodes are sampled across many training logs so the encoder learns patterns that transfer. N and K vary across steps to avoid overfitting to a single regime.

2) RETRIEVAL-DRIVEN HARD-NEGATIVE MINING

This strategy sharpens decision boundaries by constructing supports that are challenging.

1) Create a candidate batch. Draw a large set of prefixes from a training log and compute their embeddings once.
2) Guarantee a positive. For each query in the batch, include at least one support example with the same label from a different case. This prevents degenerate episodes.
3) Add hard neighbors. Fill the rest of the support with the nearest neighbors of the query in the embedding space. Exclude examples from the same case to avoid leakage. These neighbors are often close but belong to other classes, which creates hard negatives for classification and informative contrasts for regression.
4) Predict and learn. Use the nonparametric heads with the constructed support and apply the same losses as above.

By focusing on near neighbors, the encoder learns features that separate subtle variants and improves robustness to label imbalance.

3) LABEL SHUFFLING FOR ROBUST IN-CONTEXT LEARNING

We summarize the two string-channel options for Φ (pretrained text encoder vs. character-level CNN) and the episode-level label-shuffling procedure used as a regularizer in Figure 10. As an optional step to discourage reliance on any global label identity, we apply label shuffling with probability p_{shuffle}.

F. INFERENCE-TIME CONTEXT CONSTRUCTION

At test time the model adapts to a new log by building a small support set directly from that log and using it as context to predict for unseen query prefixes. In an operational monitoring setting, the support is built only from completed historical cases (so labels are known), while queries are taken from ongoing cases (so targets are unknown at prediction time); Section IV-F4 formalizes this realistic split and the time-windowed candidate pool used for per-query selection.

1) META-LEARNING MODE (N-WAY, K-SHOT)

This mode mirrors the few-shot setting used during training and is useful for controlled evaluation.

1) Collect candidates. From the target log, extract prefix examples and group them by next-activity label.
2) Select classes and shots. Choose N activities that have at least $K+1$ examples each. For every chosen activity, sample K examples for the support and one or more examples for the queries.
3) Construct the episode. The local label space is the set of the N chosen activities. The support S and queries Q are disjoint and drawn only from the target log.
4) Prevent leakage. Ensure that support and query prefixes come from different cases (no shared events), analogous to the same-case masking used in retrieval mode.
5) Predict in context. For each query, the heads use only S to produce a next-activity distribution over the N activities or a remaining-time estimate.

This procedure evaluates how well the model adapts when the label space is explicitly defined by a small sample from the target process.

2) RETRIEVAL-AUGMENTED MODE

This mode builds the support for each query by searching within the target log itself. It requires no class balancing and reflects a more operational scenario. Importantly, although the mixture contains multiple experts, the nearest-neighbor retrieval is performed using the embedding space of one fixed expert only. Once the neighbor set is retrieved, the resulting support prefixes are shared with all experts for prediction and confidence-based aggregation.

1) Fix a retrieval expert. Select a single expert index $e_{\text{ret}} \in \{1, \ldots, E\}$ (fixed for the whole run) and use its embedding function $y^{(e_{\text{ret}})}(\cdot)$ only to perform retrieval. This avoids running E separate k-NN searches and ensures a unique, deterministic support set per query.
2) Precompute retrieval embeddings. Encode all candidate prefixes from the target log with the retrieval expert, normalize the resulting embeddings, and store them together with their labels for both tasks.
3) Nearest-neighbor selection (single-expert). For a query prefix, compute its normalized embedding with the same retrieval expert and retrieve the K nearest neighbors in that single embedding space (cosine similarity on normalized embeddings) to form the support S.
4) Avoid contamination. Exclude candidates that belong to the same case as the query so the support does not reveal the future of the query case.
5) Predict in context (all experts). Pass the retrieved support prefixes S (as prefix identities, not as expert-specific embeddings) to the full mixture. Each expert e re-encodes the query and the selected support prefixes with its own $\Phi^{(e)}$ and $\Omega^{(e)}$, applies the nonparametric heads, and the mixture aggregates the resulting predictions as in Section IV-G2.

The value of K controls how much context is used. Retrieval naturally adapts to logs with skewed label frequencies and works with or without the semantic string channel, while still benefiting from multi-expert prediction through aggregation.

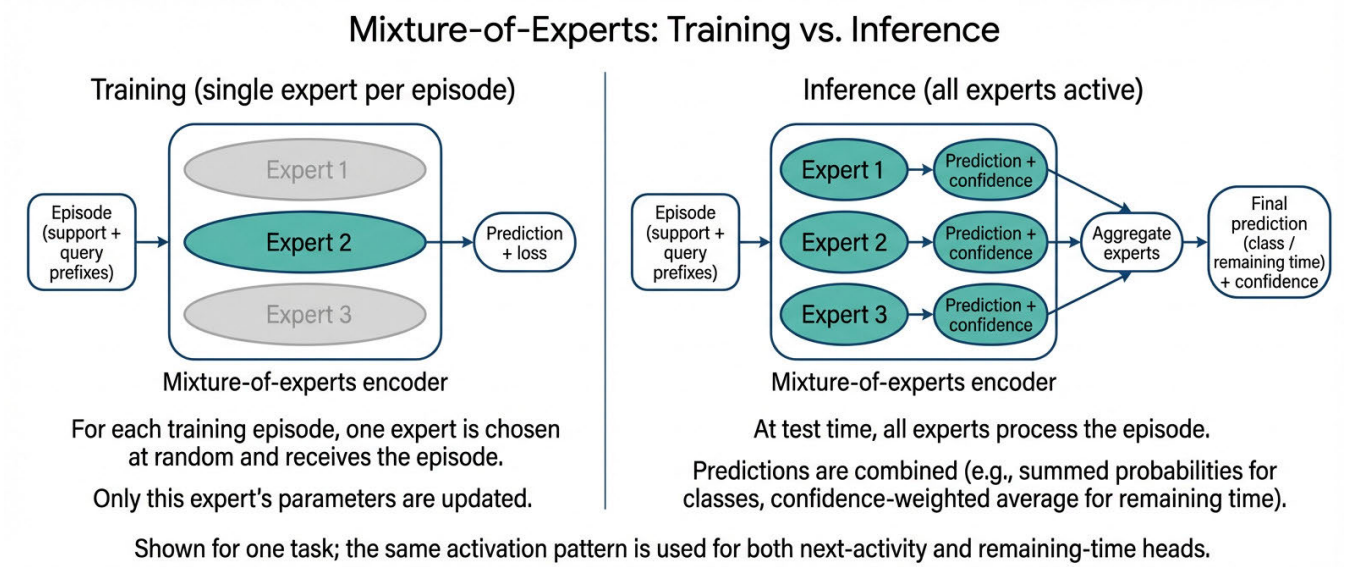


FIGURE 9. Mixture-of-experts behaviour during training and inference. During training (left), each episode is routed to a single expert chosen at random; only that expert produces a prediction and receives gradient updates, while the other experts remain inactive. During inference (right), all experts process the same episode in parallel and each returns a prediction with a confidence signal. Their outputs are then aggregated, by summing class probabilities for next-activity prediction or by a confidence-weighted average for remaining-time prediction, to obtain the final prediction and confidence.

3) LOCAL LABELING AND ACTIVITY ALIGNMENT

Predictions operate in a local label space derived from the target log. We first build the set of distinct next-activity labels in the log and assign them contiguous local IDs, decoupling from training-time identities. In meta-learning mode, the label space comprises activities in the episode support; in retrieval mode, it uses those in the query's retrieved support, with the classification head outputting probabilities solely over these. Predicted local IDs map back to original activity names for reporting, enabling handling of unseen or variant labels from training.

4) OPERATIONALIZING IN-CONTEXT ADAPTATION IN A REALISTIC MONITORING SETTING

The context-construction procedures in Section IV-F1,IV-F2 specify how to pick a small support set once a pool of candidate prefixes is available. In a realistic deployment, however, not all prefixes in the log are eligible as labeled support at prediction time: labels for next activity and remaining time are only known for completed cases. Therefore, operational monitoring requires a chronological split of the target (test) log into (i) historical completed cases that provide labeled support prefixes and (ii) ongoing cases that provide query prefixes whose targets are not yet observed. Figure 11 illustrates this split and the time-windowed candidate pool used to select support points for each query.

a: CHRONOLOGICAL AVAILABILITY MODEL

We model monitoring at a query time t_q (the timestamp of the last observed event of the query prefix), but we do not infer case completion from time alone. Instead, we assume a configurable set of end activities $A_{\text{end}} \subseteq A$ (e.g., `Close`, `Complete`, `End`) or an equivalent explicit completion flag (e.g., a lifecycle/status attribute). For a case $\tau = (e_1, \ldots, e_T)$, let $m(\tau, t_q)$ be the largest index such that $\pi_{\text{time}}(e_{m(\tau,t_q)}) \leq t_q$ (with $m(\tau, t_q) = 0$ if the case has not started by t_q). We call the case completed by t_q iff the observed history up to t_q contains an end marker, i.e.,

$$\exists j \leq m(\tau, t_q) \text{ such that } \pi_{\text{act}}(e_j) \in A_{\text{end}},$$

and we define the completion time as the timestamp of the first such end event, $t_{\text{end}}(\tau) := \min\{\pi_{\text{time}}(e_j) : j \leq m(\tau, t_q),\ \pi_{\text{act}}(e_j) \in A_{\text{end}}\}$. Support prefixes are then drawn from cases that are completed by t_q, using only prefixes $\tau_{\leq m}$ with $\pi_{\text{time}}(e_m) < t_{\text{end}}(\tau)$, so that both next-activity labels and remaining-time targets are well-defined from the observed end event. Query prefixes are taken from cases that have no end marker observed by t_q (ongoing cases).

b: SUPPORT-PREFIX STORE FROM COMPLETED CASES

From each completed historical case $\tau \in L_{\text{hist}}(t_q)$, we can extract labeled support prefixes $\tau_{\leq m}$ for $1 \leq m < T$. Each such prefix is paired with: (i) the next-activity label $a^\star(\tau_{\leq m})$ and (ii) the remaining-time target $r^\star(\tau_{\leq m})$ (cf. Section III-C), both computable because the full case is known (the case has completed). Operationally, we maintain a repository (or index) of historical labeled prefix examples:

$$\begin{aligned}\mathcal{P}(t_q) := \big\{&(\tau_{\leq m}, a^\star(\tau_{\leq m}),\\ &r^\star(\tau_{\leq m}), t_m) : \tau \in L_{\text{hist}}(t_q),\\ &1 \leq m < |\tau|\big\},\end{aligned} \tag{1}$$

where $t_m := \pi_{\text{time}}(e_m)$ is the timestamp of the last event in the prefix. The model does not update parameters; it adapts only through the selected subset of $\mathcal{P}(t_q)$ used as context.

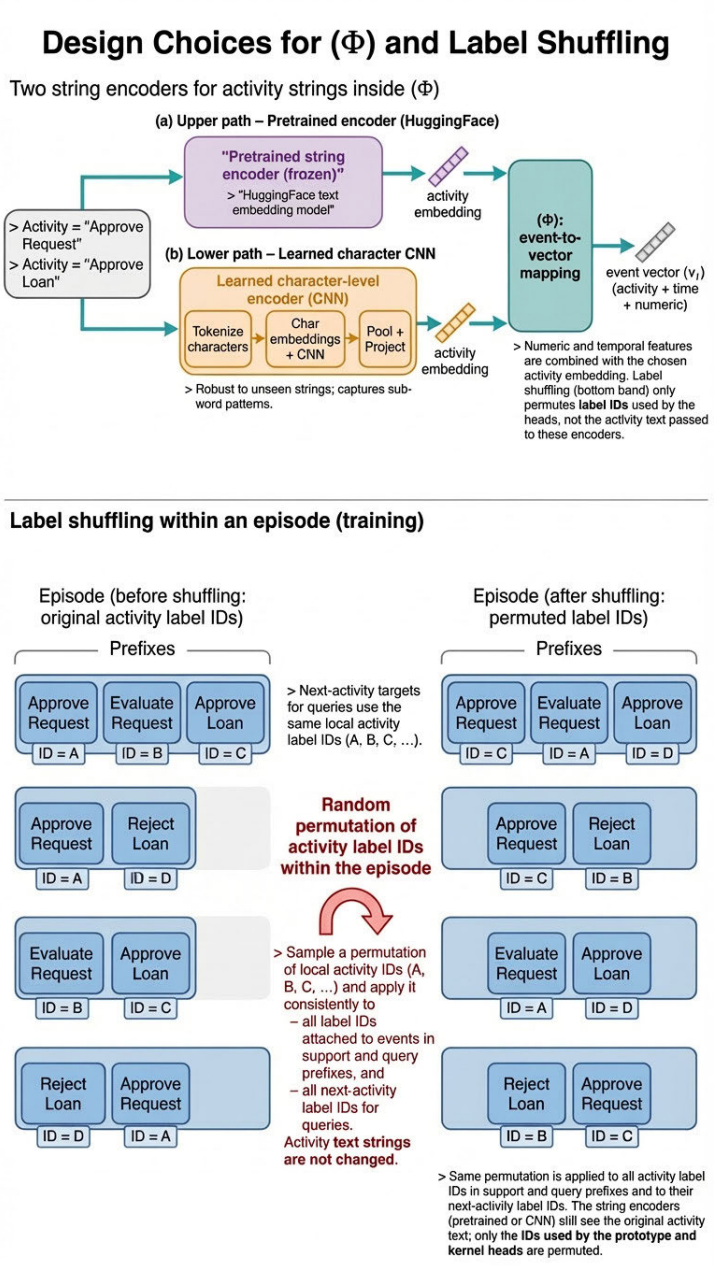


FIGURE 10. String-channel design choices for Φ and episode-level label shuffling. The event-to-vector map Φ combines temporal and numeric features with an embedding of the activity label. The string channel can either use a pretrained text encoder (a frozen HuggingFace model) or a learned character-level CNN that tokenizes characters, applies character embeddings and convolutions, pools, and projects to a fixed-size activity embedding, providing robustness to unseen strings and capturing sub-word patterns. During training, we optionally apply label shuffling at the episode level: for each episode we draw a random permutation of the activity labels and apply it consistently to every event's activity in all support and query prefixes, as well as to the corresponding next-activity targets. This forces the prototype and kernel heads to infer label meaning from the local support set and the learned embeddings, rather than relying on a fixed global identity for each activity name, and works with either choice of string encoder.

c: QUERIES FROM ONGOING CASES

A query is a prefix observed for an ongoing case at time t_q. Formally, for a running case trace $\tau = (e_1, \ldots, e_T)$ we observe only the prefix $\tau_{\leq m_q}$ where $m_q = \max\{m : \pi_{\text{time}}(e_m) \leq t_q\}$ and $m_q < T$. At prediction time, $a^\star(\tau_{\leq m_q})$ and $r^\star(\tau_{\leq m_q})$ are unknown to the predictor and are used only for offline evaluation when the full trace is later available.

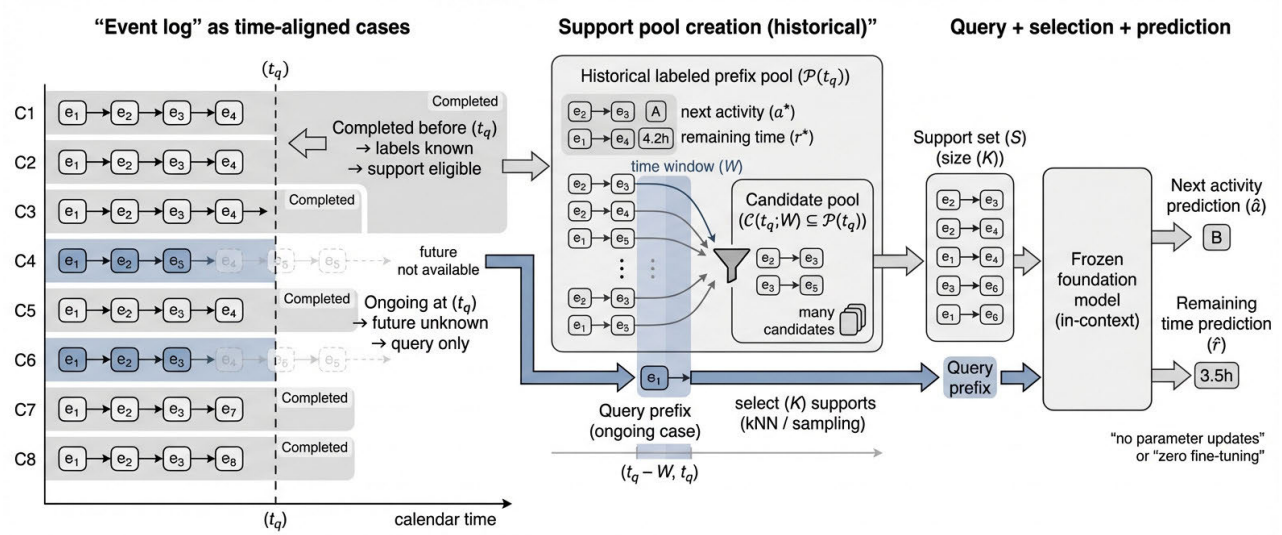


FIGURE 11. Operational split of a test log into historical support and ongoing queries. At a query time t_q (the timestamp of the last observed event of the running case), all *completed* cases with completion time $< t_q$ are treated as historical and are used to extract labeled support prefixes. Query prefixes are taken from *ongoing* cases at time t_q. To mimic realistic monitoring and handle drift, the candidate support pool can be restricted to a sliding time window $[t_q - W, t_q)$ before selecting the final K support prefixes (e.g., by nearest-neighbor retrieval in the embedding space).

d: TIME-WINDOWED CANDIDATE POOL (REALISTIC PER-QUERY SELECTION)

In production, the historical repository $\mathcal{P}(t_q)$ may be very large. Moreover, process behavior can drift over time (seasonality, policy changes, staffing changes). To reflect realistic selection and to bound the candidate set per query, we form a time-windowed candidate pool

$$\mathcal{C}(t_q; W) := \big\{(\tau_{\leq m}, a^\star, r^\star, t_m) \in \mathcal{P}(t_q) : t_q - W \leq t_m < t_q\big\}, \quad (2)$$

where W is a window length (e.g., last week/month/quarter, depending on the process cadence). This matches the operational requirement that "the selection for every query prefix is made among a large number of points in a time range preceding the current prefix": we first gather many historical candidate prefixes by time, then select a small context of size K.

Finally, we apply one of the context-selection rules from Section IV-F1,IV-F2 on $\mathcal{C}(t_q; W)$:

- **Meta-learning mode:** sample a balanced N-way/K-shot support from $\mathcal{C}(t_q; W)$ (when a controlled label space is desired).
- **Retrieval mode:** retrieve the K nearest neighbors of the query within $\mathcal{C}(t_q; W)$ in the embedding space (cosine similarity), then pass those support prefixes to all experts for prediction and aggregation.

e: LEAKAGE AVOIDANCE AND INCREMENTAL OPERATION

This operational split prevents information leakage by construction: support labels come only from cases completed before t_q, and the query case cannot appear in the support pool because it is not completed yet. In an online system, $\mathcal{P}(t)$ and any retrieval index can be updated incrementally when cases complete (new labeled prefixes become available), while queries are answered continuously for running cases at their current prefixes.

G. MoE-SPECIFIC LEARNING AND INFERENCE

The mixture-of-experts (MoE) design allows several experts with the same structure to learn different behaviors. The training loop encourages diversity without a gating network, and the inference rule combines expert outputs using simple confidence signals.

1) EXPERT SPECIALIZATION AND COVERAGE

During training, each episode is routed to a single randomly chosen expert, which is the only one updated. Over many episodes and logs, this fosters natural specialization: experts diverge across data regimes (e.g., short vs. long traces, simple vs. branching control-flow, or naming conventions), task emphases (some leaning toward classification signals, others regression, while all supporting both heads), episode types (retrieval exposing hard negatives for boundary sharpening, episodic enforcing local reasoning), and label handling (shuffling promoting support-set reliance over global identities).

At test time, all experts process the shared support set and query, building independent prototypes or kernel weights before the mixture aggregates their outputs. This ensures robust coverage: mismatched experts are compensated by others.

2) AGGREGATION RULES AND THEIR PROPERTIES

Aggregation operates on the outputs that experts already compute to make in-context predictions. No extra calibration model is required.

a: CLASSIFICATION

Each expert returns a probability vector over the episode label set A_S. The mixture computes the elementwise sum

$$\tilde{p}(a) = \sum_{e=1}^{E} p^{(e)}(a), \quad a \in A_S,$$

normalizes $\tilde{p}$ to a probability distribution, and predicts $\arg\max_{a \in A_S} \tilde{p}(a)$. The maximum entry of the normalized sum serves as an overall confidence. This rule has three properties: it operates only on A_S and does not depend on any global label space; it reduces variance by summing probabilities to smooth biases of single experts; and it reflects expert disagreement in the distribution instead of forcing a choice.

b: REGRESSION

Each expert returns a scalar prediction $\hat{t}^{(e)}$ and a confidence proxy $c^{(e)} \in [0, 1]$ that reflects the concentration of its kernel weights. The mixture forms a confidence-weighted average

$$\hat{t}_{\text{mix}} = \frac{\sum_{e=1}^{E} c^{(e)} \hat{t}^{(e)}}{\sum_{e=1}^{E} c^{(e)} + \varepsilon},$$

with a small ε to handle zero-confidence cases. The mean of $\{c^{(e)}\}$ is reported as an overall confidence. This rule uses these properties: each expert uses the same support set, so confidences compare for the current log; it assigns little weight to low-confidence experts and limits the impact of mismatched specialists; and the final estimate lies within the convex hull of expert predictions to avoid extreme values.

c: EFFECT OF THE OPTIONAL SEMANTIC CHANNEL

Both aggregation rules remain valid whether the semantic string channel is enabled or disabled. When semantics are enabled, some experts may lean on lexical cues. When disabled, experts rely on structure and timing. The mixture balances these preferences through the same confidence-aware combination.

V. IMPLEMENTATION

We implement the proposed foundation model in a modular, end-to-end PyTorch codebase that enables training across heterogeneous event logs and zero-parameter in-context adaptation at inference time. The software is released as a self-contained Python project under GPL-3.0, organized for easy progression from raw event logs to trained models and interpretable outputs (predictions, exported logs) via few entry points. The top level includes end-to-end experiment scripts: `main.py` for single runs; `batch_train.py` and `batch_train2.py` for configuration sweeps; `testing.py` and `batch_test_logs.py` for evaluation; and `xes_out_testing.py` for XES exports inspectable with process-mining tools. Core logic resides in Python modules, `training.py` for the training loops, `data_generator.py` and `collate.py` for inputs/supports, plus directories `components/`, `training_strategies/`, `evaluation/`, `util/`, and `utils/` for building blocks, schedules, metrics, and utilities. `logs` holds inputs; `checkpoints/` stores checkpoints.[3] It assumes a standard scientific Python stack: Python 3, `numpy`/`pandas`, `torch` for deep learning, plus process-mining I/O (e.g., XES) and metrics packages. Logging uses built-in `logging`; summaries are plain text. No databases or external services are needed.

For the empirical comparisons in Section VI, the evaluation code also includes two simple but strong `scikit-learn` baselines trained on the same episode supports used by the in-context heads: (i) a multinomial logistic-regression classifier (`sklearn.linear_model.LogisticRegression`) for next-activity prediction and (ii) a ridge regressor (`sklearn.linear_model.Ridge`) for remaining-time prediction. In episodic evaluation, these baselines are fit once per episode on the episode support set and evaluated on the disjoint query set; in retrieval evaluation, they are fit per query on that query's retrieved neighbor set (after same-case masking). In all cases, the baselines take as input the same fixed-dimensional prefix embeddings used by our model, i.e., the points $y(\tau_{\leq m})$ in the target embedding space produced by the (frozen) encoder at test time, paired with the corresponding next-activity or remaining-time targets.

Configuration is minimal via `config.py`: data paths/tasks/columns/exports; context modes (N-way/K-shot vs. k-NN, shuffling); architecture (experts, dimensions, layers/heads, semantic/structure-only, links); all with defaults for quick starts. A typical run begins with event logs (XES or CSV with case ID, activity, timestamp, optional attributes) in `logs/`, referenced in config. `data_generator.py` reads, timestamps-orders, and prefixes traces with targets, building inference supports; assumptions are minimal beyond IDs/timestamps, with numeric transforms for scale and semantic/ignored strings per config.

Execution is relatively simple: `main.py` loads config, sets up snapshots, curves, metrics, and builds episode iterators via generator. Few-shot episodes sample balanced supports/queries from logs (N-way/K-shot for classification, K for regression); retrieval uses nearest-neighbor supports (same-case masked).

VI. ASSESSMENT

We now assess the proposed approach through experiments and ablations.

A. GENERATION OF EVENT LOGS FOR TRAINING AND TESTING

The assessment uses a controlled set of synthetic event logs generated under a common simulation framework to isolate the in-context foundation model's effects from real-world unusual behaviors, while exposing it to diverse behaviors. All logs share uniform representations (cases, activities, timestamps, optional attributes) but vary in injected

[3] Public code at https://github.com/fit-alessandro-berti/events-transf

TABLE 1. Overview of training and test event logs and processes.

Log	Split	Name	Description
00001	Train	Order-to-Cash	A synthetic event log simulating the Order-to-Cash process, including order validation, credit checks, fulfillment, shipping, invoicing, payment reconciliation, compliance checks, escalations, and exception handling with various control-flow, temporal, resource, and cost dynamics.
00002	Train	Hire-to-Retire	A synthetic process simulating the employee lifecycle from hiring through onboarding, performance management, to retirement or exit, incorporating various control-flow, temporal, resource, and cost patterns.
00003	Train	Disaster-to-Recovery	A synthetic process simulating disaster response and recovery, incorporating assessment, containment, investigation, remediation, governance, and verification with various control-flow, temporal, resource, and cost patterns.
00004	Train	Make-to-Stock	A synthetic event log simulating a make-to-stock manufacturing process involving demand forecasting, planning, procurement, production, quality assurance, packing, stocking, and shipping, with injected patterns in control-flow, timing, resources, and costs.
00005	Train	Offer-to-Acceptance	A synthetic process simulating the journey from offer request capture to acceptance receipt, incorporating various reviews, checks, negotiations, and closures with control-flow patterns, temporal dynamics, resource allocations, and cost implications.
00006	Train	Quote-to-Order	A synthetic business process that covers the full lifecycle from receiving a customer request for quotation (RFQ), through qualification, scoping, configuration, pricing, parallel checks, optional reviews, approval, quote submission, customer negotiation/revision, order conversion, optional vendor outsourcing, and final booking/notification.
00007	Train	Opportunity-to-Quote	A synthetic event log simulating the sales process from opportunity creation through qualification, scoping, assessments, vendor sourcing, pricing, negotiation, and closing, with injected patterns for realism.
00008	Train	Lead-to-Opportunity	A synthetic process simulating the conversion of sales leads into business opportunities, involving lead qualification, data enrichment, demonstrations, proposal drafting, reviews, negotiations, escalations, and potential vendor involvement.
00009	Train	Procure-to-Pay	A synthetic Procure-to-Pay process simulating requisition, vendor selection, purchase order creation, fulfillment, goods receipt, quality assurance, invoicing, and payment, incorporating complex control-flow, temporal, resource, and cost dynamics.
00010	Train	Risk Identification to Mitigation	A synthetic process simulating risk identification, assessment, root cause analysis, mitigation planning and execution, verification, approval, and case closure, with incorporated control-flow patterns, temporal dynamics, resource management, and cost calculations.
00011	Train	Requisition-to-Receipt	A synthetic process simulating the procurement cycle from requisition creation through purchase order issuance, vendor confirmation, goods receipt and quality checks, invoice matching and validation, to payment and closure, incorporating various control-flow, temporal, resource, and cost patterns.
00012	Test	Campaign-to-Lead	A synthetic process simulating marketing campaigns from initiation through audience targeting, asset creation, execution, monitoring, lead scoring, and qualification, with integrated control-flow variations, temporal dynamics, resource allocations, and cost considerations.
00013	Test	Close-to-Report	A synthetic event log simulating the financial close-to-report process, including period opening, data collection, reconciliations, adjustments, reviews, approvals, consolidations, and period closing, with various control-flow, temporal, resource, and cost dynamics.

control-flow motifs and temporal regimes. They are described in Table 1.

Thirteen logs, labeled 00001-00013, were created. Logs 00001-00011 train the model and tune hyperparameters, mixing patterns for generic prefix representations and control-flow/time-target links. Logs 00012-00013 are held-out for testing, fully separated from training/validation to assess cross-log transfer beyond memorization.

While the simulator can inject a wide range of motifs, the overall corpus is modest in size: only eleven training logs and two held-out test logs are used. This limited number of logs bounds the variety of behaviors, naming conventions, and timestamp regimes the model can internalize during training, and it also increases the variance of any single held-out estimate. For this reason, the results in Section VI should be read primarily as controlled evidence about trends (e.g., dependence on context size K, the effect of episode construction, and architectural ablations) rather than as a definitive characterization of performance across all real-world processes. Expanding the pretraining set with more heterogeneous logs, and especially additional real logs, is expected to improve coverage and reduce sensitivity to simulator-specific regularities.

Each log simulates executions from a configurable process model: a base control-flow skeleton (sequences, choices, loops, parallels) plus explicit patterns like rework, optional skips, rare detours, synchronizing joins, or early terminations. Variations in pattern frequency and interaction emphasize process aspects while maintaining structural comparability.

Timestamps and temporal quantities are generated alongside activities: inter-event times come from log-specific distributions, with additional effects like bursts, waits, or time-dependent routing. Noise (timestamp perturbations, activity insertions/deletions, resource scrambles) adds realism and reduces reliance on brittle artifacts, though it does not capture all sources of noise and heterogeneity found in operational systems.

Training logs cover broad patterns and temporal regimes for diverse pretraining exposure. Test logs differ in pattern combinations and frequencies (e.g., branch ratios, rework variants, and time scales), probing in-context adaptation via small supports without parameter updates, within the limits of the available held-out set.

B. EVALUATION BASELINES AND CONTEXT SIZE

We compare our foundation model to two standard `scikit-learn` baselines trained on the same (small) context available at test time. Throughout Section VI, "`sklearn` baseline" refers specifically to:

- **Next-activity prediction (classification):** multinomial logistic regression implemented by `sklearn.linear_model.LogisticRegression`, trained on support pairs (y_i, a_i) where y_i is a prefix embedding and a_i is its next-activity label.
- **Remaining-time prediction (regression):** ridge regression implemented by `sklearn.linear_model.Ridge`, trained on support pairs (y_i, t_i) where y_i is a prefix embedding and t_i is the (optionally transformed) remaining time.

We ensure a controlled comparison: for both tasks, the feature vectors for the baselines are the same prefix embeddings used by our in-context heads, i.e., the fixed-dimensional vectors $y(\tau_{\leq m})$ extracted from the target log by the encoder (with no additional learning on the target log beyond the baseline fit). When remaining-time targets are evaluated on a transformed scale (cf. ψ in Section III-C), the ridge baseline

is trained on that same transformed target and is mapped back via ψ^{-1} for reporting MAE and R^2.

Two modes drive evaluation: episodic (few-shot tasks) and retrieval (per-query neighbors), both using a small context of labeled examples. The interpretation of K depends on the mode and task, and the baselines follow exactly the same context definition as the foundation model.

In episodic mode, we simulate balanced few-shot tasks. For next-activity prediction, episodes are constructed as N-way, K-shot problems:

- K support prefixes per activity (total of $N \times K$ support points in the target embedding space),
- disjoint query prefixes for accuracy evaluation.

Our model predicts in-context via prototypes; the `sklearn` baseline fits a multinomial logistic-regression classifier on the same $N \times K$ support points and predicts on the query embeddings. For remaining time, K denotes the total number of support prefixes used for kernel regression (and analogously for ridge regression).

In retrieval mode, for each query embedding y' we retrieve K nearest neighbors from the target log to form its support set S (masking prefixes from the same case to prevent leakage). Our model predicts from S via the nonparametric heads; the `sklearn` baselines fit, for that query, logistic regression or ridge regression on the retrieved neighbor points (y_i, a_i) or (y_i, t_i) and then predict for y'.

K summary:

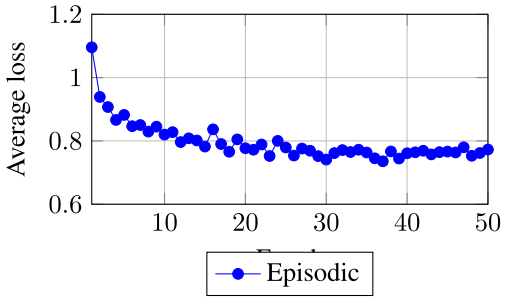
- Episodic classification: K = examples/activity (total $N \times K$ support points).
- Episodic regression: K = total support points.
- Retrieval (both tasks): K = neighbors/query.

All Section VI comparisons align K across model and baselines by construction: in every setting, the baselines are trained only on the same target-log support points that define the in-context episode/neighborhood for our method, and are evaluated on the same disjoint queries.

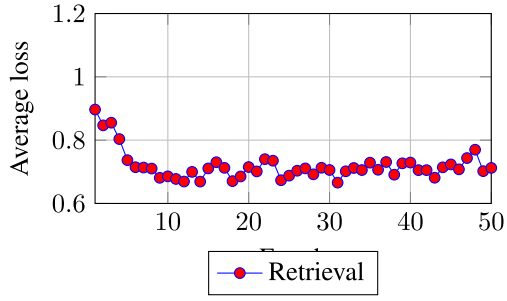
C. TRAINING SCHEDULE AND EPISODES PER EPOCH

Table labels like "Shuffle-yes + Episodic (50 ep.)" indicate the number of epochs the model is trained for. Unlike standard gradient steps on isolated events, every optimization uses a small episode, a few-shot task with a support set and disjoint queries, as in Section VI-B. Training thus repeatedly exposes the model to these tasks, conditioning predictions on the support to mimic test-time adaptation.

One epoch processes a fixed budget of freshly sampled episodes: at each step, select a training log, construct a support set and query group from its prefixes, route the episode to a single expert, compute losses, and update. This budget (`episodes_per_epoch`) is constant across experiments, so "10 epochs" means 10 passes over the same volume of new episodes, while "50 epochs" exposes the model to 50× more distinct tasks than "1 epoch". Total training scales linearly with epochs.



(a) Episodic schedule: balanced N-way, K-shot tasks sampled without favoring near neighbors.



(b) Retrieval-driven schedule: support formed by k nearest neighbors in the embedding space (same-case examples masked).

FIGURE 12. Average training loss per epoch under two episode-construction strategies with label shuffling enabled (`shuffle_yes`). The loss is the batch average of classification and regression losses; compare trends across panels rather than absolute values.

Within epochs, larger logs contribute more episodes (due to more prefixes), but all logs are revisited frequently, with cases appearing variably as supports or queries to foster generalization over memorization of fixed splits. This ensures diverse exposure to control-flow and temporal patterns across the heterogeneous training logs.

Ablations vary epochs to isolate exposure effects: increasing epochs does not alter episode structure or support/query building, it only scales the number of practiced tasks. Thus, interpret "50 epochs" as "many more distinct few-shot tasks seen during training" versus "1 epoch" as "single pass over the fixed budget", easing row comparisons in ablation tables like Table 3.

D. READING GUIDE AND METRICS

This subsection orients readers on interpreting the assessment's figures and tables. Figure 12 plots average training loss (joint classification/regression) across epochs for episodic and retrieval strategies, highlighting stability and gains from extra epochs, focus on trends, not absolute values, due to episode variability.

Performance tables report next-activity accuracy and remaining-time MAE/R^2. Higher accuracy/R^2 is better; lower MAE is better. Negative R^2 means a naïve mean baseline outperforms the model on that slice (common in few-shot regimes, signaling difficulty). Columns split Tr (training)

TABLE 2. Training configurations evaluated in Section VI. The first row ("Shuffle-yes + Episodic") is used as the default throughout the text.

Configuration	Epochs	Label Shuffle	Training Strategy	MoE Experts	d_{model}	Attn Heads	Layers	Embedding Strategy
Shuffle-yes + Episodic	50	yes	episodic	4	256	8	6	learned
Shuffle-yes + Retrieval	50	yes	retrieval	4	256	8	6	learned
Shuffle-no + Episodic	50	no	episodic	4	256	8	6	learned
Shuffle-no + Retrieval	50	no	retrieval	4	256	8	6	learned
MoE = 2	50	yes	episodic	2	256	8	6	learned
MoE = 8	50	yes	episodic	8	256	8	6	learned
Pretrained embeddings	50	yes	episodic	4	256	8	6	pretrained
d_{model} = 128	50	yes	episodic	4	128	8	6	learned
d_{model} = 512	50	yes	episodic	4	512	8	6	learned
Attn Heads = 4	50	yes	episodic	4	256	4	6	learned
Attn Heads = 16	50	yes	episodic	4	256	16	6	learned
Layers = 4	50	yes	episodic	4	256	8	4	learned
Layers = 8	50	yes	episodic	4	256	8	8	learned

TABLE 3. Episodic (meta-learning) results by configuration. Absolute Train (Tr) and Test (Te) values for Accuracy, MAE, and R^2 at $K \in 1, 5, 10, 20$. Row 1 is the default (Shuffle-yes + Episodic, 50 epochs). Higher is better for Accuracy and R^2; lower is better for MAE.

	Accuracy								MAE								R^2							
	K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Episodic (50 epochs, default)	0.781	0.605	0.831	0.711	0.842	0.718	0.851	0.732	86.3	124.8	63.0	95.6	61.7	90.6	60.1	93.1	-1.064	-1.506	-0.128	-0.224	-0.016	-0.169	0.023	-0.187
Shuffle-yes + Episodic (30 epochs)	0.755	0.621	0.811	0.713	0.826	0.730	0.848	0.737	77.3	115.9	65.7	89.5	66.5	94.2	61.6	82.5	-1.063	-2.737	-0.082	-0.216	-0.013	-0.184	0.031	-0.152
Shuffle-yes + Episodic (10 epochs)	0.727	0.589	0.796	0.696	0.804	0.730	0.806	0.742	92.7	129.2	60.9	101.3	48.7	92.3	58.8	88.5	-0.964	-0.770	-0.118	-0.256	-0.032	-0.168	0.043	-0.116
Shuffle-no + Episodic	0.773	0.600	0.830	0.700	0.841	0.711	0.834	0.721	93.6	134.7	66.2	96.5	65.6	104.0	56.4	90.6	-0.947	-2.074	-0.084	-0.218	0.014	-0.221	-0.007	-0.146
MoE = 2	0.791	0.600	0.854	0.705	0.856	0.719	0.853	0.718	83.5	126.4	60.1	101.3	58.3	92.3	54.7	92.0	-1.062	-0.762	-0.096	-0.185	0.000	-0.247	-0.006	-0.138
MoE = 8	0.760	0.616	0.813	0.706	0.822	0.742	0.826	0.764	95.2	143.3	78.5	100.8	63.1	92.6	64.6	88.8	-0.824	-0.822	-0.120	-0.221	-0.042	-0.151	0.057	-0.139
Pretrained embeddings	0.715	0.568	0.769	0.660	0.778	0.669	0.792	0.676	84.2	123.3	65.0	96.4	61.8	93.2	64.2	100.4	-1.061	-1.476	-0.132	-0.141	-0.022	-0.150	-0.023	-0.191
$d_{model} = 128$	0.771	0.600	0.822	0.677	0.837	0.711	0.835	0.721	98.3	132.2	63.7	98.0	60.7	87.6	60.0	92.5	-0.979	-1.145	-0.119	-0.185	-0.070	-0.116	-0.001	-0.146
$d_{model} = 512$	0.792	0.597	0.834	0.713	0.848	0.727	0.850	0.743	82.5	125.0	70.5	107.3	51.3	101.1	57.4	91.3	-1.073	-0.664	-0.169	-0.260	-0.002	-0.210	-0.036	-0.133
Attn Heads = 4	0.781	0.621	0.833	0.708	0.835	0.703	0.844	0.714	89.8	132.2	62.0	91.1	56.6	92.2	58.0	86.1	-1.085	-0.843	-0.082	-0.180	-0.018	-0.166	-0.027	-0.172
Attn Heads = 16	0.781	0.597	0.828	0.698	0.845	0.728	0.841	0.720	99.1	142.2	60.0	101.7	60.5	95.6	60.4	88.9	-0.979	-1.538	-0.112	-0.163	-0.006	-0.160	-0.001	-0.115
Layers = 4	0.773	0.604	0.846	0.697	0.828	0.722	0.841	0.724	92.5	122.5	71.0	96.9	62.1	84.0	58.4	92.9	-1.069	-0.855	-0.133	-0.196	0.002	-0.140	-0.004	-0.133
Layers = 8	0.781	0.610	0.827	0.709	0.841	0.725	0.840	0.731	94.6	123.6	65.2	88.3	60.0	99.5	54.1	93.6	-1.000	-1.018	-0.109	-0.188	-0.002	-0.265	0.048	-0.143

vs. Te (test) for generalization gaps; K values track support growth from K=1 to 20 (per activity/query).

Table 2 maps configuration names to settings: epochs, shuffling, strategy (episodic vs. retrieval), MoE count, d_{model}, heads/layers, and embedding type (learned/pretrained). The default is row 1 (Shuffle-yes + Episodic, 50 epochs, MoE= 4, d_{model}=256, 8 heads, 6 layers, learned).

Table 3 gives absolute episodic results (plus epoch ablations). Table 4 shows deltas vs. the `scikit-learn` baselines defined in Section VI-B (ours minus `sklearn`; i.e., multinomial `LogisticRegression` for Accuracy and `Ridge` for MAE/R^2). Table 5 isolates single changes vs. default. Table 6/Table 7 detail K-scaling: cumulative vs. stepwise deltas, spotlighting diminishing returns.

Retrieval tables (Table 8, Table 9, Table 10, Table 11, Table 12) mirror this structure. Episodic/retrieval losses/metrics are intra-block only, not cross-comparable.

E. TRAINING DYNAMICS

Figure 12 summarizes average training loss evolution across epochs for the two episode-construction strategies: episodic (balanced N-way, K-shot tasks; top panel) and retrieval-driven (nearest-neighbor supports; bottom panel). The y-axis averages classification and regression losses over mini-batches, reflecting joint task behavior. Absolute levels vary due to episode differences (classes, balance, composition), so the analysis focuses on trends: downward movement, descent speed, and plateau stability.

Both exhibit fast initial descent in early epochs, showing the encoder and heads rapidly learn from small supports. Retrieval starts lower and drops faster (first 10 epochs), as neighbor-focused episodes ease early fitting; episodic descends more gradually but with smoother mid-training behavior.

Post-initial phase, curves reach a slow-improvement plateau with minor oscillations, which is expected from log/task mixing and label shuffling (`shuffle_yes`) that injects noise to curb global label memorization and boost support reliance. This helps interpret table trends: for example, stopping at 30 epochs matches or slightly improves some test metrics relative to 50 epochs in the default episodic setup (Table 3), suggesting that validation-based early stopping can be beneficial even when training loss continues to decline.

Retrieval minima arrive sooner and lower, yet this does not automatically translate to higher episodic test accuracy, because retrieval loss emphasizes local interpolation around nearest neighbors, while episodic tests require uniform separation across a balanced label set. Thus, retrieval can look stronger in loss curves while episodic can yield higher

classification accuracy under the episodic protocol (Table 3); in this setting, a lower retrieval loss should be interpreted as improved local fit rather than as a guarantee of better cross-log transfer.

Train-test gaps stay moderate and relatively stable across epochs, suggesting that the MoE aggregation provides some robustness, although these gaps are still influenced by the limited number of training and test logs. Larger capacity or mixture width can speed early descent but may amplify late oscillations, echoing some ablation behaviors (Table 5) and reinforcing the role of validation metrics over loss alone when selecting checkpoints.

Curve shapes also mirror the observed scaling with context size K (see Table 6 and Table 7): the rapid early drops in loss correspond to the large performance gains from $K = 1$ to $K = 5$, while the later plateaus reflect smaller, diminishing returns between $K = 10$ and $K = 20$. In summary, within this experimental corpus, training for roughly 20-30 epochs with validation-based early stopping often balances stability and generalization, but the optimal stopping point and the relative benefit of episodic versus retrieval schedules may change when training on a larger and more diverse set of logs.

F. EPISODIC (META-LEARNING) RESULTS

In the episodic setting the model receives small, balanced support sets and must adapt in context without parameter updates. Absolute metrics for the main configurations are shown in Table 3; differences against the `scikit-learn` baseline appear in Table 4; and sensitivity to the default configuration, as well as the effect of increasing the number of shots K, are summarized in Table 5, Table 6, and Table 7. Because episodic test results are computed on two held-out synthetic logs, absolute values should be interpreted with some caution; we therefore emphasize consistent trends across K and across configurations.

The overall pattern is consistent within this setup: more shots tend to lift next-activity accuracy and reduce remaining-time error. In the default configuration (Shuffle-yes + Episodic, Table 2), test accuracy rises from 0.605 at K=1 to 0.711 at K=5, 0.718 at K=10, and 0.732 at K=20 (Table 3). Test MAE drops from 124.8 at K=1 to 95.6 at K=5 and 90.6 at K=10, rebounding slightly to 93.1 at K=20; R^2 improves but stays negative (from -1.506 to -0.224, -0.169, and -0.187), indicating that errors shrink with more context but that explaining variance remains challenging under strict few-shot transfer, especially when temporal scales vary within the sampled supports.

The K-effect tables make trends explicit. Relative to K=1, the default gains $+0.106$, $+0.113$, and $+0.127$ test accuracy at K=5, 10, 20; MAE improves by -29.255, -34.221, and -31.713; and R^2 by $+1.282$, $+1.337$, and $+1.319$ (Table 6). Stepwise deltas show that most benefits accrue between K=1 and 5; gains slow down from K=5 to 10, and flatten or slightly worsen MAE from 10 to 20 (Table 7), which is consistent with nonparametric predictors where additional support can dilute locality if the added examples are less relevant.

Training duration regularizes: in this corpus, stopping earlier can sometimes improve test metrics despite ongoing loss drops. At 30 epochs, K=20 accuracy reaches 0.737 and MAE 82.5; at 10 epochs, 0.742 and 88.5 (Table 3; see deltas in Table 5), suggesting that the best generalization may occur before full convergence, though the exact optimum may vary with a larger or more heterogeneous training set.

Architecture ablations point toward moderate capacity: expanding MoE to eight experts increases K=20 accuracy to 0.764 and yields MAE 88.8 (vs. default 0.732/93.1); d_{model}=512 yields a smaller improvement (0.743/91.3); and d_{model}=128 maintains classification accuracy while limiting regression. Varying the number of heads or depth changes results modestly, and label shuffling is generally helpful for classification with mixed regression effects. Pretrained embeddings reduce accuracy in this setup (0.676 at K=20), indicating that, under our training protocol (including label shuffling), structural and temporal cues can dominate lexical semantics; this conclusion should be revisited when training on more varied real logs where naming conventions may carry stronger signal.

Against the `scikit-learn` baselines, the model is generally competitive on classification and often favorable on regression for $K \geq 5$: accuracy gaps narrow with context, and MAE improvements are common across configurations (Table 4). At the same time, given the limited number of held-out logs, these comparisons should be seen as indicative rather than conclusive; broader benchmarking would better quantify when the nonparametric in-context heads consistently dominate per-log fitted baselines.

In practical terms, a handful of shots per activity can reach near-peak next-activity performance on this episodic benchmark, and the kernel-style head can substantially reduce remaining-time error even when R^2 remains difficult in strict few-shot settings. In these experiments, early stopping (e.g., 20-30 epochs), moderate mixture widths (e.g., 4-8 experts), and structure-focused embeddings tend to work well, but these preferences may shift as the training corpus grows in size and diversity. Overall, the results support the qualitative insight that a small number of informative local examples can drive most of the gains in meta-learning-style adaptation, while also highlighting that the current evidence is bounded by the limited set of available training and test logs.

G. RETRIEVAL-AUGMENTED RESULTS

This subsection reports results for the retrieval-augmented setting, where the support set for each query is assembled by nearest-neighbor search within the target log (with same-case masking) rather than by balanced class sampling. Absolute metrics for the two retrieval configurations are shown in Table 8; differences against the `scikit-learn` baseline appear in Table 9; and sensitivity to the default configuration, as well as the effect of increasing the number of shots K, are

TABLE 4. Episodic: difference to scikit-learn baseline (ours − sklearn). Positive deltas are better for Accuracy and R^2; negative deltas are better for MAE. Cells shaded green mark improvements; cells shaded red highlight large losses in Accuracy/R^2 (worse by > 0.10).

	Accuracy								MAE								R^2							
	K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Episodic (50 ep.)	-0.002	-0.002	-0.011	-0.042	-0.043	-0.073	-0.046	-0.093	+11.573	-8.534	-2.359	-14.995	-33.883	-27.660	-24.882	-64.586	-0.172	-0.096	+0.301	+0.123	+1.357	+0.743	+1.108	+11.761
Shuffle-yes + Episodic (30 ep.)	-0.016	+0.010	-0.036	-0.031	-0.056	-0.059	-0.043	-0.095	-7.909	-18.192	-16.999	-18.795	-26.863	-10.403	-16.594	-59.613	-0.109	-1.735	+0.553	+0.171	+17.857	+0.355	+0.939	+3.724
Shuffle-yes + Episodic (10 ep.)	+0.003	-0.021	-0.029	-0.051	-0.055	-0.052	-0.079	-0.095	+9.674	+7.496	-17.572	+7.165	-41.547	-26.288	-19.048	-22.317	-0.052	+1.590	+2.050	-0.068	+3.106	+1.711	+1.936	+1.130
Shuffle-no + Episodic	+0.002	-0.032	-0.014	-0.046	-0.031	-0.099	-0.053	-0.114	+13.272	-2.235	-10.478	-24.431	-21.865	-56.642	-21.957	-85.014	-0.099	-1.094	+0.587	+0.264	+2.703	+10.599	+3.179	+12.213
MoE = 2	-0.010	-0.019	-0.005	-0.044	-0.018	-0.061	-0.046	-0.106	-12.567	-9.256	-29.197	-86.193	-86.469	-79.294	-117.763	-67.773	-0.090	-0.126	+3.589	+45.241	+74.505	+5.209	+21.136	+16.198
MoE = 8	-0.002	+0.006	-0.026	-0.059	-0.054	-0.062	-0.065	-0.090	+8.674	+29.209	+5.441	+2.189	-5.938	-5.208	+2.138	-20.701	+0.147	-0.188	+0.025	+0.045	+1.071	+0.322	+0.165	+0.278
Pretrained embeddings	-0.006	+0.023	-0.036	-0.029	-0.038	-0.086	-0.043	-0.123	-12.226	-13.534	-17.423	-16.938	-17.238	-149.949	-11.649	-650.629	-0.111	+0.195	+1.550	+0.675	+2.498	+48.756	+2.559	+4426.000
$d_{model} = 128$	-0.010	+0.003	-0.025	-0.071	-0.037	-0.084	-0.061	-0.102	+16.839	+5.272	-4.738	-7.741	-5.185	-85.123	-0.665	-12.642	+0.002	-0.593	+0.715	+0.304	+0.629	+36.086	+0.175	+1.489
$d_{model} = 512$	+0.000	-0.010	-0.031	-0.025	-0.027	-0.046	-0.045	-0.070	-6.486	-4.861	-2.776	-3.317	-54.099	-28.556	-72.032	-50.399	-0.032	+0.344	+1.206	+0.391	+8.698	+2.306	+51.296	+10.648
Attn Heads = 4	-0.005	+0.023	-0.006	-0.045	-0.036	-0.080	-0.044	-0.112	-1.911	-1.721	-14.826	-21.940	-13.335	-35.561	-35.660	-37.936	-0.188	+0.964	+0.462	+1.062	+5.217	+4.463	+4.064	+2.367
Attn Heads = 16	-0.007	-0.015	-0.023	-0.071	-0.038	-0.055	-0.046	-0.103	+6.231	+15.594	-16.328	-16.626	-19.197	-36.341	-15.912	-52.354	+0.121	-0.406	+0.676	+0.756	+2.061	+2.387	+2.668	+5.188
Layers = 4	-0.016	-0.013	-0.003	-0.041	-0.039	-0.064	-0.051	-0.100	+2.461	-11.283	+2.401	-16.598	-15.411	-41.685	-28.969	-33.098	+0.101	+0.056	+0.336	+3.375	+0.940	+2.576	+5.267	+12.455
Layers = 8	+0.010	+0.007	-0.028	-0.033	-0.046	-0.067	-0.053	-0.094	+2.711	-1.117	-0.316	-10.939	-21.657	-66.030	-16.657	-33.988	-0.099	-0.364	+0.999	+0.397	+4.236	+18.686	+1.177	+2.033

TABLE 5. Episodic: difference to the default configuration (ours − default; default = Shuffle-yes + Episodic, 50 epochs). Green denotes improvement (higher Accuracy/R^2 or lower MAE); red highlights substantial degradations as defined in the text.

	Accuracy								MAE								R^2							
	K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Episodic (50 epochs, default)	0.781	0.605	0.831	0.711	0.842	0.718	0.851	0.732	86.3	124.8	63.0	95.6	61.7	90.6	60.1	93.1	-1.064	-1.506	-0.128	-0.224	-0.016	-0.169	0.023	-0.187
Shuffle-yes + Episodic (30 epochs)	-0.026	+0.016	-0.020	+0.002	-0.016	+0.012	-0.003	+0.005	-9.019	-8.951	+2.704	-6.115	+4.824	+3.573	+1.555	-10.626	+0.001	-1.231	+0.045	+0.007	+0.003	-0.015	+0.008	+0.036
Shuffle-yes + Episodic (10 epochs)	-0.054	-0.016	-0.034	-0.015	-0.038	+0.012	-0.044	+0.010	+6.355	+4.391	-2.103	+5.756	-12.934	+1.697	-1.235	-4.671	+0.100	+0.736	+0.010	-0.032	-0.016	+0.002	+0.020	+0.072
Shuffle-no + Episodic	-0.008	-0.005	-0.001	-0.011	-0.001	-0.007	-0.017	-0.010	+7.254	+9.868	+3.196	+0.939	+3.917	+13.355	-3.705	-2.556	+0.117	-0.568	+0.044	+0.006	+0.030	-0.052	-0.030	+0.042
MoE = 2	+0.009	-0.005	+0.023	-0.006	+0.014	+0.001	+0.002	-0.014	-2.882	+1.560	-2.844	+5.743	-3.316	+1.703	-5.358	-1.110	+0.002	+0.744	+0.032	+0.039	+0.016	-0.078	-0.029	+0.049
MoE = 8	-0.021	+0.011	-0.018	-0.005	-0.020	+0.025	-0.025	+0.033	+8.891	+18.455	+15.572	+5.240	+1.484	+1.934	+4.513	-4.283	+0.240	+0.684	+0.007	+0.003	-0.026	+0.018	+0.034	+0.048
Pretrained embeddings	-0.066	-0.036	-0.062	-0.051	-0.064	-0.049	-0.059	-0.055	-2.196	-1.548	+2.064	+0.848	+0.182	+2.632	+4.170	+7.298	+0.003	+0.030	-0.004	+0.082	-0.006	+0.019	-0.045	-0.004
$d_{model} = 128$	-0.010	-0.005	-0.009	-0.034	-0.005	-0.006	-0.016	-0.011	+11.968	+7.396	+0.684	+2.453	-0.967	-2.993	-0.044	-0.584	+0.085	+0.362	+0.008	+0.039	-0.055	+0.053	-0.024	+0.042
$d_{model} = 512$	+0.011	-0.007	+0.003	+0.002	+0.006	+0.009	-0.001	+0.011	-3.823	+0.159	+7.570	+11.703	-10.330	+10.517	-2.725	-1.835	-0.009	+0.842	-0.041	-0.036	+0.014	-0.041	-0.059	+0.054
Attn Heads = 4	+0.000	+0.017	+0.002	-0.004	-0.007	-0.015	-0.007	-0.018	+3.438	+7.332	-0.974	-4.513	-5.010	+1.557	-2.067	-6.985	-0.021	+0.664	+0.046	+0.043	-0.003	+0.003	-0.050	+0.016
Attn Heads = 16	+0.000	-0.008	-0.003	-0.013	+0.003	+0.010	-0.010	-0.012	+12.736	+17.344	-2.978	+6.131	-1.149	+4.963	+0.293	-4.227	+0.085	-0.032	+0.015	+0.061	+0.009	+0.009	-0.024	+0.073
Layers = 4	-0.008	-0.000	+0.016	-0.014	-0.014	+0.005	-0.009	-0.008	+6.126	-2.302	+8.045	+1.294	+0.472	-6.636	-1.630	-0.198	-0.005	+0.651	-0.006	+0.028	+0.018	+0.029	-0.027	+0.054
Layers = 8	-0.001	+0.006	-0.004	-0.002	-0.001	+0.008	-0.011	-0.001	+8.247	-1.204	+2.256	-7.292	-1.666	+8.887	-5.997	+0.442	+0.064	+0.489	+0.019	+0.036	+0.014	-0.096	+0.025	+0.044

TABLE 6. Episodic: effect of increasing K within each configuration. Columns for K=1 report absolute Tr/Te values; columns for $K \in 5, 10, 20$ report deltas relative to K=1 (same row). Green indicates improvement (higher Accuracy/R^2 or lower MAE); red marks notable deterioration.

	Accuracy								MAE								R^2							
	K=1		K=5−K=1		K=10−K=1		K=20−K=1		K=1		K=5−K=1		K=10−K=1		K=20−K=1		K=1		K=5−K=1		K=10−K=1		K=20−K=1	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Episodic (50 epochs, default)	0.781	0.605	+0.049	+0.106	+0.061	+0.113	+0.069	+0.127	86.3	124.8	-23.375	-29.255	-24.690	-34.221	-26.273	-31.713	-1.064	-1.506	+0.936	+1.282	+1.048	+1.337	+1.087	+1.319
Shuffle-yes + Episodic (10 epochs)	0.727	0.589	+0.069	+0.107	+0.077	+0.141	+0.079	+0.153	92.7	129.2	-31.834	-27.890	-43.980	-36.915	-33.863	-40.775	-0.964	-0.770	+0.846	+0.514	+0.932	+0.602	+1.006	+0.654
Shuffle-yes + Episodic (30 epochs)	0.755	0.621	+0.056	+0.092	+0.071	+0.109	+0.093	+0.116	77.3	115.9	-11.653	-26.419	-10.847	-21.697	-15.699	-33.388	-1.063	-2.737	+0.981	+2.520	+1.051	+2.553	+1.094	+2.585
Shuffle-no + Episodic	0.773	0.600	+0.057	+0.100	+0.068	+0.111	+0.061	+0.121	93.6	134.7	-27.433	-38.185	-28.027	-30.735	-37.233	-44.136	-0.947	-2.074	+0.863	+1.856	+0.961	+1.853	+0.940	+1.928
MoE = 2	0.791	0.600	+0.063	+0.105	+0.066	+0.119	+0.062	+0.118	83.5	126.4	-23.337	-25.073	-25.123	-34.079	-28.749	-34.383	-1.062	-0.762	+0.966	+0.577	+1.062	+0.515	+1.056	+0.624
MoE = 8	0.760	0.616	+0.053	+0.090	+0.062	+0.126	+0.066	+0.148	95.2	143.3	-16.695	-42.471	-32.098	-50.743	-30.651	-54.452	-0.824	-0.822	+0.704	+0.601	+0.783	+0.671	+0.881	+0.683
Pretrained embeddings	0.715	0.568	+0.054	+0.092	+0.063	+0.100	+0.077	+0.108	84.2	123.3	-19.115	-26.859	-22.312	-30.041	-19.907	-22.867	-1.061	-1.476	+0.929	+1.334	+1.039	+1.326	+1.038	+1.285
$d_{model} = 128$	0.771	0.600	+0.051	+0.078	+0.066	+0.111	+0.064	+0.121	98.3	132.2	-34.660	-34.199	-37.625	-44.610	-38.286	-39.693	-0.979	-1.145	+0.859	+0.960	+0.908	+1.029	+0.978	+0.999
$d_{model} = 512$	0.792	0.597	+0.041	+0.115	+0.056	+0.130	+0.058	+0.146	82.5	125.0	-11.982	-17.711	-31.197	-23.863	-25.175	-33.707	-1.073	-0.664	+0.904	+0.405	+1.071	+0.454	+1.037	+0.531
Attn Heads = 4	0.781	0.621	+0.051	+0.086	+0.054	+0.082	+0.062	+0.093	89.8	132.2	-27.787	-41.101	-33.138	-39.997	-31.779	-46.030	-1.085	-0.843	+1.003	+0.662	+1.067	+0.676	+1.058	+0.671
Attn Heads = 16	0.781	0.597	+0.046	+0.101	+0.064	+0.131	+0.059	+0.123	99.1	142.2	-39.090	-40.469	-38.575	-46.603	-38.716	-53.284	-0.979	-1.538	+0.866	+1.375	+0.972	+1.378	+0.977	+1.423
Layers = 4	0.773	0.604	+0.074	+0.092	+0.055	+0.118	+0.069	+0.120	92.5	122.5	-21.456	-25.660	-30.343	-38.556	-34.030	-29.609	-1.069	-0.855	+0.935	+0.659	+1.071	+0.715	+1.065	+0.722
Layers = 8	0.781	0.610	+0.046	+0.098	+0.061	+0.115	+0.059	+0.120	94.6	123.6	-29.366	-35.344	-34.603	-24.130	-40.517	-30.067	-1.000	-1.018	+0.890	+0.829	+0.998	+0.752	+1.048	+0.874

TABLE 7. Episodic: stepwise deltas as K increases (5−1), (10−5), (20−10), for Accuracy, MAE, and R^2 (Tr/Te). Green indicates improvement; red marks notable deterioration.

	Accuracy								MAE								R^2							
	K=1		K=5−K=1		K=10−K=5		K=20−K=10		K=1		K=5−K=1		K=10−K=5		K=20−K=10		K=1		K=5−K=1		K=10−K=5		K=20−K=10	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Episodic (50 epochs, default)	0.781	0.605	+0.049	+0.106	+0.011	+0.007	+0.009	+0.014	86.3	124.8	-23.375	-29.255	-1.314	-4.966	-1.583	+2.509	-1.064	-1.506	+0.936	+1.282	+0.112	+0.055	+0.039	-0.018
Shuffle-yes + Episodic (10 epochs)	0.727	0.589	+0.069	+0.107	+0.007	+0.034	+0.003	+0.012	92.7	129.2	-31.834	-27.890	-12.145	-9.025	+10.117	-3.860	-0.964	-0.770	+0.846	+0.514	+0.086	+0.088	+0.074	+0.052
Shuffle-yes + Episodic (30 epochs)	0.755	0.621	+0.056	+0.092	+0.015	+0.017	+0.022	+0.007	77.3	115.9	-11.653	-26.419	+0.806	+4.723	-4.852	-11.691	-1.063	-2.737	+0.981	+2.520	+0.070	+0.033	+0.043	+0.032
Shuffle-no + Episodic	0.773	0.600	+0.057	+0.100	+0.011	+0.011	-0.007	+0.010	93.6	134.7	-27.433	-38.185	-0.594	+7.450	-9.205	-13.402	-0.947	-2.074	+0.863	+1.856	+0.098	-0.003	-0.021	+0.075
MoE = 2	0.791	0.600	+0.063	+0.105	+0.003	+0.015	-0.004	-0.001	83.5	126.4	-23.337	-25.073	-1.786	-9.006	-3.625	-0.304	-1.062	-0.762	+0.966	+0.577	+0.096	-0.062	-0.006	+0.109
MoE = 8	0.760	0.616	+0.053	+0.090	+0.009	+0.036	+0.004	+0.022	95.2	143.3	-16.695	-42.471	-15.403	-8.271	+1.446	-3.709	-0.824	-0.822	+0.704	+0.601	+0.079	+0.070	+0.098	+0.012
Pretrained embeddings	0.715	0.568	+0.054	+0.092	+0.009	+0.009	+0.014	+0.008	84.2	123.3	-19.115	-26.859	-3.197	-3.182	+2.404	+7.174	-1.061	-1.476	+0.929	+1.334	+0.110	-0.009	-0.001	-0.041
$d_{model} = 128$	0.771	0.600	+0.051	+0.078	+0.015	+0.034	-0.002	+0.009	98.3	132.2	-34.660	-34.199	-2.965	-10.412	-0.660	+4.918	-0.979	-1.145	+0.859	+0.960	+0.049	+0.069	+0.069	-0.030
$d_{model} = 512$	0.792	0.597	+0.041	+0.115	+0.014	+0.014	+0.002	+0.016	82.5	125.0	-11.982	-17.711	-19.215	-6.152	+6.022	-9.844	-1.073	-0.664	+0.904	+0.405	+0.167	+0.049	-0.034	+0.077
Attn Heads = 4	0.781	0.621	+0.051	+0.086	+0.002	-0.004	+0.009	+0.011	89.8	132.2	-27.787	-41.101	-5.351	+1.104	+1.360	-6.033	-1.085	-0.843	+1.003	+0.662	+0.064	+0.014	-0.009	-0.005
Attn Heads = 16	0.781	0.597	+0.046	+0.101	+0.017	+0.029	-0.004	-0.008	99.1	142.2	-39.090	-40.469	+0.515	-6.134	-0.141	-6.681	-0.979	-1.538	+0.866	+1.375	+0.106	+0.003	+0.005	+0.046
Layers = 4	0.773	0.604	+0.074	+0.092	-0.019	+0.026	+0.014	+0.001	92.5	122.5	-21.456	-25.660	-8.887	-12.896	-3.686	+8.946	-1.069	-0.855	+0.935	+0.659	+0.136	+0.056	-0.006	+0.007
Layers = 8	0.781	0.610	+0.046	+0.098	+0.014	+0.017	-0.001	+0.005	94.6	123.6	-29.366	-35.344	-5.236	+11.214	-5.915	-5.937	-1.000	-1.018	+0.890	+0.829	+0.108	-0.077	+0.050	+0.122

summarized in Table 10, Table 11, and Table 12. As with episodic evaluation, these results are obtained on a limited number of held-out logs, so we emphasize robust trends across K and across shuffling variants.

Adding more in-context examples helps, but gains saturate sooner for classification than for regression. For the default Shuffle-yes + Retrieval configuration in Table 8, test accuracy increases from 0.405 at K=1 to 0.475 at K=10, with a decline at K=20. Remaining-time prediction benefits more steadily: test MAE decreases from 90.6 at K=1 to 68.1 at K=20, and R^2 moves from negative at K=1 to positive for $K \geq 5$, reaching 0.411 at K=20. The stepwise deltas in Table 12 confirm that the largest accuracy gain arrives between K=1 and K=5, while regression continues to benefit from larger neighborhoods, albeit with diminishing returns.

Label shuffling improves adaptation: compared to Shuffle-no + Retrieval in Table 8, shuffling yields higher test accuracy at every K and consistently lower MAE, with the gap most visible around K=5-10. Positive R^2 for $K \geq 5$ is also larger with shuffling, as reflected by the default-vs-variant deltas in Table 10. In this retrieval regime, shuffling appears to act as a form of local calibration that encourages reliance on neighborhood structure rather than on any fixed global label identity.

Compared to the episodic protocol in Section VI-F, classification accuracies are lower under retrieval, which is expected given the natural, imbalanced label distribution and long-tail activities versus balanced N-way sampling. However, regression is stronger: test R^2 becomes positive for moderate K, consistent with the idea that retrieving close prefixes provides log-specific temporal anchors. The degree to which this holds can depend on how well the embedding space aligns with the target log; with substantially noisier real logs or with strong covariate shifts, retrieval quality may become a primary bottleneck.

Against the `scikit-learn` baseline, the retrieval-augmented model is competitive on classification and favorable on remaining time in this evaluation. In Table 9, Shuffle-yes + Retrieval reduces test MAE for all K (negative deltas), while accuracy differences are small and vary with K. R^2 differences are mixed and small in magnitude, reinforcing that retrieval regression benefits are more consistently reflected in MAE than in variance-explained metrics under limited-context evaluation.

In practical terms, retrieval mode supports the model's deployment intuition: a compact set of relevant neighbors (e.g., K = 5-10) can yield clear improvements in remaining-time MAE and produce meaningful positive R^2 values on the evaluated logs. For next-activity prediction, gains emerge quickly from K = 1 to 5 and then tend to plateau or decline at K = 20, underscoring the value of targeted supports over large, potentially heterogeneous neighborhoods. These conclusions, however, remain conditional on the limited set of logs studied here; broader testing on additional and more diverse real logs is needed to establish how robust these retrieval behaviors are in operational settings.

H. CASE STUDY: WABO ENVIRONMENTAL PERMIT RECEIPT PHASE

To complement the synthetic logs used in Section VI-A, we evaluate the foundation model on a real-life process: the Receipt phase of an environmental permit application process (WABO) from the CoSeLoG project. We run `testing.py` in `meta_learning` mode on the pre-trained 4-expert mixture, using the default learned embedding strategy. For fairness, we reuse exactly the same prefix embeddings for both our in-context heads and the `scikit-learn` baselines.

Crucially, in this case study the `scikit-learn` baselines are trained only on the target-log support points (i.e., points in the target embedding space) used as in-context examples for the foundation model. Concretely, for each support size $K \in \{1, 2, 3, 4, 5, 6, 7, 8, 10, 15, 20, 40, 50, 75, 100, 200, 350, 500, 750\}$ we construct meta-testing episodes with a support set S_K and a disjoint query set Q_K, both sampled from prefixes of the WABO receipt log:

- **Next-activity (classification).** The support set provides labeled points (y_i, a_i), where $y_i = y(\tau_{\leq m})$ is the embedding of a WABO prefix and $a_i = a^\star(\tau_{\leq m})$ is its next-activity label. Our method uses S_K to form prototypes and predicts on the query embeddings. The baseline fits a multinomial logistic-regression classifier (`sklearn.linear_model.LogisticRegression`) on the same support points (y_i, a_i) (i.e., the episode's target-space training set) and is evaluated on the same query set.
- **Remaining time (regression).** The support set provides labeled points (y_i, t_i) with remaining-time targets (optionally on the transformed scale ψ). Our kernel regressor predicts from S_K; the baseline fits a ridge regressor (`sklearn.linear_model.Ridge`) on the same support points (y_i, t_i) and is evaluated on the same query embeddings (with ψ^{-1} applied if needed for MAE/R^2 reporting).

No prefixes outside these episode supports from the WABO receipt log are used to fit the baselines, and no synthetic training logs are used for baseline training. Moreover, the episodes for different values of K are sampled independently: the support set for K=10 is not a superset of the one for K=5, and both the composition of S_K and the associated query set Q_K change with K. Hence the curves in Figure 13 should be read as performance at a given support size rather than as a strictly nested learning curve.

Figure 13 summarizes these results. The left panel plots the classification accuracy as a function of the support size K. At K=1, the foundation model slightly outperforms the logistic baseline (0.52 vs. 0.51 accuracy), indicating that the in-context prototype head extracts more from a very small target-log context. As K increases, the logistic regression benefits from parametric fitting and eventually reaches higher asymptotic accuracy (around 0.90 at large K), whereas

TABLE 8. Retrieval-augmented results by configuration. Absolute Train (Tr) and Test (Te) values for Accuracy, MAE, and R^2 at $K \in 1, 5, 10, 20$, where K is the number of retrieved neighbors per query (same-case examples masked). Higher is better for Accuracy and R^2; lower is better for MAE.

	Accuracy								MAE								R^2							
	K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Retrieval	0.520	0.405	0.525	0.455	0.540	0.475	0.535	0.443	51.1	90.6	40.3	75.1	37.6	69.7	36.0	68.1	-0.011	-0.122	0.239	0.292	0.303	0.387	0.356	0.411
Shuffle-no + Retrieval	0.515	0.352	0.535	0.412	0.520	0.395	0.525	0.420	39.5	92.1	41.2	78.9	40.3	73.5	40.4	70.7	0.139	-0.142	0.375	0.169	0.419	0.264	0.441	0.291

TABLE 9. Retrieval: difference to scikit-learn baseline (ours − sklearn). Positive deltas are better for Accuracy and R^2; negative deltas are better for MAE. Green shading marks improvements; red shading highlights notable degradations.

	Accuracy								MAE								R^2							
	K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Retrieval	+0.020	-0.020	+0.020	+0.028	+0.040	-0.027	+0.045	-0.027	+9.134	-13.310	+5.139	-10.374	+2.621	-9.624	+1.427	-8.566	-0.180	+0.012	-0.114	+0.014	-0.104	-0.012	-0.090	-0.000
Shuffle-no + Retrieval	+0.005	-0.040	+0.020	+0.017	+0.005	-0.037	+0.015	-0.015	-6.381	+1.414	+9.890	+3.053	+2.924	+1.841	+4.044	+2.320	-0.030	-0.226	-0.037	-0.175	-0.063	-0.139	-0.058	-0.105

TABLE 10. Retrieval: difference to the default configuration (ours − default; default = Shuffle-yes + Retrieval). Green denotes improvement (higher Accuracy/R^2 or lower MAE); red marks substantial degradations.

	Accuracy								MAE								R^2							
	K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20		K=1		K=5		K=10		K=20	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Retrieval (default)	0.520	0.405	0.525	0.455	0.540	0.475	0.535	0.443	51.1	90.6	40.3	75.1	37.6	69.7	36.0	68.1	-0.011	-0.122	0.239	0.292	0.303	0.387	0.356	0.411
Shuffle-no + Retrieval	-0.005	-0.053	+0.010	-0.043	-0.020	-0.080	-0.010	-0.023	-11.600	+1.517	+0.840	+3.802	+2.704	+3.818	+4.428	+2.584	+0.150	-0.020	+0.136	-0.123	+0.116	-0.123	+0.085	-0.120

TABLE 11. Retrieval: effect of increasing K within each configuration. Columns for K=1 show absolute Tr/Te values; columns for $K \in 5, 10, 20$ show deltas relative to K=1 (same row). Green indicates improvement; red marks notable deterioration.

	Accuracy								MAE								R^2							
	K=1		K=5−K=1		K=10−K=1		K=20−K=1		K=1		K=5−K=1		K=10−K=1		K=20−K=1		K=1		K=5−K=1		K=10−K=1		K=20−K=1	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Retrieval (default)	0.520	0.405	+0.005	+0.050	+0.020	+0.070	+0.015	+0.037	51.1	90.6	-10.732	-15.506	-13.504	-20.850	-15.105	-22.424	-0.011	-0.122	+0.250	+0.414	+0.314	+0.509	+0.367	+0.533
Shuffle-no + Retrieval	0.515	0.352	+0.020	+0.060	+0.005	+0.043	+0.010	+0.068	39.5	92.1	+1.708	-13.221	+0.800	-18.549	+0.923	-21.358	0.139	-0.142	+0.236	+0.310	+0.280	+0.406	+0.302	+0.433

TABLE 12. Retrieval: stepwise deltas as K increases (5−1), (10−5), (20−10), for Accuracy, MAE, and R^2 (Tr/Te). Green indicates improvement; red marks notable deterioration.

	Accuracy								MAE								R^2							
	K=1		K=5−K=1		K=10−K=5		K=20−K=10		K=1		K=5−K=1		K=10−K=5		K=20−K=10		K=1		K=5−K=1		K=10−K=5		K=20−K=10	
Configuration	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
Shuffle-yes + Retrieval (default)	0.520	0.405	+0.005	+0.050	+0.015	+0.020	-0.005	-0.032	51.1	90.6	-10.732	-15.506	-2.773	-5.343	-1.600	-1.574	-0.011	-0.122	+0.250	+0.414	+0.064	+0.095	+0.053	+0.024
Shuffle-no + Retrieval	0.515	0.352	+0.020	+0.060	-0.015	-0.017	+0.005	+0.025	39.5	92.1	+1.708	-13.221	-0.909	-5.328	+0.124	-2.809	0.139	-0.142	+0.236	+0.310	+0.044	+0.095	+0.022	+0.027

the foundation model saturates in the mid-0.7 range. This illustrates a typical trade-off: when a substantial number of labeled prefixes is available for a specific log, a strong per-log discriminative classifier on top of the shared embedding can still be competitive or superior for next-activity prediction.

The regression metrics tell a complementary story. The middle panel of Figure 13 shows that the kernel head of the foundation model achieves substantially lower or comparable remaining-time MAE than ridge regression for most moderate context sizes (e.g., $K \in 2, 3, 4, 5, 7, 8, 10, 15, 20, 50, 75, 100, 350$), and remains relatively stable as K grows. In contrast, the ridge baseline exhibits pronounced sensitivity to K, with some configurations (e.g., K=3, K=10, K=20) producing very large errors. The right panel of Figure 13 highlights this effect even more clearly: while our kernel head stays near zero with modestly negative R^2 (reflecting a robust but conservative fit in a strict few-shot regime), the ridge baseline yields extremely negative R^2 values for certain K, indicating catastrophic overfitting to the small support sets.

The non-monotonic shapes of the `scikit-learn` curves, where the error can increase when K increases, are expected in this evaluation protocol for several reasons. First, as mentioned above, each value of K corresponds to a different randomly sampled episode. The support and query sets at K=20 are not obtained by simply adding examples

to those at $K=10$, so performance differences across K also reflect ordinary sampling variance: some episodes are intrinsically harder than others due to which prefixes and next activities happen to be drawn. Second, the logistic and ridge models are re-fit from scratch for every K using fixed hyperparameters (e.g., regularization strength), without per-K cross-validation. With very small or moderately sized supports, this can easily lead to ill-conditioned design matrices in the prefix-embedding space and to either under- or over-regularized solutions. Adding more points then changes both the conditioning of the problem and the mixture of local temporal regimes seen by the model; with fixed regularization, this can move the fitted parameters into a region that generalizes worse on the associated queries, even though K is larger. Third, the MAE and R^2 curves are based on finite query sets per K, so high variance in a few poorly fitted queries (e.g., when ridge regression extrapolates with large coefficients) can sharply deteriorate the aggregate error for specific support sizes.

Overall, this case study confirms the qualitative findings from the synthetic logs: per-log linear baselines can be strong when enough labeled data is available, but the nonparametric in-context heads of the foundation model yield more stable remaining-time predictions under tight data budgets and across a wide range of support sizes. The occasional increases in error for the `scikit-learn` baselines as K grows are therefore a consequence of the episodic few-shot setting (different episodes for different K), fixed hyperparameters, and the high variance of small-sample linear models in a high-dimensional embedding space, rather than an indication that additional information is intrinsically harmful.

I. STRENGTHS OBSERVED

Across tables and settings, two recurring strengths are observed in this evaluation: adaptation with small in-context data and a degree of robustness to several design choices and label variability. In episodic evaluation, increasing shots from one to a handful systematically boosts next-activity accuracy and typically improves remaining-time quality, with accuracy rising from $K=1$ to $K=5$ and $K=10$, while MAE falls and R^2 improves (Table 6, Table 7). This trend holds across variants that change model width (e.g., $d_{model}=128$ or 512), heads, or depth, with test scores often remaining close to the default (Table 3, Table 5), suggesting that the in-context mechanism contributes substantially to performance within this setup.

The approach performs particularly well for regression under retrieval: nearest-neighbor supports steadily cut MAE and yield positive R^2 at modest K, improving further with larger supports (Table 8, Table 11). Stepwise deltas show the largest gains from $K=1$ to 5, with positive but diminishing returns beyond (Table 12), which aligns with a practical use case where a small pool of comparable prefixes anchors predictions to a local time scale. Next-activity accuracy also improves up to $K=10$ before saturating.

The mixture-of-experts contributes stability without complex routing: varying expert counts yields broadly similar test performance in this benchmark (Table 3), consistent with variance reduction from aggregation, while keeping the heads nonparametric for adaptation. That said, because the training corpus is limited, specialization opportunities are likewise limited; a larger and more heterogeneous set of training logs would likely allow experts to cover a wider range of regimes more reliably.

Label shuffling acts as a regularizer in both regimes, improving test accuracy and often improving regression metrics (Table 4, Table 9, Table 10). It curbs reliance on global label identities and encourages support-set reasoning, which is beneficial when activity names differ across systems. Finally, training dynamics are stable across strategies, with losses dropping smoothly and with performance trends aligning as context grows (Figure 12). These strengths should be interpreted as observed behaviors on the evaluated logs; confirming them across a broader set of real-world logs remains an important next step.

J. LIMITATIONS AND FAILURE MODES

The results reveal clear boundaries for the approach, despite its strengths. A prominent limitation is remaining-time prediction in the episodic protocol: MAE decreases steadily with K, but R^2 remains negative in the default configuration (from -1.506 at $K=1$ to -0.224 at $K=5$, stabilizing around -0.18 for larger K; Table 3). This indicates that the few-shot kernel head can struggle to outperform a naive mean baseline when support examples mix disparate temporal scales or when the sampled supports are not sufficiently localized in the embedding space, even with stabilization. Retrieval mode largely mitigates this issue in our evaluation, yielding positive R^2 for $K \geq 5$ by selecting log-specific neighbors (Table 8), but this also makes performance dependent on retrieval quality.

Generalization gaps persist: episodic training metrics exceed test values (e.g., 0.851 vs. 0.732 accuracy, 60.1 vs. 93.1 MAE at $K=20$; Table 3), which can indicate overfitting to the training corpus and to the episode distribution, despite shuffling and heterogeneous training logs. Lighter training (10-30 epochs) sometimes generalizes better, underscoring the value of validation-driven checkpoint selection.

A broader limitation concerns the size and nature of the training and test corpus. The model is trained on a relatively small set of synthetic logs and evaluated on two held-out synthetic logs plus one real-life case study later in the section. This limits the diversity of organizational conventions, attribute sparsity patterns, noise sources, and control-flow idiosyncrasies encountered during training, and it also limits how confidently one can generalize from the reported test numbers. In particular, some of the learned regularities may be specific to the simulator design, and some failure modes that are common in operational settings (e.g., inconsistent timestamp granularity, concept drift across time, heterogeneous attribute schemas, or systematic missingness) are only partially represented. Expanding the pretraining corpus to include many more logs, especially real ones with different data quality profiles, is likely to both improve model

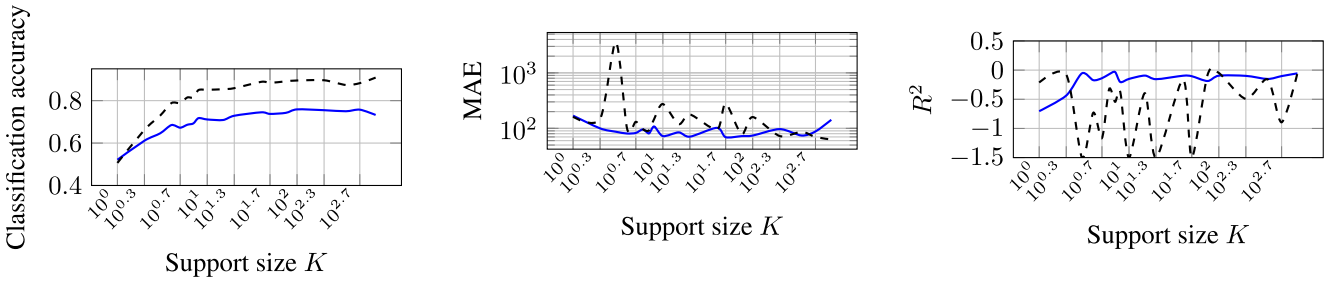


FIGURE 13. Performance on the WABO receipt log as a function of support size K (meta-learning mode). Left: Classification accuracy. Solid blue lines indicate the foundation model using a prototype-based in-context head; dashed lines indicate the logistic-regression baseline (scikit-learn) trained on the same prefix embeddings. Middle: Remaining-time mean absolute error (MAE; lower is better). Solid blue lines indicate the foundation model using a nonparametric kernel regressor; dashed lines indicate the ridge-regression baseline (scikit-learn) on the same prefix embeddings. Right: Coefficient of determination R^2 for remaining-time prediction (vertical scale clipped to $[-1.5, 0.5]$ for readability; several ridge-regression configurations yield much more negative R^2 values, e.g., $K = 3$, $K = 10$, $K = 20$, highlighting their instability in the few-shot regime compared to the foundation model).

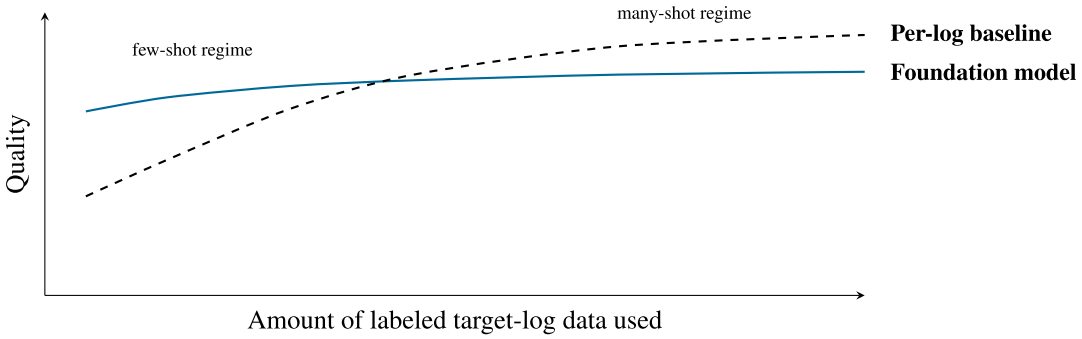


FIGURE 14. Qualitative expectation for classification on a new target log. With very few labeled target-log examples, an in-context foundation model can leverage its pretrained prior and outperform a per-log baseline. As the number of labeled examples grows, a dedicated per-log model can specialize and eventually surpass the foundation model.

quality and sharpen the understanding of when the in-context heads are preferable to per-log fitted models.

VII. ALTERNATIVE CHOICES

The architecture in Section IV is one concrete point in a broader design space for sequence-based prediction on event logs. Below we outline principled alternatives for each major component, highlight when they may be preferable, and provide links to representative papers.

Alternative encoders for prefix sequences offer diverse trade-offs in expressivity and efficiency. Recurrent architectures like LSTMs or GRUs provide stable summaries for short to medium prefixes with low latency, as widely used in predictive process monitoring [4]. Temporal convolutional networks (TCNs) enable parallel computation and long receptive fields via dilated convolutions, competing effectively on sequential tasks [16]. Within transformers, relative positional encodings or efficient variants such as Longformer reduce quadratic costs for longer cases [17]. Graph-aware options, including features from mined Petri nets or graph neural networks like GCNs, incorporate control-flow structure when predictions is based on routing or global neighborhoods [1]. For time-focused encoding, neural temporal point processes such as RMTPP model inter-event intensities, improving calibration for remaining-time tasks over discriminative regressors [18].

In-context adaptation and prediction heads can be varied to suit update budgets and output needs. Cross-attention mechanisms like Matching Networks allow direct query-support interaction without external prototypes, simplifying few-shot adaptation [19]. Gradient-based meta-learning, such as MAML, enables light test-time tuning when small updates are feasible [20]. For heads, relation networks generalize prototypes by learning metrics [21], while sequence-to-sequence models with attention predict multi-step suffixes [22]. Quantile regression or survival heads like DeepSurv extend remaining-time outputs to intervals or censored risks [23]. Event-to-vector mappings may enhance temporal signals with Time2Vec embeddings or leverage BERT for semantic strings, though lexical similarity requires structural fusion to avoid behavioral mismatches [24].

Training and context strategies further broaden the model's applicability in several ways. For instance, self-supervised pretraining, using techniques like masked modeling or contrastive losses (e.g., TS2Vec), can prime the encoder for better downstream transfer across tasks [25]. To handle uncertainty, methods such as deep ensembles or conformal prediction enable calibrated confidence scores, supporting decisions like abstention on low-certainty predictions [26]. Episodes can be enhanced with curricula to progressively increase difficulty or with hard-negative mining to sharpen decision boundaries. For simpler setups, classical non-neural

baselines (e.g., hidden Markov models or gradient boosting on engineered prefix features) offer strong interpretability and efficiency in data-scarce scenarios [27]. Finally, sparse-gated mixtures of experts with learned routing can improve specialization beyond random assignment [28].

VIII. CONCLUSION

This paper proposed an event-log-native foundation model for predictive process monitoring. The model encodes prefixes of running cases as sequences of timestamped events and adapts to a new log at inference time through in-context examples, without any parameter updates. To support both next-activity prediction and remaining-time estimation, we combined a shared prefix encoder with purely nonparametric in-context heads (prototype classification and kernel regression) and aggregated predictions across a small mixture of experts.

The core idea does work: a single pretrained model can transfer across logs and improve when provided with a small target-log support set, showing that in-context adaptation is a viable mechanism for process-mining prediction tasks. Across both episodic and retrieval-based contexts, the model consistently benefits from moving from extremely small contexts to modest ones (e.g., from $K=1$ to $K=5$-10), indicating that the representation learned during pretraining can be reused and anchored effectively by a handful of in-log examples.

Returning to the research questions posed in Section I, our experiments provide concrete evidence for **RQ1** and **RQ2**. For **RQ1 (cross-log generalization)**, the evaluation protocol mirrors deployment: pretrained checkpoints remain frozen, with predictions produced purely via context without gradient updates. For **RQ2 (how to form the context)**, we compare balanced episodic contexts and retrieval-based contexts (selected within the target log under same-case masking). Results show the expected trade-off: balanced few-shot episodes provide a controlled view of label-space adaptation, while retrieval-based contexts better anchor the temporal scale for remaining-time prediction when appropriate neighbors exist (cf. Section VI-F and Section VI-G).

The ablation suite and baselines address **RQ3–RQ5** (and partially **RQ4**). For **RQ3 (design choices)**, ablations indicate that context size K is the strongest driver, with label shuffling and mixture width offering smaller effects. For **RQ5 (baselines under the same data budget)**, comparisons are ''apples-to-apples'': baselines are fit only on the same support points and frozen prefix embeddings as the in-context heads, as described in Section V. For **RQ4 (reliability)**, confidence proxies (e.g., max softmax probability for prototypes and kernel concentration $\max_i w_i$ for regression, plus expert-level confidences) are implemented, but a dedicated calibration study remains a next step.

At the same time, the absolute results reported in this paper are not very good yet compared to what one would ultimately want from a production-ready foundation model. This is largely explained by the limitations discussed in the evaluation: the pretraining corpus is small (only a limited number of training logs), much of it is synthetic, and the held-out testing set is also small (two synthetic logs plus a single real-life case study). These constraints bound the diversity of control-flow patterns, naming conventions, attribute schemas, temporal regimes, and noise profiles the model can learn during pretraining, and they make test estimates sensitive to the particular sampled logs and episodes. Moreover, our method intentionally avoids target-log fine-tuning; consequently, when a large number of labeled target-log examples is available, a per-log discriminative baseline fitted on that log can still have an advantage, especially for classification.

Looking forward, we expect the qualitative behavior illustrated in Figure 14 to become more pronounced as the pretraining corpus grows. In particular, for classification tasks, a stronger and more diverse pretraining set should improve the model's inductive bias and make the foundation model exceed the baseline in a more significant way when only a small number of target-log examples is available (the regime where retraining is expensive or impossible). At the same time, we still expect the foundation model to be beaten when the number of target-log examples increases substantially, because per-log training can exploit abundant supervision to specialize to the idiosyncrasies of a single process.

The most direct next step is to scale the training and evaluation to a substantially larger and more heterogeneous collection of event logs, especially real logs with varying data quality and schemas. Beyond data scale, promising directions include improved support-set construction (more selective retrieval and diversity-aware contexts), stronger temporal modeling for remaining-time prediction under mixed regimes, and uncertainty-aware predictions and calibration. Finally, hybrid adaptation strategies (e.g., optional small-head fine-tuning or log-specific calibration on top of frozen prefix embeddings) could help bridge the gap between few-shot robustness and many-shot asymptotic performance while keeping the ''train once, reuse broadly'' foundation-model goal intact.

REFERENCES

[1] W. M. P. van der Aalst, *Process Mining—Data Science in Action*, 2nd ed., Cham, Switzerland: Springer, 2016.
[2] I. Teinemaa, M. Dumas, M. L. Rosa, and F. M. Maggi, ''Outcome-oriented predictive process monitoring: Review and benchmark,'' *ACM*, vol. 13, no. 2, pp. 17:1–17:57, 2017.
[3] I. Verenich, M. Dumas, M. L. Rosa, F. M. Maggi, and I. Teinemaa, ''Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring,'' *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 4, pp. 1–34, Jul. 2019.
[4] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas, ''Predictive business process monitoring with LSTM neural networks,'' in *Proc. CAiSE*, 2017, pp. 477–492.
[5] R. Bommasani et al., ''On the opportunities and risks of foundation models,'' 2021, *arXiv:2108.07258*.
[6] A. Das, W. Kong, R. Sen, and Y. Zhou, ''A decoder-only foundation model for time-series forecasting,'' in *Proc. ICML*, 2023.
[7] N. Hollmann, S. Müller, K. Eggensperger, and F. Hutter, ''TabPFN: A transformer that solves small tabular classification problems in a second,'' in *Proc. ICLR*, Aug. 2022.

[8] M. Spinaci, M. Polewczyk, M. Schambach, and S. Thelin, "ConTextTab: A semantics-aware tabular in-context learner," 2025, *arXiv:2506.10707*.
[9] SAP SE. (2025). *SAP-RPT-1-OSS Model Card*. [Online]. Available: https://huggingface.co/SAP/sap-rpt-1-oss
[10] J. Evermann, J.-R. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decis. Support Syst.*, vol. 100, pp. 129–140, Aug. 2017.
[11] M. Camargo, M. Dumas, and O. González-Rojas, "Learning accurate LSTM models of business processes," in *Proc. BPM*, 2019, pp. 286–302.
[12] T.-H. Nguyen, D.-L. Pham, and K. P. Kim, "Next process-activity prediction using switch-transformer: Approach, visualization, and performance evaluation," *Knowl. Inf. Syst.*, vol. 67, no. 12, pp. 1–28, Dec. 2025.
[13] S. Weinzierl, S. Zilker, S. Dunzer, and M. Matzner, "Machine learning in business process management: A systematic literature review," *Expert Syst. Appl.*, vol. 253, Nov. 2024, Art. no. 124181.
[14] A. Garza, C. Challu, and M. Mergenthaler-Canseco, "TimeGPT-1," 2023, *arXiv:2310.03589*.
[15] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," in *Proc. NIPS*, 2017, pp. 4077–4087.
[16] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018, *arXiv:1803.01271*.
[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. NIPS*, vol. 30, 2025, pp. 5998–6008.
[18] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song, "Recurrent marked temporal point processes: Embedding event history to vector," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1555–1564.
[19] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proc. NIPS*, vol. 29, 2016, pp. 3637–3645.
[20] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. ICML*, 2017, pp. 1126–1135.
[21] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1199–1208.
[22] D. Bahdanau, "Neural machine translation by jointly learning to align and translate," in *Proc. ICLR*, 2014.
[23] J. L. Katzman, U. Shaham, A. Cloninger, J. Bates, T. Jiang, and Y. Kluger, "DeepSurv: Personalized treatment recommender system using a cox proportional hazards deep neural network," *BMC Med. Res. Methodol.*, vol. 18, no. 1, p. 24, Dec. 2018.
[24] S. M. Kazemi, R. Goel, S. Eghbali, J. Ramanan, J. Sahota, S. Thakur, S. Wu, C. Smyth, P. Poupart, and M. Brubaker, "Time2Vec: Learning a vector representation of time," 2019, *arXiv:1907.05321*.
[25] Z. Yue, Y. Wang, J. Duan, T. Yang, C. Huang, Y. Tong, and B. Xu, "TS2Vec: Towards universal representation of time series," in *Proc. AAAI*, vol. 36, 2022, pp. 8980–8987.
[26] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Proc. NIPS*, 2016, pp. 6402–6413.
[27] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.
[28] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *Proc. ICLR*, 2017.