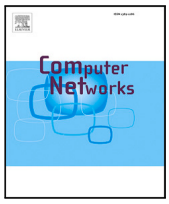
Computer Networks 269 (2025) 111395

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Exploring the application of Time Series Foundation Models to network monitoring tasks

Nikolas Wehner [a], Pedro Casas [b], Katharina Dietz [a], Stefan Geißler [a], Tobias Hoßfeld [a], Michael Seufert [c,*]

[a] University of Würzburg, Würzburg, Germany
[b] AIT Austrian Institute of Technology GmbH, Vienna, Austria
[c] University of Augsburg, Augsburg, Germany

ARTICLE INFO

Keywords:
Network monitoring
Time Series Foundation Models
Quality of Experience
Video streaming

ABSTRACT

Modern network monitoring applications often rely on traditional machine learning models conceived for specific analysis tasks, which require extensive feature engineering, retraining for different use cases, and struggle with generalization. This lack of adaptability makes the deployment of AI/ML solutions in network monitoring a daunting task, as each new scenario requires significant reconfiguration, manual tuning, and retraining efforts, undermining the broader adoption of AI/ML for network traffic analysis.

Time Series Foundation Models (TSFMs), pre-trained on vast and diverse time-series datasets, offer a promising alternative in the network monitoring realm by enabling zero-shot and few-shot adaptability across different monitoring scenarios. In this work, we explore the potential of TSFMs for network monitoring by evaluating their performance in a challenging analysis task: estimating video streaming Quality of Experience (QoE) from encrypted network traffic. Our study assesses the zero-shot and few-shot capabilities of state-of-the-art TSFMs, the impact of time-series granularity, and the role of common traffic features in performance.

Using real-world video streaming QoE datasets, we show that TSFMs achieve competitive results in a zero-shot setting — plug-and-play approach, and that their performance can be easily and cost-effectively improved through few-shot learning techniques, even when applied on NetFlow-like features with coarse granularity. Beyond the specific video streaming QoE monitoring application, our findings demonstrate the viability and broader applicability of TSFMs to network monitoring tasks, opening the door to more scalable and generalizable network management solutions.

1. Introduction

Estimating application-level Quality of Experience (QoE) from encrypted network traffic remains a complex-to-tackle challenge for modern network management systems. With the proliferation of encryption techniques, traditional packet inspection methods have become obsolete, leaving network operators with limited visibility into the performance of highly relevant services like video streaming. Recent years have witnessed a surge in Artificial Intelligence and Machine Learning (AI/ML) approaches to address this challenge. These methods typically extract session-based or window-based features from encrypted network traffic time-series data to predict QoE. While promising, AI/ML approaches come with significant hurdles when it comes to model training, deployment, and maintenance. Indeed, even if these approaches are data-driven, they often require extensive feature engineering, model architecture tailoring, and fine-tuning for specific applications. In addition, their limited generalization capabilities and transfer learning challenges require individual models or retraining for each service and each network, thereby reducing scalability [1,2].

To address these limitations, a new paradigm has recently emerged for time-series data: Time Series Foundation Models (TSFMs) [3]. Inspired by the success of Large Language Models (LLMs), TSFMs are designed to leverage extensive pre-training on diverse time-series datasets, enabling them to learn complex temporal patterns and make accurate predictions for unseen time series in a zero-shot learning (ZSL) or few-shot learning (FSL) setting [4]. With ZSL, no training is required at all, while with FSL only a few samples are utilized to fine-tune a model on a specific downstream task, e.g., video quality classification or anomaly detection. These capabilities are particularly appealing for

* Corresponding author.
E-mail address: michael.seufert@uni-a.de (M. Seufert).

https://doi.org/10.1016/j.comnet.2025.111395
Received 31 January 2025; Received in revised form 11 April 2025; Accepted 16 May 2025
Available online 2 June 2025

network monitoring, as they reduce the need for extensive training for each specific application. TSFMs have demonstrated remarkable performance across various domains without the need for task-specific training [5–10], significantly reducing the associated costs of model development and deployment. Further, a single TSFM can generalize across multiple services, addressing the scalability challenges of conventional methods.

The application of TSFMs to network monitoring holds several promising advantages: (i) firstly, it significantly reduces model training effort — TSFMs are "plug-and-play" models, and unlike traditional AI/ML approaches, TSFMs can potentially perform well without domain-specific training; (ii) due to their universal applicability, a single TSFM could potentially be used across multiple domains, significantly enhancing scalability, not only in a per-service or per-network basis, but also when it comes to deal with hundreds or thousands of time series generated by network monitoring systems; (iii) by eliminating the need for continuous retraining and maintenance of multiple models, TSFMs may substantially reduce operational costs. For these reasons, as model architectures evolve and become more efficient, we expect TSFMs to play an increasingly important role in network management.

Both the ZSL and FSL concepts are powerful and appealing for network monitoring applications. These approaches offer several inherent advantages: first, ZSL simplifies the application of the model for time-series modeling, in the best case, eliminating the requirement for specialized knowledge of fine-tuning techniques; secondly, they naturally align with scenarios characterized by limited data availability, where training or fine-tuning data is limited; lastly, by harnessing the comprehensive pattern extrapolation capabilities of extensively pre-trained models, it circumvents the substantial time, effort, and domain-specific expertise typically demanded for crafting dedicated time-series models.

There is a recent surge in papers targeting the conception of foundation models for time-series data, capable of generating accurate predictions for diverse datasets not seen during training. The underlying concept of these models is to rely on highly expressive, large-scale architectures which are trained on billions of time-series data points, coming from very diverse domains and having high heterogeneity in terms of temporal behaviors and characteristics. TimeGPT-1 [11], PromptCast [12], LLMTime [13], TimesFM [14], Lag-Llama [15], and Time-LLM [16] are examples of novel foundation models for time-series forecasting targeting ZSL and FSL applications. Adapting these models to network monitoring tasks remains a largely unexplored field, with only a few initial works exploring this problem [17].

Despite their promising advantages, the application of TSFMs is not without challenges. One significant limitation is the high computational cost associated with pre-training and inference of many models, which can render their deployment infeasible for resource-constrained environments or real-time network monitoring scenarios. Additionally, while TSFMs offer broad applicability, their generalist nature may lead to suboptimal performance in specific use cases where domain-specific fine-tuning can significantly enhance results. Furthermore, TSFMs often lack explainability, making it difficult to interpret model decisions or detect biases. These challenges underscore the importance of systematically evaluating state-of-the-art TSFMs to understand their practical applicability, identify scenarios where they excel, and determine the trade-offs involved in their deployment. This study aims to address these gaps by benchmarking state-of-the-art TSFMs for video streaming monitoring. We further analyze the impacts of network traffic granularity and effectiveness of common features on the performance of TSFMs. Additionally, the study evaluates the ZSL and FSL capabilities of TSFMs.

In this paper, we thus explore the potential of TSFMs for estimating video streaming Quality of Experience (QoE) from encrypted network traffic. Our key contributions are as follows:

- **Application of TSFMs to Network Monitoring:** we demonstrate the applicability of TSFMs for network monitoring, focusing on the analysis of encrypted traffic to estimate video streaming session health, while also providing an introduction to TSFMs for networking engineers or researchers unfamiliar with this domain.
- **Benchmarking State-of-the-Art TSFMs:** we evaluate the performance of state-of-the-art TSFMs on a large-scale video streaming QoE dataset, assessing their ability to predict QoE metrics in a zero-shot setting without task-specific training.
- **Analysis of Common Traffic Features:** we assess the role of widely used NetFlow-like network traffic features in shaping TSFM predictions, offering guidance for feature selection in network monitoring applications.
- **Analysis of Traffic Granularity Levels:** we investigate the impact of different temporal granularities on TSFM performance, providing insights into the most effective temporal resolution for network monitoring tasks.
- **Fine-Tuning with Tiny Time Mixers:** we showcase the performance improvements achievable by fine-tuning a TSFM such as Tiny Time Mixers in the few shot setting, provide guidance on fine-tuning techniques, and demonstrate that existing models can be effectively fine-tuned in practice, while maintaining low hardware and time requirements.

2. Related work

2.1. QoE estimation from encrypted network traffic

Traditionally, deep packet inspection (DPI) has been employed to analyze complete packet streams, enabling the identification of applications and their QoE within a network, e.g., in [18]. However, the widespread adoption of end-to-end encryption has significantly reduced network operators' visibility into the performance of services consumed by their users. To address these challenges, artificial intelligence (AI) and machine learning (ML) technologies are increasingly being adopted as scalable solutions for advanced traffic monitoring and performance assessment. To address this, [19] explored the use of features exclusively derived from encrypted traffic, although the application-level ground truth was obtained from non-encrypted flows captured via a web proxy. Subsequently, [20–23] introduced methodologies fully operable in encrypted traffic contexts relying solely on the statistical properties of encrypted traffic volumes, and thus, eliminating the need for access to packet payloads at any stage of model development or training. These studies demonstrated the feasibility of accurately classifying YouTube video streams into QoE categories on a per-video and session-level basis. A related approach was developed in [24], where ML-based QoE prediction used simple flow-level traffic characteristics in combination with mobile network connectivity features derived from passive in-device measurements as input.

In addition to session-level solutions, which are better suited for long-term network planning, network operators also desire real-time monitoring capabilities to dynamically monitor QoE and allocate resources more effectively. Several real-time KPI estimation methods have been proposed [25–35]. While these approaches perform well in lab environments, the practical deployment of these solutions often falls short. The deployment of ML-based models in the 5G NWDAF (Network Data Analytics Function) was tackled in [36–38] and assessed via simulations. Marina [2] provides accurate ML-based real-time QoE monitoring at terabit scale by spreading monitoring over a highly efficient data plane, such as P4, which can extract traffic statistics at line rate, and a powerful ML server, which can run monitoring inference using complex ML models. This approach is applicable to a wide range of real-time network monitoring tasks including traffic classification, device classification, and intrusion detection, which can also be tackled with representation learning [39].

The proposed approaches for QoE estimation have demonstrated strong performance when tailored to specific use cases, such as a particular streaming service or a controlled experimental setup. However, their ability to generalize across diverse networks and varying network conditions remains uncertain. In practical applications, the input data for prediction models often evolves over time, leading to performance degradation as new data diverges from the original training distribution. Additionally, subtle differences can arise between similar datasets generated by different sources or environments. This challenge, broadly known as *dataset shift* or *dataset drift*, can significantly affect model reliability [40,41].

Cross-testing efforts have highlighted the challenges of generalization. For instance, studies such as [22,42] evaluated the transferability of YouTube QoE/KPI classification models trained in a controlled lab environment when applied to data collected from an operational mobile network. Similarly, models trained on data from Android platforms were tested on datasets from iOS, revealing significant performance declines [22]. Despite these limitations, the creation of general models trained on heterogeneous datasets containing samples from both platforms has shown promise, achieving performance comparable to platform-specific models [32]. Similar conclusions have been presented in [31], where general models could perform well if the training set included data from all services. General models can also be learned via federated learning and model sharing from local data and models, which has already been explored in the QoE domain, e.g., in [43–46].

Given that the availability of labeled data in general, and from different sources in particular, for training a general model is often very limited, another option is explicitly improving the transfer of specialized models, i.e., trained on a single domain, to other domains. Reference [47] examined the performance of ML-based QoE estimation models for encrypted network traffic using video streaming measurements from two different locations. It was found that general models perform comparably to network-specific ones, while cross-network testing shows significant performance degradation. Extending this work, reference [1] subsequently explored the performance of different methods to enhance cross-network applicability requiring no or only minimal additional labeled data from the new domain, including methods based on scaling, decomposition, manifold learning, ML-based feature representation transfer, drift elimination, and enrichment with labeled data from the new domain. Reference [48] demonstrates for the domain of cloud gaming that transfer learning and fine-tuning significantly improve the generalization of QoE estimation models across diverse gaming contexts, reducing both error and the need for extensive labeled data in new environments.

In this work, we tackle a third approach to this challenge by relying on foundation models, which are pre-trained using large amounts of diverse data, such that they can generalize and perform well on arbitrary prediction tasks with little additional information, including network monitoring in the presence of dataset shift or drift.

2.2. Generative artificial intelligence for time series

Modern approaches to time-series analysis based on deep learning technology have flourished in recent years, in particular for the problem of anomaly detection [49], a cornerstone of network management and a critical application in time-series analysis. Due to their data-driven nature and outstanding performance in multiple domains, generative models such as Variational Autoencoder (VAEs) and Generative Adversarial Networks (GANs) [50] have gained relevance in the anomaly detection field [51–57].

VAEs [58–60] represent a powerful and widely-used class of models to learn complex data distributions. Unlike GANs, a potential limitation of VAEs is the prior assumption that latent sample representations are independent and identically distributed. While this is the most common assumption followed in the literature, there is ongoing research on the benefits of accounting for covariances between samples in time to improve model performance [61–64].

Modeling data sequences through a combination of variational inference and deep learning architectures has been vastly researched in other domains in recent years, mostly by extending VAEs to Recurrent Neural Networks (RNNs), with architectures such as STORN [65], VRNN [66], and Bi-LSTM [67] among others. Convolutional layers with dilation have been also incorporated into some of these approaches [68,69], allowing to speed up the training process based on the possibilities of parallelization offered by these architectures. One of these approaches using Dilated Convolutional Neural Networks as the encoder–decoder architecture for VAEs is the DC-VAE model [70,71].

Transformer-based models [72] are gaining popularity in recent years for time-series analysis, given their remarkable performance in large-scale settings, such as long sequence time-series forecasting (LSTF). LSTF requires capturing long-range dependencies between input and output efficiently. Earlier examples include the TFT interpretable model [73] and the MQTransformer model [74]. The Informer model [75] introduced Transformers for long sequence forecasting through sparse self-attention mechanisms. This concept has since been further refined through various forms of inductive bias and attention mechanisms in models like Autoformer [76] and FEDformer [77].

Finally, there is a recent surge in papers targeting the conception of TSFMs, capable of generating accurate predictions for diverse time-series datasets not seen during training. The underlying concept of these models is to rely on highly expressive, large-scale architectures which are trained on millions or billions of time-series data points, coming from very diverse domains and having high heterogeneity in terms of temporal behaviors and characteristics [5,8–16].

2.3. Generative artificial intelligence for networks

Similarly, there has been a recent interest in generative applications for networking, for example, for the generation of synthetic network traffic and for network monitoring and management or other related tasks [78–80].

In earlier days, many works attempted to generate synthetic network traffic with GANs [81–83], the standard for image generation in Computer Vision (CV) at that time. Today, mainly diffusion models [84] are used for image generation. As a consequence, researchers have also recently proposed diffusion-based models suitable for the generation of synthetic network traffic [85–87]. Additionally, there are now also Transformer-based approaches, such as PAC-GPT [88], to produce realistic synthetic traffic traces. This GPT-3-based model translates natural language descriptions of network flow requirements into tabular data, then passes them onto the Transformer-based packet generator to produce a PCAP file.

Similarly to the time-series domain, Transformer-based models have also been adopted in various works for network monitoring [89–94]. Most of these models utilize a modified form of Masked Language Modeling for pre-training [95], where parts of the input are masked and have to be reconstructed from context. As Transformer-based models require either discrete input tokens or embeddings as input, the considered network traffic has to be transformed to meaningful tokens or embeddings first. The preparation of these tokens or embeddings is also one of the major differences between existing works. While one work directly uses packet payloads [89], another work uses the concept of building datagrams from packet bursts [90], and other works consider (multi-)flow representations as inputs [92,94]. After pre-training, these models are evaluated on common encrypted network traffic downstream tasks, e.g., service and application classification, IoT device classification, or intrusion detection.

Unlike previously discussed methods that involve training Transformer-based models directly on networking data, there exist also approaches that exploit the capabilities of already pretrained Large Language Models (LLMs). NetLLM [96] leverages LLMs combined with

Table 1
Overview of TSFMs considered in this work. The character *T* corresponds to Tiny, *B* to Base, *S* to Small, *L* to Large, *R1* to Revision 1, and *R2* to Revision 2. The representation corresponds to the number of elements in the 1D vector representing the encoded time series.

	Chronos-T5 [5]	Lag-LLama [6]	TimesFM [7]	Moirai [8]	MOMENT [9]	Tiny Time Mixers [10]
Publication	2024	2023	2023	2024	2024	2024
Publisher	Amazon	Meta	Google	Salesforce	Auton lab	IBM
Statistics	Univ.	Univ.	Univ.	Multiv.	Multiv.	Multiv.
Model variants	T	B	B	S/B/L	S/B/L	R1/R2
Model size [MB]	34	9.8	814	55/365/1240	152/454/1390	0.377
Parameters [M]	8.4	2.45	200	13.8/91.4/311	37.9/113/346	0.805
Representation	256	144	1280	384/768/1024	512/768/1024	1536
Context length	512	512	512	512	512	512/1024/1536
Tokenization	Quantization	Lagging	Patching	Patching	Patching	Patching
Backbone	Encoder+Decoder	Decoder	Decoder	Encoder	Encoder	TSMixer

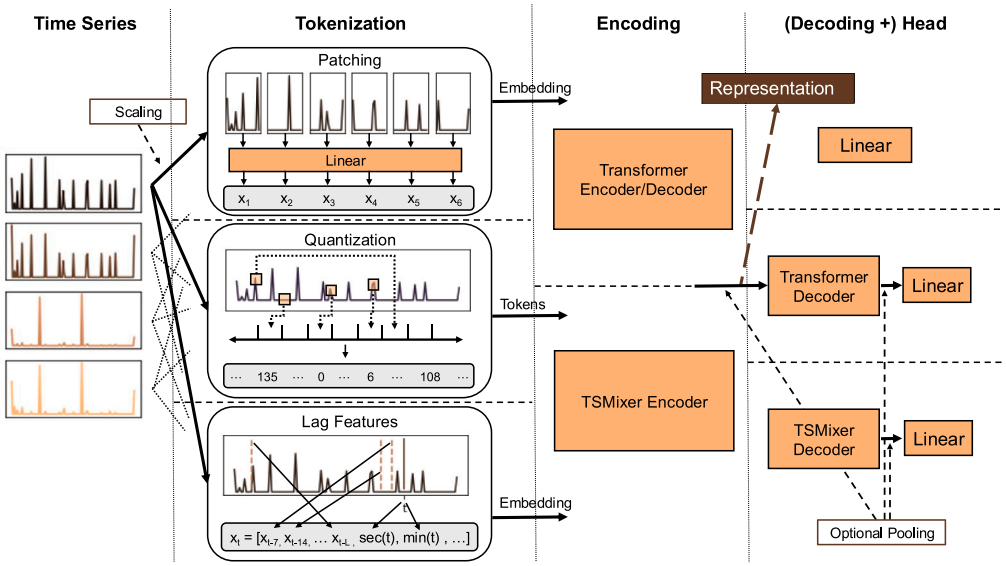


Fig. 1. A simplified overview of TSFMs internals.

a multimodal encoder and low-rank adaptation [97] to tackle various networking tasks, e.g., viewport prediction, adaptive bitrate streaming, and cluster job scheduling. The multimodal encoder is required to map arbitrary networking data inputs like time series, image, or graphs to token embeddings, which are then fed to an LLM. With ShieldGPT [98], the authors propose a method to perform automated Distributed Denial of Service (DDoS) mitigation with LLMs. The method first classifies the DDoS attack type with a pre-trained classifier and additionally extracts network traffic representations. Together with domain knowledge, an explanation and a mitigation prompt are generated based on pre-defined templates for an LLM. The LLM then returns an explanation for the DDoS behavior and a DDoS mitigation strategy.

In the networking domain, too, foundation models recently tracted interest, leading to the development of models like LENS [99] and netFound [100]. However, these models share many similarities to earlier transformer-based approaches like ET-BERT [90].

In contrast to previous work, we deploy already pretrained TSFMs for tackling QoE estimation from encrypted network traffic, neither directly relying on LLMs nor pretraining any model from scratch.

3. Time series foundation models

The recent advances in AI, in particular, in the domain of Natural Language Processing (NLP) and CV, led to the development of foundation models [101], which are models pre-trained on huge amounts of data at scale and usable across different downstream tasks, e.g., GPT-4 [102], SAM [103], and DALL-E [104]. This paradigm shift also generated research interests in other domains, e.g., geography [105] or medicine [106], and subsequently also in the time series domain [3].

3.1. Overview

TSFMs are pre-trained on large amounts of time-series data and are thus able to perform different tasks effectively, e.g., imputation, classification, anomaly detection, and forecasting [3]. In this work we consider six TSFMs, published since end of 2023, and evaluate their applicability for network monitoring. We provide an overview of these models in Table 1.

All models, except for Tiny Time Mixers (TTM) [10], are based on the Transformer architecture [107] and use either the Transformer encoder, decoder or both the encoder and decoder in their architecture. In contrast, TTM is based on TSMixer [108,109], a time-series forecasting architecture that relies only on Multilayer Pereceptrons (MLPs). As a consequence, TTM contains fewer model parameters, leading to a smaller model size compared to the Transformer-based models.

We refer the reader to the individual papers for an in-depth explanation on how the architecture of each foundation model specifically looks and how each foundation model was exactly trained. In Fig. 1, we provide only an abstract view of the most common techniques used to make TSFMs work: First, the input time series is scaled (either manually or by the model). In the multivariate case, the time series of each channel, i.e., time-series feature, is scaled independently of other channels. The scaled time series are then split into tokens using either most commonly patching, quantization (Chronos-T5 [5]), or lagged features (Lag-Llama [15]). Again, this processing is performed simultaneously and independent for each channel.

Patching consists of two operations: First, the time series is split into non-contiguous, fixed length sub-sequences. Second, the sub-sequences are then projected to d-dimensional embeddings, which represent the

actual tokens. With quantization, the continuous time series is binned uniformly or based on quantiles, so that the bin number corresponds to the respective token. Lagged features for a timestamp comprise past values with different lag steps along with date-time features characterizing the current timestamp *t*, e.g., the second of *t*, the minute of *t*, the month of *t*, etc. The resulting vector corresponds to the token for the timestamp. The tokenized or embedded time series are next encoded with the pre-trained backbone architecture, i.e., Transformer encoder, Transformer decoder, or TSMixer encoder. The representations produced by this encoding are then fed to the fine-tuned head of the foundation model, i.e., the final layer, to perform the desired downstream task. In the multivariate setting, the channel-wise representations must be additionally mixed first by using for example average pooling, maximum pooling, or more sophisticated methods like Tiny Time Mixer's exogenous channel mixing to obtain the final representations. Afterwards, the head of the model performs the actual prediction based on these representations. Note that we also exploit these representations to perform our zero-shot inference later.

Similar to related work and most foundation models from NLP and CV, most TSFMs are also pretrained by masking random parts of the input time series (or random tokens of the quantized time series) and then learn to predict the masked parts or tokens based on context only. Additionally, data augmentation is an often used tool to increase variance in the training data. The pretraining is performed on a training corpus consisting of several datasets from different domains (electricity, weather, sensor data, etc.) with huge amounts of time-series data.

3.2. Implementation

All models and variants presented in Table 1 are publicly available on HuggingFace[1] and each of these models comes by default with a head trained on the forecasting task. Note that we do not use all existing model variants for our analysis in this work, e.g., Chronos-T5 has also larger variants with more parameters, but list only the used variants in this work. Inference time increases for larger models significantly, making these variants in their current state undesirable for efficient network management. Additionally, we want to remark that all these models were trained on a granularity of one minute or higher, i.e., they were not trained to perform predictions on intervals of seconds at all, as would be required for real-time network monitoring tasks, e.g., Marina [2].

To perform zero-shot inference, we need to extract the representations of the time series provided by the backbone encoder. Fortunately, the HuggingFace models of Chronos-T5 and MOMENT already provide methods to retrieve these representations. We adjust the code of the remaining four models such that they also provide methods to return the representations. For Tiny Time Mixers, we extract the representations returned by the TSMixer backbone encoder. For the other three models, we use the representations returned by the respective Transformer encoder or decoder.

4. Data & preprocessing

4.1. Datasets

To assess the potential of TSFMs for network monitoring, we evaluate their performance on a challenging analysis task: estimating video streaming QoE from encrypted network traffic, using source files from a video streaming measurement dataset [1,47]. These measurements were conducted with a Selenium-based framework [26,110] that automatically starts an isolated Chrome browser session and browses to the video streaming service main page to access a single video page. A JavaScript-based monitoring script is then injected into the web page by the browser to monitor the current timestamp, the current video playtime, buffered playtime, video resolution, and player state periodically every 250 ms. Simultaneously, tshark captures the resulting network traffic. As a consequence, the data provides both network-level and application-level information. Measurements were conducted in Würzburg, Germany and in Zagreb, Croatia, in both cases in a fixed broadband network. To generate variation in the network, network conditions were emulated with Linux tc. In detail, no limitation, a fixed limitation of 1 Mbps, and a stochastic limitation following an exponential distribution with a mean of 1 Mbps were applied. During 2020 and 2021, a total of 2000 different, popular videos were streamed for around 180 s or till the end of the video for the various network conditions in both locations, with and without ad-blocking, resulting in a total of approximately 48,000 streamed video sessions. In this work, we take about half of these video sessions to accelerate the analysis.

4.2. Preprocessing

In the following, we describe the methodology to perform zero-shot classification with TSFMs on the video streaming use case, i.e., we do not perform any training at all, but simply download the models and weights, extract the representations during inference, and compare them to context samples. We provide a general overview of this methodology in Fig. 2. Time series with network-level feature inputs are transformed with TSFMs to vector representations, which are supposed to encode characteristics of the input time series. In the figure, we use t-SNE to perform dimensionality reduction to visualize the relationship between the representations of the instances (each dot is a test instance). Using an expert's domain knowledge, we generate representations for context samples with known labels, such that the distances between the representations of context samples and test instances can be used to label test instances in a zero-shot manner.

4.3. Inputs

While packet-level monitoring, e.g., with Intel Tofino P4 switches, allows the computation of packet inter-arrival times or more sophisticated features, commonly implemented solutions like Netflow provide only flow summary statistics, e.g., the total downlink and uplink volume or the number of downlink and uplink packets, in specific time intervals. However, many works have also shown that packet inter-arrival times are an important feature for encrypted network traffic analytics [2]. To have a trade-off, we thus restrict our feature set to the total volume and the mean inter-arrival times for uplink and downlink. For each timestep in our time series, regardless of granularity, we calculate the following four features based on all packets since the previous timestep: downlink volume (DV), uplink volume (UV), downlink mean inter-arrival time (DI), and uplink mean inter-arrival time (UI). To analyze the impact of flow statistics granularity, we aggregate the streaming sessions with different **granularities**, resulting in different time series lengths. In detail, we consider granularities of 1 s, 3 s, 5 s, and 10 s. The resulting CDF for the number of timesteps per granularity is depicted in Fig. 3(a).

As ML models require a fixed input shape and the duration of the video streaming sessions varies, our inputs must be padded to the same length. We refer to this length here as **context length** as it provides the context for our classification and is a commonly used term in time-series modeling. As depicted in Table 1, most models require an input with context length 512. We can easily see that most of our (resampled) video streaming sessions do not get even close to this context length. This may become problematic as TSFM performance is supposed to deteriorate with fewer inputs. To nevertheless obtain these context lengths, we pad all time series with zeros and provide masks to the TSFMs if applicable. Note further that we assume a perfect global standard scaler, i.e., the features of each time series are standardized across all samples in our dataset.

[1] https://huggingface.co/.

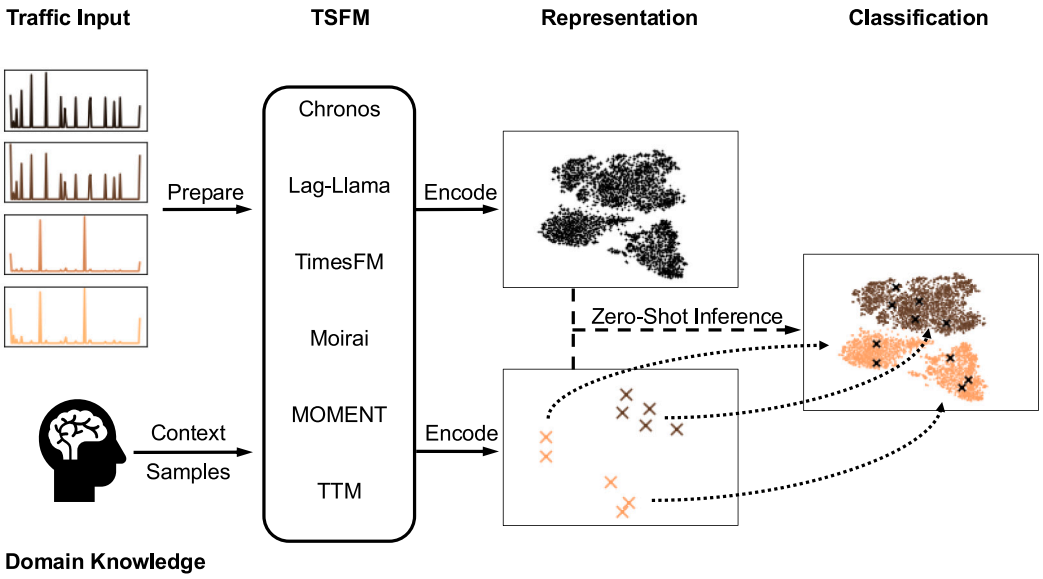
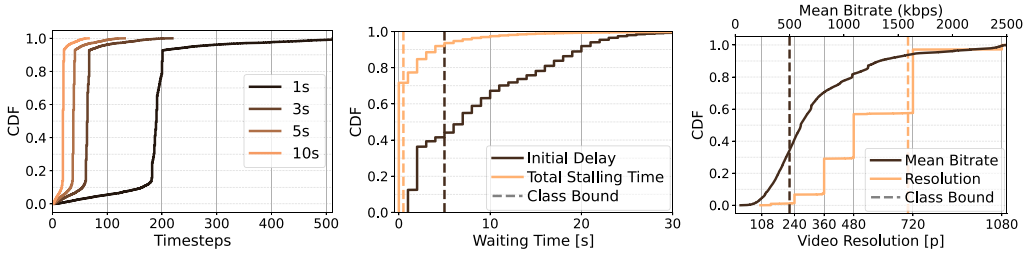


Fig. 2. An overview of zero-shot inference with TSFMs.



(a) Granularities. (b) Waiting times. (c) Video quality.

Fig. 3. Data and class characterization.

4.4. Encoding

We feed the padded time-series inputs to the TSFM backbone encoder to extract time-series representations of the individual models. For the univariate models, we extract the representation for each feature separately, before we perform an additional average or maximum pooling on the stacked feature representations to obtain the final time-series representation. For the multivariate models, MOMENT directly provides the representations, while Moirai and Tiny Time Mixer return a representation for each timestep in the time series. To obtain the final time-series representation, we thus apply average or maximum pooling over the timesteps. The vector representations are supposed to capture input time-series characteristics, clustering similar time series and distancing dissimilar ones in representation space.

4.5. Labels

Seufert and Orsolic infer QoE KPIs from encrypted network traffic using window-based and session-based (i.e., per video, tabular data) features [1] and discuss how to improve transfer learning on these features. Opposed to their work, we consider time-series data here, covering the entire video session. As both our time-series data and the session-based data consider the whole session, we reuse parts of the session-based classification settings of Seufert and Orsolic in this work. In detail, we define binary classification settings for the application-level KPIs **initial delay** (<5 s), **stalling** (true/false), **video resolution** (<700 p), and **mean video bitrate** (<500 kbps) [111]. The initial delay refers to the waiting time between the user initiating the playback and the actual video playback start. During this period, the video player downloads the initial video segments, potentially causing bursts of downlink traffic in the network. Stalling is defined as the playback interruption caused by buffer underrun, i.e., the video buffer contains no more content for playback. Buffer underrun is caused by the network not being able to sustain the current video quality or by network outages, often leading to periods of low network activity. Here, we do not distinguish between single or multiple stalling events within a session. Instead, we focus on predicting whether any stalling event occurred in a session. Video quality is determined by the video resolution and the bitrate used to encode video segments (measured here as the mean bitrate across the entire video). Higher video quality generally results in larger video segment download sizes, leading to more network traffic. While the video bitrate is designed to align with the downlink bitrate, video resolution is more challenging to infer as it is not directly reflected in the network. An overview on the characteristics of the waiting times (initial delay and stalling in form of total stalling time) and the video quality (video resolution and mean bitrate) is provided in the form of CDFs over the entire dataset in Figs. 3(b) and 3(c), respectively. The vertical dashed lines denote the classification setting, i.e., the point where the respective classes start and end. Note further that there are two x-axes in Fig. 3(c) and that the lower x-axis belongs to the video resolution (orange) and that the upper x-axis belongs to the mean bitrate (brown). We justify the choice of binary classification with the fact that, from the perspective of a network or service provider, a rough binary estimate might be sufficient to judge service or network performance. Additionally, it allows us to compare our results to related work [1].

5. Zero-shot learning

With zero-shot learning (ZSL), a model is able to perform tasks on data or recognize classes never seen during training. To make this work, the model requires supplementary context information to be able to generalize beyond the training data. In NLP, this supplementary context is generated by extending the actual prompts, e.g., by providing a written description of how classes are defined. However, in this work, we focus on a zero-shot classification task for time-series data, where the concept of textual prompting does not apply. Nevertheless, we must define classification context to enable the model to identify classes never seen during training. To create this context, we thus define samples used as anchors for indicating where the new class is located in representation space. We refer to these samples as **context samples** for the rest of this work. Obtaining such context samples requires a context sampling strategy and domain knowledge though, as these samples should be highly representative for a specific class. In this work, however, we focus only on random sampling, leveraging domain knowledge by utilizing the actual labels. Fig. 2 shows that a domain expert identified context samples (in representation space) for two classes (brown and orange) and that the context samples are used to classify the test instances with unknown labels. The classification of a test instance corresponds to either comparing the distances between the representation of the test instance and the representations of the context samples, or to training an additional supervised classifier, such as Support Vector Machine or Decision Tree, on the extracted representations. In this work, we use K-Nearest Neighbor (KNN) for classification, which classifies a sample based on a majority vote of the k closest neighbors, with closeness defined as the distance between representations, e.g., Euclidean distance. For all experiments, we fix $k = 5$ as it aligns with the number of context samples considered, and use the Euclidean metric for distance computation. Note that these hyperparameters could be tuned in practice to optimize results.

We conduct extensive experiments covering the introduced parameter space and consider six TSFMs with different sizes and different pooling strategies, resulting in 19 model variations. We evaluate model performance with the macro F1-score as it incorporates both precision and recall and counterbalances class imbalances. For all experiments, we evaluate 30 folds and compute mean and standard deviation for the observed macro F1-scores. Additionally, we set the same random seed before each such experiment so that always the same context samples per experiment are sampled and results for different parameter settings are comparable.

5.1. Baseline

As an initial experiment, we evaluate all 19 model variants using a single feature – downlink volume (DV) – with a 1-s granularity, a context length of 512, and a context size of 100. The results from this experiment serve as the baseline for all subsequent analyses.

Fig. 4 depicts the obtained results for the baseline experiment. The figure contains a sub-figure for each application-level target KPI, i.e., initial delay, stalling, resolution, and average bitrate. The x-axis denotes the macro F1-score from 0 to 1, where higher scores indicate better performance. The shared y-axis lists the model variations, with the model size and model pooling strategy in brackets. The horizontal bars show the achieved macro F1-score and the error bars indicate the standard deviation observed over the 30 folds, i.e., 30 context sample variations. The numbers to the right of the bars indicate the mean. This applies to all bar figures throughout this work. Here, we focus on the filled bars, indicating a perfect global scaler. We observe that most models perform excellent for the initial delay, achieving macro F1-scores of at least 0.80 and up to 0.85, e.g., MOMENT (Large). Only TimesFM fails to perform well here. It can be easily seen that TimesFM fail to perform well for the other KPIs, too. For the stalling estimation, i.e., the presence of stalling events in a session, we observe that the performance is generally low. Most models achieve a macro F1-score of around 0.52, with TTM (R2) achieving the highest scores of 0.55. Our results thus indicate that the models were not able to derive any meaningful relationships between specific time-series patterns and the appearance of stalling events in the zero-shot setting. For the estimation of the video resolution and the average bitrate, we obtain macro F1-scores of up to 0.72 and 0.68, respectively, a generally acceptable performance. MOMENT (Large) and Chronos-T5 (Tiny + Max) achieved the best performance across the KPIs, although the performance differences compared to the other models were relatively minor.

When comparing the results to related work [1], as depicted by the blue horizontal bars, we detect that the session model trained on tabular data (Seufert & Orsolic '23) outperforms the zero-shot TSFMs by 0.13 points for the KPIs initial delay, resolution, and average bitrate, and by a substantial 0.34 points for stalling detection. Note that the performance is not completely comparable, as different parts of the datasets were used and the test set varied in size. Nevertheless, understanding the reasons for these performance differences, in particular for stalling detection, is an interesting research question and should be addressed in the future. Since the tabular session model uses statistical descriptors for six features as model input, TSFM performance in stalling detection may be enhanced by identifying key statistical descriptors and incorporating them into the TSFM input, too.

5.2. Impact of scaling

In our next experiment, we analyze the impact of the scaling technique. Remember that we assumed a perfect scaler in the previous experiment, i.e., the standard scaler was fitted on the entire dataset. In this experiment, we compare this perfect standard scaler to a local standard scaler, i.e., a scaler which standardizes each session (time series) independently of all other sessions. Note that we explicitly utilize the standard scaler here, as most TSFMs require standardized input. A local standard scaler would be beneficial from the point of deployment as it would reduce dependencies on other data points. Empty bars in Fig. 4 correspond to the results of the local scaler. First of all, we observe that some models are more robust to scaling changes than others. For example, MOMENT and TTM show stronger differences for all KPIs compared to MOMENT and Moirai-1.1. Further, it is visible that basically all models perform slightly better with a global scaler for all KPIs, except for stalling where performance is rather identical. In particular for video quality estimation (resolution and average bitrate), the global scaler clearly improves the performance of most models. This is reasonable, as the amplitudes of an independently standardized session are not comparable to other sessions. Without comparable amplitudes, estimating video quality based on download volume becomes more challenging.

Summarizing, a well-designed global standard scaler can improve TSFM performance, particularly for video quality estimation. However, local scaling offers slightly lower performance while providing a practical trade-off by removing the need to collect representative samples for fitting a high-performing standard scaler.

5.3. Impact of features

So far, we used only the downlink volume to characterize a video session. Next, we analyze whether additional or alternative features significantly improve performance compared to the baseline experiment. We keep all parameters as in the baseline experiment, and vary the feature set only. We analyze the following feature sets: each feature individually (DV, UV, DI, or UI); combined downlink features (DV + DI); combined uplink features (UV + UI); volume-related features only (DV + UV); inter-arrival time features only (DI + UI); and all features together (DV + UV + DI + UI).

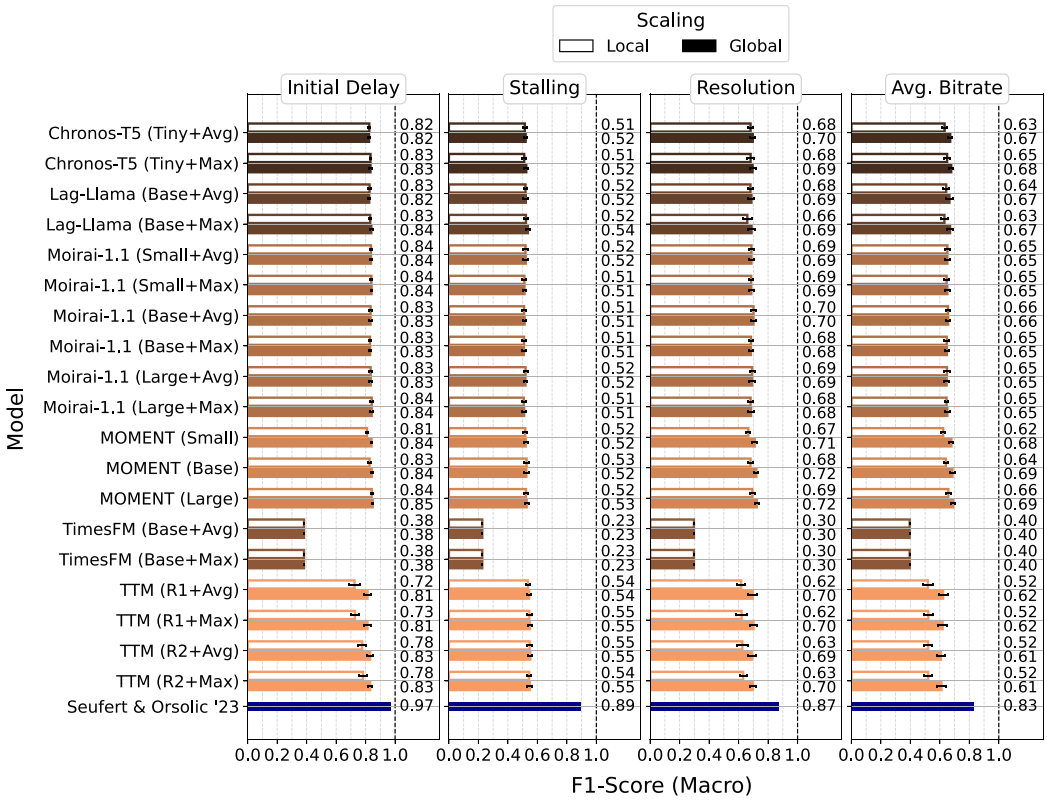


Fig. 4. Performance evaluation of baseline and scaling experiment.

Fig. 5 depicts the results for selected, best performing variants per model and the individual feature sets. Bars are colored according to the selected feature set and indicate mean and standard deviation of the observed macro F1-scores across 30 folds. Additionally, the bar for the feature downlink volume is hatched and represents the baseline, corresponding to the results of the previous experiments. For Chronos-T5, we observe that the gains of using additional features than the downlink volume is rather low, but that there is still a small positive effect visible. The same applies to Moirai-1.1 and MOMENT. On the other hand, for the low-performing TimesFM in previous experiments, we can see that additional or other features significantly increase performance here, leading to performances comparable to the other models. Interestingly, TTM also gains a minor performance boost for the initial delay when utilizing the uplink volume only or both uplink and downlink volume as features. In this case, TTM achieves a macro F1-score of 0.91 for the initial delay (compared to 0.83 for downlink volume only). In general, the figure shows that the uplink volume is also a suitable feature to perform such a classification. Further, inter-arrival time features unexpectedly are of less relevance to the models and provide only marginal gains.

Summarizing, the results suggest that both the downlink volume and the uplink volume are indeed suitable and sufficient standalone features for the considered use case and for most TSFMs to achieve acceptable performance for most KPIs in the video streaming QoE scenario. Nevertheless, using additional features can boost TSFM performance.

5.4. Impact of traffic granularity

The granularity of time-series data is an important factor when it comes to deployment of efficient network monitoring. Fine granularities, e.g., below 1 s as required for real-time analytics [2], consume more hardware resources on one hand, but usually also provide better performance as more information is available on the other hand. Network monitoring with coarser granularities (above 1 s) is easier to deploy, but usually suffers from the fact that important information gets lost due to aggregation strategies. As a consequence, we analyze in this experiment how the performance of TSFMs is affected by different monitoring granularities. We reuse the baseline experiment with granularity 1 s and now evaluate additional granularities of 3 s, 5 s, and 10 s, to observe its impact on performance. We simulate the different granularities by aggregating the 1 s data to the targeted granularity and prepare the resulting time series for the TSFMs as before. Remember that the number of timesteps in the time series, i.e., the context lengths, are subsequently reduced, as depicted in Fig. 3(a). To meet the required context length of 512 time steps, the time series is padded by either prepending or appending NaNs to the original time series until the desired context length is reached. In this work, we exclusively append NaNs. For example, when resampling a 200 s video session at a granularity of 10 s, the resulting time series of length 20 requires appending $512 - 20 = 492$ NaNs. We intentionally use NaNs for padding as some TSFMs generate observation masks based on them. These models are able to ignore regions padded with NaNs. However, since padding reduces the amount of usable context for model prediction, it is still to be expected that increased padding generally leads to decreased model performance.

We show the results for selected model variants in Fig. 6. Each bar is hatched according to the used traffic granularity, and mean and standard deviation across the 30 folds are depicted. The baseline bar, representing a granularity of 1 s, is solely color-filled and does not contain any hatching. For the initial delay, for example, we can see that the TSFMs suffer slight performance losses for coarser granularities. The strength of this performance loss, however, strongly depends on the considered TSFM. While Chronos-T5 performs relatively similarly across all granularities, the performance of MOMENT starts to decrease strongly. Similar observations can be made for the other KPIs.

Summarizing, it is difficult to identify the best suitable granularity with respect to performance, for which to target in network monitoring. Instead, our results suggest that coarser granularities of 5 s to 10 s are

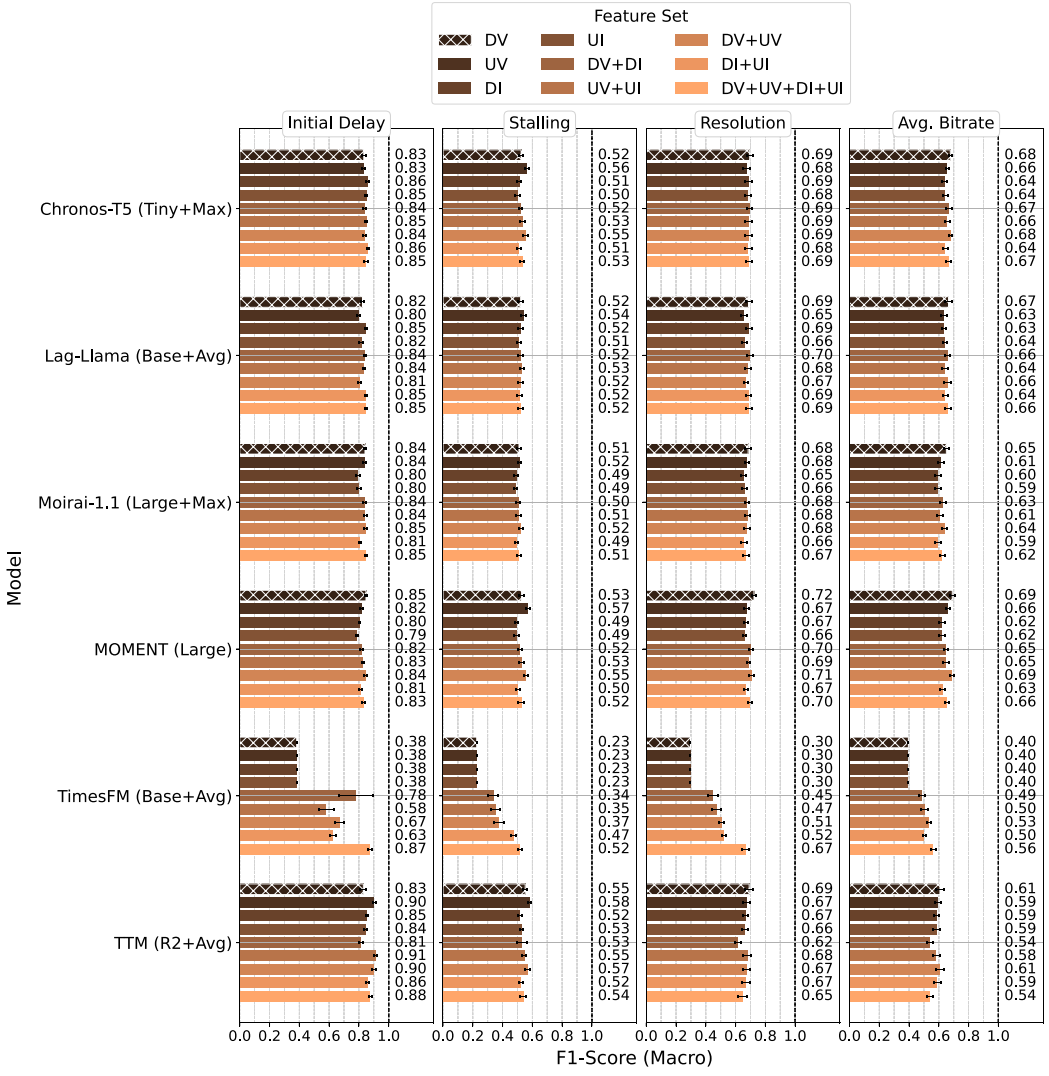


Fig. 5. Performance evaluation of feature set experiment.

still sufficient to obtain reasonable results, when selecting appropriate TSFMs. Note that in Section 6, we reconsider this finding after fine-tuning TTM for different granularities.

5.5. Impact of number of context samples

So far, we have focused on the data preparation in our experiments. Next, we want to shortly discuss some parameters of the zero-shot methodology. In previous experiments, we have always used 100 context samples per class. However, it is not yet clear how performance is affected when using less context samples. Remember that context samples are required to guide the TSFM into the right direction for getting an intuitive understanding of the considered classes. For the purpose of this experiment, we thus set the number of available context samples either to 5, 10, 20, 50, or 100. Note that this number corresponds to the number of context samples per class, i.e., for a context size of 100 and two classes we actually select 200 context samples.

The obtained results for varying numbers of context samples are displayed in Fig. 7. The x-axis depicts the number of context samples, while the y-axis denotes the macro F1-score. Each marker, styled according to the used TSFM, denotes mean macro F1-score with standard deviation as error bars across the 30 folds. The major observation from the figure is that an increasing number of context samples leads to more stable macro F1-scores. With only 5 to 20 context samples, the error bars are easily visible. The standard deviations range from 0.03 to 0.07 for the initial delay, 0.02 to 0.07 for stalling, 0.05 to 0.11 for resolution, and 0.06 to 0.09 for the average bitrate, and are thus rather high. As we increase up to 100 context samples, the error bars are basically no longer visible for most TSFMs. Chronos-T5 shows a particularly robust behavior, as the standard deviation is high only when the number of context samples is 10 or lower. Summarizing, fewer context samples lead to higher variations in the results, but can be sufficient to approximate the true performance, in particular, when evaluating multiple folds. To obtain accurate results, though, a higher number of context samples is required. Our results suggest that 50 context samples per class may be sufficient.

5.6. Impact of measurement points

So far, we have not differentiated between video sessions originating from Würzburg and Zagreb, even though we know from previous work that these sessions differ in terms of network behavior [1]. In these experiments, we thus analyze the transfer learning performance of TSFMs by randomly selecting context samples either from the Würzburg measurement point only, from the Zagreb measurement point only, or from both, and then evaluate the TSFMs' zero-shot performance for all remaining data, while distinguishing between the measurement points.

Fig. 8 summarizes the results for this experiment. The x-axis denotes the macro F1-score, while the y-axis denotes the used TSFM.

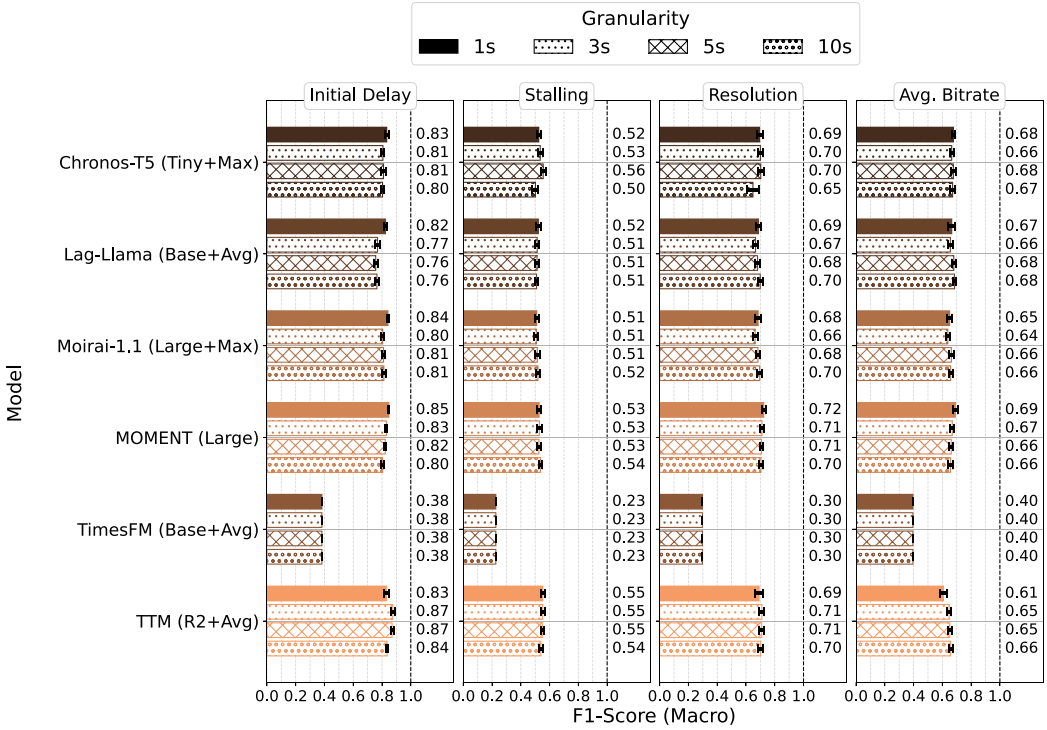


Fig. 6. Performance evaluation of granularity experiment.

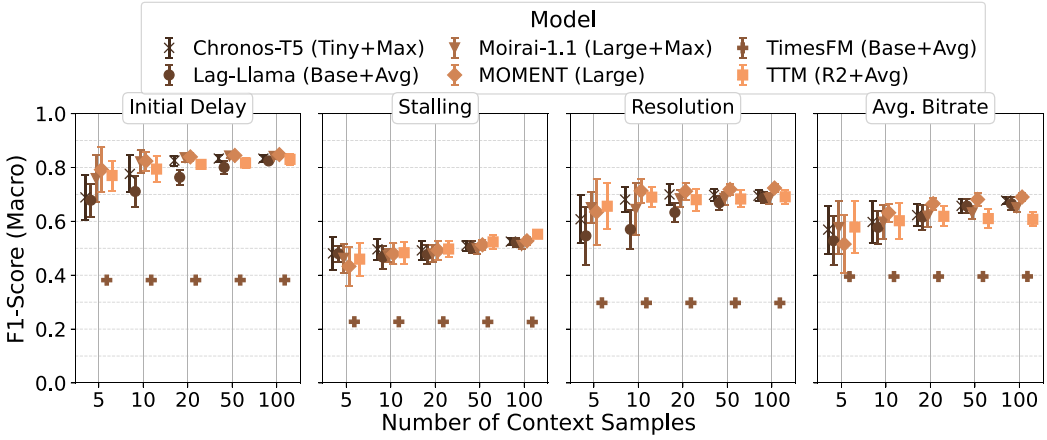


Fig. 7. Performance evaluation of number of context samples experiment.

Each horizontal bar again denotes mean macro F1-score and standard deviation across the 30 folds, and each bar is colored according to the used test data, where all data is black, Würzburg is brown, and Zagreb is orange. Hatched bars correspond to context samples drawn from all data, filled bars to context samples drawn from Würzburg data only, and empty bars to context samples drawn from Zagreb data only. Interestingly, for the initial delay, stalling, and the average bitrate the differences in performance between the measurement points are rather small, indicating that no strong data drifts are present in the data. For the resolution, however, differences between the measurement points can be observed. Surprisingly, the TSFM performance is much lower when evaluating Würzburg data only (macro F1-score of approx. 0.60), independent of the used context samples, compared to the Zagreb test data (macro F1-score of approx. 0.80).

Our results indicate that feature drifts are less prevalent for most KPIs when using time-series data and TSFMs. One reason for this is likely closely related to the general performance differences between the TSFMs and related work, as discussed previously (cf. Fig. 4). Some of the features and statistical descriptors used for training the tabular session model seem to be required to obtain high performance for all KPIs, but are explicitly not present in our time-series data, such that the TSFMs cannot *overfit* on these aspects of the data (and lose generalizability across different datasets), leading to similar performance across KPIs and measurement points. As a consequence, using (univariate) time-series data, instead of tabular data, seems to increase generalizability across measurement points. Nevertheless, we could also observe minor performance differences for the resolution between measurement points, suggesting that small drifts are still present.

5.7. Inference times

Last but not least, we shortly want to discuss TSFM inference times. Low inference times play a fundamental role in the practical deployment of such models. Therefore, we reuse the baseline experiment and measure the time it takes to infer 1000 samples for each model with a batch size of 1. First, we conduct experiments on a CPU (Apple M1

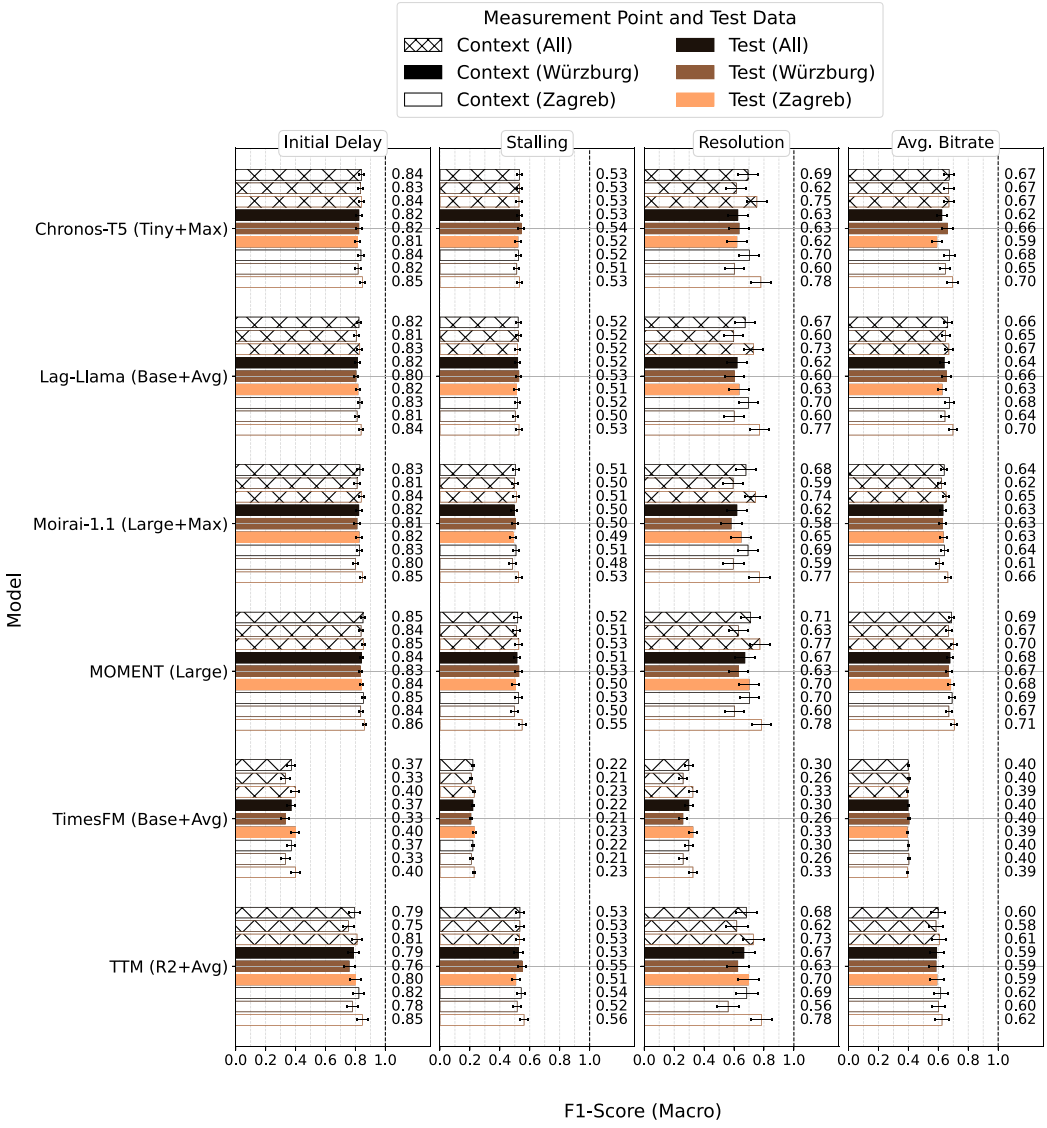


Fig. 8. Performance evaluation of zero-shot transfer learning experiment.

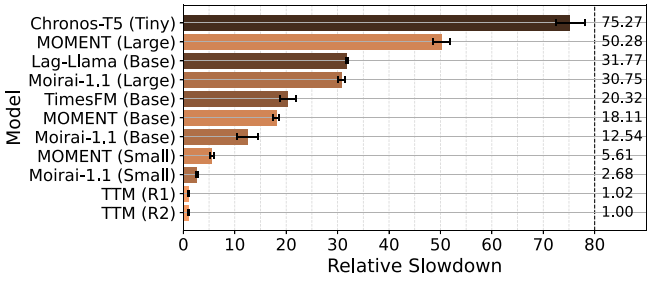


Fig. 9. Performance evaluation of inference times experiment.

Chip, 8 cores), repeating each run 30 times. This allows to assess the feasibility of running TTM on commodity hardware, which offers a cheaper alternative to dedicated GPUs.

Fig. 9 depicts the mean relative slowdown and standard deviation of all models normalized relative to the fastest model TTM (R2), with a mean inference time of 4.13 s. The x-axis depicts the normalized relative slowdown and the y-axis lists the models along with their sizes. The models are already sorted in an ascending fashion according to the achieved inference times and thus slowdowns. TTM (R1 & R2) exhibited by far the lowest inference times, followed by Moirai-1.1 (Small) with a relative slowdown of 2.68 and MOMENT (Small) with a relative slowdown of 5.61, i.e., the models take 2.68 and 5.61 times longer for predictions than TTM. In contrast, even for the Tiny version of Chronos-T5 a slowdown of 75 could be observed, making the model less usable in practice. Considering both performance and inference times, we argue that TTM is thus the best suited model for network monitoring, especially for reactive network management.

To assess the suitability of TTM for real-time network monitoring, we measure inference times on a single GPU (NVIDIA GeForce RTX 2080 TI). We infer 100,000 samples across various batch sizes (1024, 2048, 4096, 8192, 16 384), repeating each configuration over 30 iterations to maximize GPU utilization. Under these conditions, TTM achieves an average throughput of approximately 67,000 session predictions per second. However, real-time constraints typically require keeping the end-to-end monitoring delay below 1 s. This limits the usable inference window to a fraction of 1 s, e.g., 0.4 s–0.6 s for systems like Marina [2], resulting in an effective throughput of 27,000 to 40,000 predictions per second for TTM. While this is considerably lower than the 524,000 sessions per second reported for Marina, it remains within an acceptable range for real-time monitoring use cases. The size of the inference window and thus the inference throughput is primarily constrained by data access latency, which depends on the

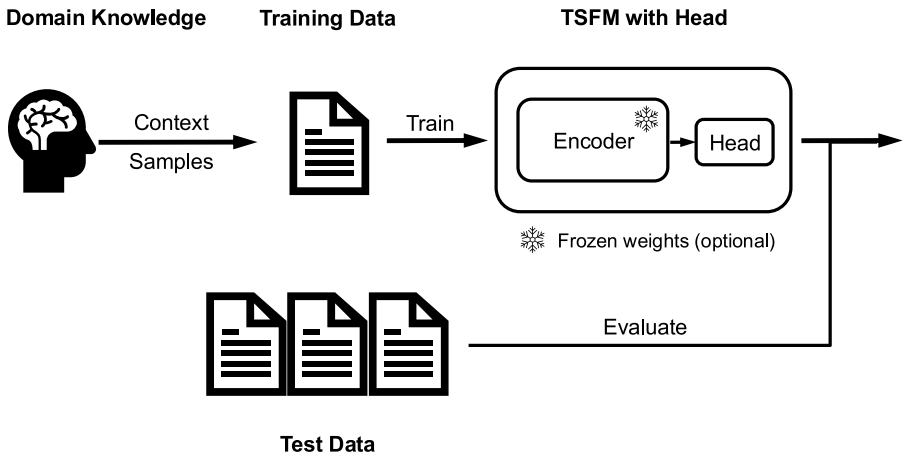


Fig. 10. Few-shot learning with Tiny Time Mixers (TTM).

monitoring hardware, and network transmission delays, which depend on the network stack. Performance can thus be further improved by adopting newer technologies such as field programmable gate arrays (FPGAs), data processing units (DPU) and smart network interface cards (SmartNICs) [112]. Additionally, optimizing the network stack with low-latency transport protocols such as RDMA over Converged Ethernet (RoCE) [113], which has not been implemented in Marina, could reduce transmission delays. Finally, inference scalability can be improved by leveraging multi-GPU setups and optimizing resource orchestration across the system.

6. Few-shot learning

In the following, we analyze whether fine-tuning a TSFM yields additional benefits, in particular, in the few-shot learning (FSL) setting. With FSL, a model is fine-tuned on a small number of training examples for a particular, but novel task. Remember that the TSFMs have been designed for time-series forecasting and that we fine-tune them now on a classification downstream task. Fine-tuning generally refers to the task of adapting the model weights of an existing pre-trained model during training such that its performance for a specific downstream task is improved. For our few-shot experiment, we restrict our analysis on TTM (R2), as we could see in the previous evaluations that the model is extraordinarily fast and simultaneously offers an acceptable performance compared to other models.

We provide an overview of our few-shot methodology in this work in Fig. 10. Similar to the zero-shot setting, we first require context samples based on domain knowledge, i.e., representative training samples for the two classes, our few-shots. Note that, if applicable, we reuse the same context samples as in the zero-shot setting for training. As we operate in the few-shot setting, we only collect small amounts of context samples, i.e., 5, 10, 20, 50, 100, 200, and 500 samples per class. We also use large sample sizes of 200, and 500 here to see if there are any additional benefits when using many samples, even though it theoretically can no longer be considered FSL. All other samples comprise our test dataset, which is used to quantify the performance of the fine-tuned models. Before fine-tuning TTM, it is possible to freeze the weights of the backbone encoder for training, i.e., they become untrainable and remain as they are, while the weights of the head are adjusted. We evaluate both scenarios, the one, in which we freeze the backbone encoder and only the head is fine-tuned, and the one, in which the entire model is fine-tuned. Additionally, we replace the original head for the forecasting downstream task with a new head designed for our classification downstream task. The head of TTM consists of a light TSMixer-based decoder block and a linear layer, which projects the decoded representation to the forecast. To adapt it to the classification downstream task, we initialize a new TSMixer decoder block, add a Dropout layer for regularization, and finally a linear layer with a single neuron. As with the forecasting head, the output of the decoder block is flattened, before the linear layer projects this input to the final logits, used to derive the prediction. Negative logits correspond to a prediction of class 0, while positive logits including zero correspond to a prediction of class 1. As predictions can now be directly inferred from the model, an additional supervised classifier like KNN is no longer required.

6.1. Baseline

For all our experiments, we fine-tune the model with the AdamW optimizer and test different learning rates (1e−2, 1e−3, 1e−4, 1e−5) for a different number of epochs (1, 3, 5, 10, 30, 50, 100). We choose a batch size of 1024 such that in each training round all training samples (context samples or shots) are simultaneously fed to the model. As a consequence, the number of epochs also corresponds to the number of training steps. To optimize the model, we then use the binary cross-entropy loss based on the obtained logits.

In the baseline experiment, we fine-tune a model for each parameter combination. Except for the number of training epochs (which is fixed to 100 here), Fig. 11 provides an overview of the obtained results, where each subfigure shows the results for a specific KPI and learning rate, and where the x-axis always denotes the number of training samples (two times the number of context samples) and the y-axis always quantifies performance in terms of macro F1-score. The results for head only and complete model fine-tuning are illustrated with different marker shapes and colors. Additionally, the dotted dark blue line indicates the baseline performance of the zero-shot setting for the respective KPIs.

Considering the learning rate, we can quickly observe that lower learning rates like 1e−04 and 1e−05 resulted in lower performance, and that especially for a learning rate of 1e−02 the best performance was obtained. Subsequently, we focus only on this learning rate in the following. Further, the figure displays that a higher number of training samples results in higher performance, e.g., the average bitrate and resolution only start to improve upon the baseline when more than 200 training samples are used. On the other hand, stalling and initial delay require less training samples (around 40) to improve upon the baseline, suggesting that fine-tuning is more effective for these KPIs.

Even though the model has been fine-tuned now and performance has improved for all KPIs, the stalling detection of TTM performs still worse than window-based approaches [2,33]. As a consequence, we assume that stalling detection from pure time-series data is impractical and highly difficult. This finding is also kind of intuitive, as a time-series model cannot relate specific points in a time series with stalling events. These events are not directly visible in the time-series data and

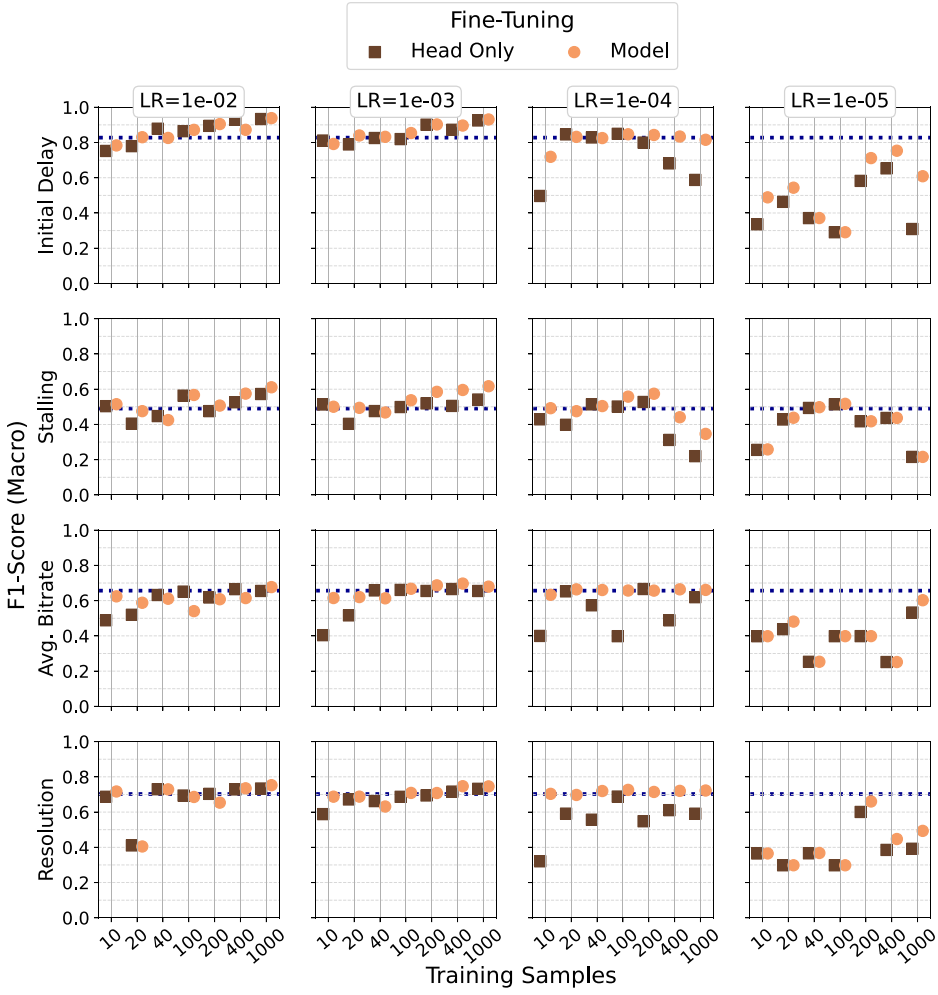


Fig. 11. Performance evaluation of baseline fine-tuning experiment.

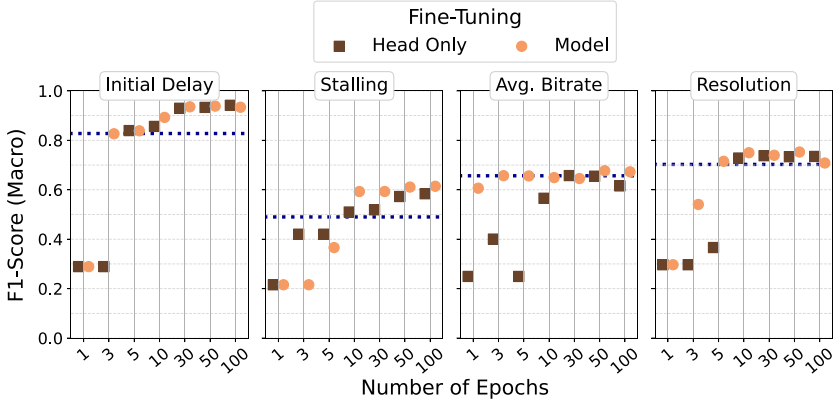


Fig. 12. Performance evaluation of number of epochs fine-tuning experiment.

are actually caused by the application's video player buffer. To make such a stalling detection approach feasible, it would be necessary to label specific points in a time series as indicators for stalling such that a TSFM can learn to identify stalling patterns. However, we leave this open for future work.

6.2. Impact of number of epochs

As we have not considered the number of training epochs or steps in the previous experiment, we analyze its impact on the fine-tuned model's performance in the following. We keep the learning rate fixed to 1e−02 and the number of training samples fixed to 1000.

The impact of the number of epochs on the performance is shown in Fig. 12. The x-axis denotes the number of epochs from 1 to 100 and the y-axis denotes the macro F1-score. Again, the markers differentiate between fine-tuning the head only (brown) and fine-tuning the entire model (orange). The results indicate that performance starts to improve only with a higher number of training epochs (30+) and that only a few epochs (1–5) are not sufficient to obtain better results than the dotted dark blue baseline from the zero-shot setting. Subsequently, training

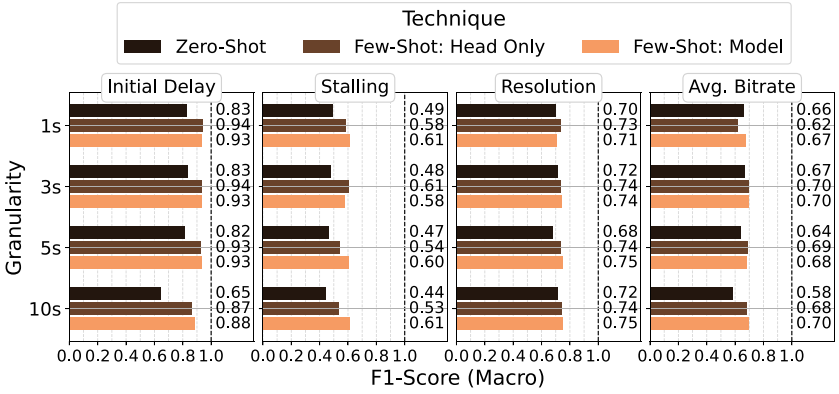
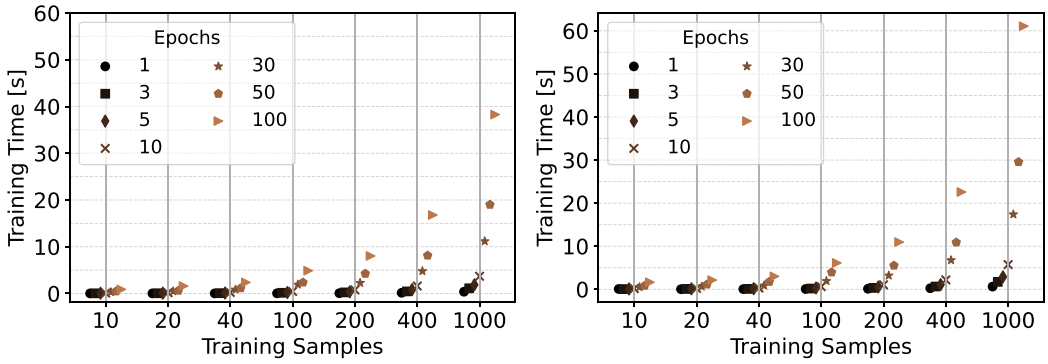


Fig. 13. Performance evaluation of granularity fine-tuning experiment.



(a) Fine-tuning of model head only.

(b) Fine-tuning of entire model.

Fig. 14. Performance evaluation of training durations.

times may affect the deployment of such fine-tuning approach, which we investigate later.

6.3. Impact of granularity

In the zero-shot setting, we have observed strong performance decreases for TTM when confronted with coarser monitoring granularities. In this experiment, we fine-tune the model on each of the different granularities and compare the resulting performance to the zero-shot performance. We use 100 epochs, 500 context samples, and a learning rate of 1e−02 as these parameters yielded the best results for a granularity of 1 s.

The obtained results are depicted in Fig. 13, where the x-axis denotes the macro F1-score, the y-axis denotes the traffic granularity of 1 s, 3 s, 5 s, or 10 s, and each horizontal bar denotes the mean performance for the different experiment settings and KPIs. We show the respective zero-shot results in black as baseline, and the brown and orange bars depict fine-tuning of the model head only and the entire model, respectively. Foremost, the figure shows significant performance gains for all granularities and KPIs compared to the zero-shot setting. Further, we observe only minor differences between fine-tuning the model head only and the entire model. Nevertheless, even in the fine-tuning setting, we can observe minor performance losses for higher granularities, e.g., a macro F1-score of 0.87 for the initial delay and a granularity of 10 s as compared to approx. 0.95 for a granularity of 1 s. Similar, but less severe observations can be made for the other KPIs. As a result, it depends on the network operator to trade-off the prediction accuracy and monitoring granularity. Nevertheless, our results indicate that even with higher granularities good performance can be achieved when additionally fine-tuning the TSFM.

6.4. Training times

Finally, we analyze the fine-tuning training duration in dependency of the number of training epochs/steps and the number of training samples. In practice, low training times are desirable for network operators to enable them to quickly adapt their monitoring system to new requirements.

Fig. 14 depicts the observed training durations for the fine-tuning of TTM for the head only (cf. Fig. 14(a)) and for the entire model (cf. Fig. 14(b)). The x-axis denotes the number of overall training samples, i.e., two times the number of context samples, and the y-axis denotes the training time in seconds. The marker shape and color denote the number of training epochs. The figure shows the highest observed training time of around 60 s for 1000 training samples and 100 training epochs, when fine-tuning the entire model (cf. Fig. 14(b)). In contrast, when freezing the backbone encoder weights, we observe a training time of around 40 s, and thus, a 33% training time reduction. As noted earlier, the performance gap between head-only fine-tuning and full-model fine-tuning is only marginal. Both aspects thus suggest to fine-tune the head only to save resources, even though the training times are generally very low in this case. Note that the increase of training time can be modeled with a linear function in dependency of the number of training samples, and can thus be easily interpolated for more training samples. Summarizing, we note that a TSFM like TTM can be quickly fine-tuned while simultaneously increasing performance.

7. Discussion

Our evaluation demonstrated that acceptable performance can be achieved in zero-shot settings for most KPIs, even when leveraging only the downlink volume as a feature. This confirms that simple NetFlow

features, already available in most network monitoring systems, are sufficient for a TSFM to capture the patterns necessary to perform adequate predictions. Achieving such performance, however, relies significantly on a global scaler with excellent knowledge of the representation space, especially for video-quality-related KPIs. Carefully curating representative samples from the data used to derive such a scaler is thus very important in practice. Once a perfect scaler is obtained, it is sufficient to randomly sample context samples from the data, thereby simplifying data preparation efforts. In general, it can be stated that the performance improves the more context samples are used. Moreover, lower temporal monitoring granularities lead to better model performance as less information content is lost in the data.

Aside from the evaluation of the impact of technical model parameters, we have observed that time-series modeling can prove advantageous in managing feature drifts across different networks, a common challenge in network traffic analysis. TSFMs, however, achieve robust performance by capturing temporal patterns, effectively handling the inherent variability present in encrypted network traffic. Additionally, low inference times for models such as Tiny Time Mixers (TTM), Moirai, and MOMENT indicate their practicality in real-world settings, even feasible for real-time deployments.

Our findings have several implications for network monitoring, in particular, and network management, in general. The feasibility of zero-shot inference with lightweight and easy to measure features enables rapid deployment of monitoring systems. Optional fine-tuning can yield significant additional benefits, as shown for TTM. The quick fine-tuning of this model, e.g., 100 epochs with 1000 samples in less than one minute, makes the model highly suitable for situations where rapid adaptation to specific use cases or settings is required. Furthermore, we have observed that fine-tuning can even yield high model performance for coarser monitoring granularities, e.g., 10 s. This is especially interesting for network operators aiming to reduce monitoring efforts by sampling at larger intervals while maintaining precise predictions. Even though we focused on video session health, we emphasize that the methodologies presented in this work are easily transferable to other use cases, e.g., intrusion detection or fingerprinting, as only the input time-series data has to be substituted.

When comparing the zero-shot and few-shot performance of TSFMs in this work to the lightweight Random Forests from related work [1], we observed that TSFMs performed significantly worse in the zero-shot setting. However, fine-tuning TSFMs significantly narrowed the performance gap. As previously noted, the results are not directly comparable. In this work, TSFMs were limited to a small set of input features (at most four), derived from time-series data readily available via telemetry systems like NetFlow, commonly deployed in today's networks. However, TSFMs are expensive to deploy as they usually require dedicated GPUs and inference times are significantly higher compared to lightweight models. In contrast, the Random Forests in related work considered 207 input features in the form of tabular data, covering a variety of different network traffic statistics, which enabled more accurate predictions of video session health. However, obtaining such extensive feature sets (in real-time) typically requires specialized telemetry systems and thus expensive hardware, e.g., programmable switches capable of line-rate statistics extraction and high engineering efforts. This highlights a fundamental trade-off between feature engineering and telemetry complexity, and model deployment costs. Using complex, extensive feature sets with lightweight models shifts the burden to feature engineering and telemetry infrastructure. Conversely, relying on simple features with more complex models shifts the complexity to the models themselves, which must learn to perform well given the limited input.

We also stress the fact that current TSFMs are not explicitly designed for network traffic. This presents an opportunity for future research, as the development of TSFMs tailored to encrypted network traffic analytics could result in further zero-shot performance improvements, even in the presence of coarse monitoring granularities and shorter context lengths. Such zero-shot networking-specific models would advance data-driven network management significantly, while also strongly reducing monitoring costs. While current TSFMs are well-suited for real-time network monitoring at moderate scale, their relatively high inference times compared to lightweight models limit their scalability for monitoring hundreds of thousands of sessions concurrently. Scaling to such levels would require infrastructure investments, e.g., multiple GPUs, advanced resource orchestration, and optimized hardware and network stacks, which in turn increases overall complexity and deployment costs. Nonetheless, future technological advances and the development of TSFMs tailored specifically for networking are expected to improve their scalability and cost-efficiency for large-scale deployment.

In summary, the combination of zero-shot capabilities, rapid and effective fine-tuning, and swift inference underscores the practicality of most TSFMs for network management. Our work, thus, demonstrated the feasibility of deploying TSFMs in both zero-shot and few-shot settings in practice.

8. Conclusion

In this work, we explored the use of Time Series Foundation Models (TSFMs) for network monitoring tasks. TSFMs are models pre-trained on large-scale time-series data, enabling them to perform well in ZSL and FSL scenarios across diverse downstream tasks. To assess their potential for network monitoring, we focused on the specific challenge of estimating video streaming session QoE from encrypted network traffic. We first evaluated the zero-shot capabilities of TSFMs, assessing their ability to estimate session health without any prior task-specific training. We then studied their performance when fine-tuned on a limited amount of training data. Additionally, we investigated relevant model and monitoring parameters, including the temporal monitoring granularity, the applied data scaler, the used features, and the size of the context and training data.

Our analysis revealed that TSFMs are indeed ready for practical deployments and that they perform acceptable in a zero-shot setting, while most models also provide low inference times. Our fine-tuning experiments with Tiny Time Mixers emphasized the fact that only few samples are sufficient for training to improve performance significantly, even for monitoring granularities beyond 5 s, while keeping training times low.

The considered TSFMs were all pretrained on data from different domains, such as weather, finance, energy, and sensors [3]. These data captured various time series lengths, sampling frequencies (ranging from seconds to years), trends, and seasonal patterns. However, network traffic data has not been included in the pretraining data of those models, but usually exhibits unique characteristics, such as the on-off burst patterns seen in video streaming [114,115]. Furthermore, network traffic often operates at a millisecond granularity, a resolution not covered by the domains used for pretraining. Despite this mismatch, the benchmarked TSFMs demonstrated strong few-shot performance on network tasks. This, however, also highlights the opportunity of developing TSFMs specifically pretrained on network monitoring data in the future. Such TSFMs may yield significant improvements in zero-shot performance, and thus decrease operational costs for network operators.

CRediT authorship contribution statement

Nikolas Wehner: Writing – review & editing, Writing – original draft, Visualization, Validation, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Pedro Casas:** Writing – review & editing, Writing – original draft, Validation, Methodology, Data curation, Conceptualization. **Katharina Dietz:** Writing – review & editing, Writing – original draft, Validation,

Methodology, Conceptualization. **Stefan Geißler:** Writing – review & editing, Writing – original draft, Validation, Conceptualization. **Tobias Hoßfeld:** Writing – review & editing, Validation, Supervision, Methodology, Conceptualization. **Michael Seufert:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Data curation, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used ChatGPT in order to improve language and readability. After using this tool, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partly funded by Deutsche Forschungsgemeinschaft (DFG), Germany under grant SE 3163/3-1, project number: 500105691. The authors alone are responsible for the content.

Data availability

The data that has been used is confidential.

References

[1] M. Seufert, I. Orsolic, Improving the transfer of machine learning-based video QoE estimation across diverse networks, IEEE Trans. Netw. Serv. Manag. (2023).

[2] M. Seufert, K. Dietz, N. Wehner, S. Geißler, J. Schüler, M. Wolz, A. Hotho, P. Casas, T. Hoßfeld, A. Feldmann, Marina: Realizing ML-driven real-time network traffic monitoring at terabit scale, IEEE Trans. Netw. Serv. Manag. (2024).

[3] Y. Liang, H. Wen, Y. Nie, Y. Jiang, M. Jin, D. Song, S. Pan, Q. Wen, Foundation models for time series analysis: A tutorial and survey, in: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2024, pp. 6555–6565.

[4] S. Rahman, S. Khan, F. Porikli, A unified approach for conventional zero-shot, generalized zero-shot, and few-shot learning, IEEE Trans. Image Process. 27 (11) (2018) 5652–5667.

[5] A.F. Ansari, L. Stella, C. Turkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S.S. Rangapuram, S.P. Arango, S. Kapoor, et al., Chronos: Learning the language of time series, 2024, arXiv preprint arXiv:2403.07815.

[6] K. Rasul, A. Ashok, A.R. Williams, A. Khorasani, G. Adamopoulos, R. Bhagwatkar, M. Biloš, H. Ghonia, N.V. Hassen, A. Schneider, et al., Lag-llama: Towards foundation models for time series forecasting, 2023, arXiv preprint arXiv:2310.08278.

[7] A. Das, W. Kong, R. Sen, Y. Zhou, A decoder-only foundation model for time-series forecasting, 2023, arXiv preprint arXiv:2310.10688.

[8] G. Woo, C. Liu, A. Kumar, C. Xiong, S. Savarese, D. Sahoo, Unified training of universal time series forecasting transformers, 2024, arXiv preprint arXiv:2402.02592.

[9] M. Goswami, K. Szafer, A. Choudhry, Y. Cai, S. Li, A. Dubrawski, Moment: A family of open time-series foundation models, 2024, arXiv preprint arXiv:2402.03885.

[10] V. Ekambaram, A. Jati, N.H. Nguyen, P. Dayama, C. Reddy, W.M. Gifford, J. Kalagnanam, TTMs: Fast multi-level tiny time mixers for improved zero-shot and few-shot forecasting of multivariate time series, 2024, arXiv preprint arXiv:2401.03955.

[11] A. Garza, M. Mergenthaler-Canseco, TimeGPT-1, 2023, arXiv:2310.03589.

[12] H. Xue, F.D. Salim, PromptCast: A new prompt-based learning paradigm for time series forecasting, IEEE Trans. Knowl. Data Eng. (2023) 1–14, http://dx.doi.org/10.1109/TKDE.2023.3342137.

[13] N. Gruver, M. Finzi, S. Qiu, A.G. Wilson, Large language models are zero-shot time series forecasters, 2023, arXiv:2310.07820.

[14] A. Das, W. Kong, R. Sen, Y. Zhou, A decoder-only foundation model for time-series forecasting, 2024, arXiv:2310.10688.

[15] K. Rasul, A. Ashok, A.R. Williams, H. Ghonia, R. Bhagwatkar, A. Khorasani, M.J.D. Bayazi, G. Adamopoulos, R. Riachi, N. Hassen, M. Biloš, S. Garg, A. Schneider, N. Chapados, A. Drouin, V. Zantedeschi, Y. Nevmyvaka, I. Rish, Lag-llama: Towards foundation models for probabilistic time series forecasting, 2024, arXiv:2310.08278.

[16] M. Jin, S. Wang, L. Ma, Z. Chu, J.Y. Zhang, X. Shi, P.-Y. Chen, Y. Liang, Y.-F. Li, S. Pan, Q. Wen, Time-LLM: Time series forecasting by reprogramming large language models, in: International Conference on Learning Representations, ICLR, 2024.

[17] G.G. González, P. Casas, E. Martínez, A. Fernández, On the quest for foundation generative-AI models for anomaly detection in time-series data, in: 2024 IEEE European Symposium on Security and Privacy Workshops, 2024, pp. 252–260, http://dx.doi.org/10.1109/EuroSPW61312.2024.00034.

[18] P. Casas, M. Seufert, R. Schatz, YOUQMON: A system for on-line monitoring of YouTube QoE in operational 3G networks, ACM SIGMETRICS Perform. Eval. Rev. 41 (2) (2013) 44–46.

[19] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, K. Papagiannaki, Measuring video QoE from encrypted traffic, in: Internet Measurement Conference, 2016, pp. 513–526.

[20] I. Orsolic, D. Pevec, M. Suznjevic, L. Skorin-Kapov, YouTube QoE estimation based on the analysis of encrypted network traffic using machine learning, in: 2016 IEEE Globecom Workshops, IEEE, 2016, pp. 1–6.

[21] I. Orsolic, D. Pevec, M. Suznjevic, L. Skorin-Kapov, A machine learning approach to classifying YouTube QoE based on encrypted network traffic, Multimed. Tools Appl. 76 (21) (2017) 22267–22301.

[22] I. Orsolic, M. Suznjevic, L. Skorin-Kapov, YouTube QoE estimation from encrypted traffic: Comparison of test methodologies and machine learning based models, in: 10th International Conference on Quality of Multimedia Experience, QoMEX, IEEE, 2018, pp. 1–6.

[23] P. Schmitt, F. Bronzino, R. Teixeira, T. Chattopadhyay, N. Feamster, Enhancing Transparency: Internet Video Quality Inference from Network Traffic, TPRC, 2018.

[24] P. Casas, A. D'Alconzo, F. Wamser, M. Seufert, B. Gardlo, A. Schwind, P. Tran-Gia, R. Schatz, Predicting QoE in cellular networks using machine learning and in-smartphone measurements, in: 9th International Conference on Quality of Multimedia Experience, QoMEX, Erfurt, Germany, 2017.

[25] M.H. Mazhar, Z. Shafiq, Real-time video quality of experience monitoring for HTTPS and QUIC, in: INFOCOM 2018 - Conference on Computer Communications, IEEE, 2018, pp. 1331–1339.

[26] M. Seufert, P. Casas, N. Wehner, L. Gang, K. Li, Stream-based machine learning for real-time QoE analysis of encrypted video streaming traffic, in: 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN, IEEE, 2019, pp. 76–81.

[27] M. Seufert, P. Casas, N. Wehner, L. Gang, K. Li, Features that matter: Feature selection for on-line stalling prediction in encrypted video streaming, in: Conference on Computer Communications Workshops, INFOCOM WKSHPS, IEEE, 2019, pp. 688–695.

[28] C. Gutterman, K. Guo, S. Arora, X. Wang, L. Wu, E. Katz-Bassett, G. Zussman, Requet: Real-time QoE detection for encrypted YouTube traffic, in: 10th ACM Multimedia Systems Conference, 2019, pp. 48–59.

[29] S. Wassermann, M. Seufert, P. Casas, L. Gang, K. Li, Let me decrypt your beauty: Real-time prediction of video resolution and bitrate for encrypted video streaming, in: Network Traffic Measurement and Analysis Conference, TMA, IEEE, 2019, pp. 199–200.

[30] S. Wassermann, M. Seufert, P. Casas, L. Gang, K. Li, I see what you see: Real time prediction of video quality from encrypted streaming traffic, in: 4th Internet-QoE Workshop on QoE-Based Analysis and Management of Data Communication Networks, 2019, pp. 1–6.

[31] F. Bronzino, P. Schmitt, S. Ayoubi, G. Martins, R. Teixeira, N. Feamster, Inferring streaming video quality from encrypted traffic: Practical models and deployment experience, Meas. Anal. Comput. Syst. 3 (3) (2019) 1–25.

[32] I. Orsolic, L. Skorin-Kapov, A framework for in-network QoE monitoring of encrypted video streaming, IEEE Access 8 (2020) 74691–74706, http://dx.doi.org/10.1109/ACCESS.2020.2988735.

[33] S. Wassermann, M. Seufert, P. Casas, L. Gang, K. Li, Vicrypt to the rescue: Real-time, machine-learning-driven video-qoe monitoring for encrypted streaming traffic, IEEE Trans. Netw. Serv. Manag. 17 (4) (2020) 2007–2023.

[34] F. Loh, F. Poignée, F. Wamser, F. Leidinger, T. Hoßfeld, Uplink vs. downlink: Machine learning-based quality prediction for http adaptive video streaming, Sensors 21 (12) (2021) 4172.

[35] P. Casas, S. Wassermann, M. Seufert, N. Wehner, O. Dinica, T. Hoßfeld, X-Ray goggles for the ISP: Improving in-network web and app qoe monitoring with deep learning, in: V. Bajpai, H. Haddadi, O. Hohlfeld (Eds.), 6th Network Traffic Measurement and Analysis Conference, TMA 2022, June 27-30, 2022, IFIP, 2022.

[36] S. Schwarzmann, C. Cassales Marquezan, M. Bosk, H. Liu, R. Trivisonno, T. Zinner, Estimating video streaming QoE in the 5G architecture using machine learning, in: 4th Internet-QoE Workshop on QoE-Based Analysis and Management of Data Communication Networks, 2019, pp. 7–12.

[37] S. Schwarzmann, C.C. Marquezan, R. Trivisonno, S. Nakajima, T. Zinner, Accuracy vs. Cost trade-off for machine learning based QoE estimation in 5G networks, in: International Conference on Communications, ICC, IEEE, 2020, pp. 1–6.

[38] S. Schwarzmann, C.C. Marquezan, R. Trivisonno, S. Nakajima, V. Barriac, T. Zinner, ML-based qoe estimation in 5g networks using different regression techniques, IEEE Trans. Netw. Serv. Manag. 19 (3) (2022) 3516–3532.

[39] N. Wehner, M. Ring, J. Schüler, A. Hotho, T. Hoßfeld, M. Seufert, On learning hierarchical embeddings from encrypted network traffic, in: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2022, pp. 1–7.

[40] J. Gama, P. Medas, G. Castillo, P. Rodrigues, Learning with drift detection, in: Brazilian Symposium on Artificial Intelligence, Springer, 2004, pp. 286–295.

[41] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, N.D. Lawrence, Dataset Shift in Machine Learning, The MIT Press, 2009.

[42] I. Orsolic, P. Rebernjak, M. Suznjevic, L. Skorin-Kapov, In-network QoE and KPI monitoring of mobile YouTube traffic: Insights for encrypted iOS flows, in: 14th International Conference on Network and Service Management, CNSM, IEEE, 2018, pp. 233–239.

[43] S. Ickin, K. Vandikas, F. Moradi, J. Taghia, W. Hu, Ensemble-based synthetic data synthesis for federated QoE modeling, in: 2020 6th IEEE Conference on Network Softwarization, NetSoft, 2020, pp. 72–76, http://dx.doi.org/10.1109/NetSoft48620.2020.9165379.

[44] S. Ickin, M. Fiedler, K. Vandikas, QoE modeling on split features with distributed deep learning, Network 1 (2) (2021) 165–190.

[45] S. Porcu, A. Floris, L. Atzori, CB-FL: Cluster-based federated learning applied to quality of experience modelling, in: 2022 16th International Conference on Signal-Image Technology & Internet-Based Systems, SITIS, IEEE, 2022, pp. 585–591.

[46] S. Porcu, A. Floris, L. Atzori, A clustered federated learning approach for estimating the quality of experience of web users, in: 2023 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops, ICASSPW, IEEE, 2023, pp. 1–5.

[47] I. Orsolic, M. Seufert, On machine learning based video QoE estimation across different networks, in: 2021 16th International Conference on Telecommunications, ConTEL, IEEE, 2021, pp. 62–69.

[48] M. Carvalho, D. Soares, D.F. Macedo, Transfer learning-based qoe estimation for different cloud gaming contexts, in: 2023 IEEE 9th International Conference on Network Softwarization, NetSoft, IEEE, 2023, pp. 71–79.

[49] G. Pang, C. Shen, L. Cao, A.V.D. Hengel, Deep learning for anomaly detection: A review, ACM Comput. Surv. 54 (2) (2021).

[50] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, vol. 27, 2014.

[51] S. Zavrak, M. Iskefiyeli, Anomaly-based intrusion detection from network flow features using variational autoencoder, IEEE Access 8 (2020) 108346–108358.

[52] H. Zenati, C.S. Foo, B. Lecouat, G. Manek, V.R. Chandrasekhar, Efficient GAN-based anomaly detection, 2018, arXiv preprint arXiv:1802.06222.

[53] R.-Q. Chen, G.-H. Shi, W. Zhao, C.-H. Liang, A joint model for IT operation series prediction and anomaly detection, Neurocomputing 448 (2021) 130–139.

[54] J. Donahue, P. Krähenbühl, T. Darrell, Adversarial feature learning, 2016, arXiv preprint arXiv:1605.09782.

[55] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, S.-K. Ng, MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks, in: International Conference on Artificial Neural Networks, Springer, 2019, pp. 703–716.

[56] A. Geiger, D. Liu, S. Alnegheimish, A. Cuesta-Infante, K. Veeramachaneni, TadGAN: Time series anomaly detection using generative adversarial networks, in: 2020 IEEE International Conference on Big Data, Big Data, IEEE, 2020, pp. 33–43.

[57] G. García González, P. Casas, A. Fernández, G. Gómez, On the usage of generative models for network anomaly detection in multivariate time-series, SIGMETRICS Perform. Eval. Rev. 48 (4) (2021) 49–52, http://dx.doi.org/10.1145/3466826.3466843.

[58] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, 2013, CoRR abs/1312.6114. arXiv:1312.6114. URL https://arxiv.org/abs/1312.6114.

[59] C. Doersch, Tutorial on variational autoencoders, 2016, arXiv preprint arXiv:1606.05908.

[60] D.P. Kingma, M. Welling, An introduction to variational autoencoders, 2019, arXiv preprint arXiv:1906.02691.

[61] F.P. Casale, A.V. Dalca, L. Saglietti, J. Listgarten, N. Fusi, Gaussian process prior variational autoencoders, in: Advances in Neural Information Processing Systems, 2018.

[62] L. Girin, F. Roche, T. Hueber, S. Leglaive, Notes on the use of variational autoencoders for speech and audio spectrogram modeling, in: DAFx 2019-22nd International Conference on Digital Audio Effects, 2019, pp. 1–8.

[63] V. Fortuin, D. Baranchuk, G. Rätsch, S. Mandt, GP-VAE: Deep probabilistic time series imputation, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 1651–1661.

[64] S. Ramchandran, G. Tikhonov, K. Kujanpää, M. Koskinen, H. Lähdesmäki, Longitudinal variational autoencoder, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2021, pp. 3898–3906.

[65] J. Bayer, C. Osendorfer, Learning stochastic recurrent networks, 2014, arXiv preprint arXiv:1411.7610.

[66] J. Chung, K. Kastner, L. Dinh, K. Goel, A.C. Courville, Y. Bengio, A recurrent latent variable model for sequential data, Adv. Neural Inf. Process. Syst. 28 (2015).

[67] S. Shabanian, D. Arpit, A. Trischler, Y. Bengio, Variational bi-LSTMs, 2017, arXiv preprint arXiv:1711.05717.

[68] Z. Yang, Z. Hu, R. Salakhutdinov, T. Berg-Kirkpatrick, Improved variational autoencoders for text modeling using dilated convolutions, in: International Conference on Machine Learning, PMLR, 2017, pp. 3881–3890.

[69] G. Lai, B. Li, G. Zheng, Y. Yang, Stochastic WaveNet: A generative latent variable model for sequential data, 2018, arXiv preprint arXiv:1806.06116.

[70] G. García González, S. Martinez Tagliafico, A. Fernández, G. Gómez, J. Acuña, P. Casas, One model to find them all – deep learning for multivariate time-series anomaly detection in mobile network data, IEEE Trans. Netw. Serv. Manag. (2023) http://dx.doi.org/10.1109/TNSM.2023.3340146, 1–1.

[71] G. García González, S. Martinez Tagliafico, A. Fernández, G. Gómez, J. Acuña, P. Casas, DC-VAE, fine-grained anomaly detection in multivariate time-series with dilated convolutions and variational auto encoders, in: 2022 IEEE European Symposium on Security and Privacy Workshops, 2022, pp. 287–293.

[72] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, vol. 30, 2017.

[73] B. Lim, S. Arik, N. Loeff, T. Pfister, Temporal fusion transformers for interpretable multi-horizon time series forecasting, Int. J. Forecast. 37 (4) (2021) 1748–1764.

[74] K.C. Chen, L. Dicker, C. Eisenach, D. Madeka, MQTransformer: Multi-horizon forecasts with context dependent attention and optimal bregman volatility, in: KDD 2022 Workshop on Mining and Learning from Time Series – Deep Forecasting: Models, Interpretability, and Applications, 2022.

[75] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, W. Zhang, Informer: Beyond efficient transformer for long sequence time-series forecasting, in: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI Press, 2021, pp. 11106–11115, http://dx.doi.org/10.1609/AAAI.V35I12.17325.

[76] H. Wu, J. Xu, J. Wang, M. Long, Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting, 2021, CoRR abs/2106.13008. arXiv:2106.13008. URL https://arxiv.org/abs/2106.13008.

[77] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, R. Jin, FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting, 2022, CoRR abs/2201.12740. arXiv:2201.12740. URL https://arxiv.org/abs/2201.12740.

[78] S. De, M. Bermudez-Edo, H. Xu, Z. Cai, Deep generative models in the industrial internet of things: a survey, IEEE Trans. Ind. Inform. 18 (9) (2022) 5728–5737.

[79] K. Dietz, M. Seufert, T. Hoßfeld, Want more WANs? Comparison of traditional and GAN-based generation of wide area network topologies via graph and performance metrics, IEEE Trans. Netw. Serv. Manag. (2023).

[80] A. Karapantelakis, P. Alizadeh, A. Alabassi, K. Dey, A. Nikou, Generative AI in mobile networks: a survey, Ann. Telecommun. 79 (1) (2024) 15–33.

[81] M. Ring, D. Schlör, D. Landes, A. Hotho, Flow-based network traffic generation using generative adversarial networks, Comput. Secur. 82 (2019) 156–172.

[82] Z. Lin, A. Jain, C. Wang, G. Fanti, V. Sekar, Generating high-fidelity, synthetic time series datasets with doppelganger, 2019, arXiv preprint arXiv:1909.13403.

[83] Y. Yin, Z. Lin, M. Jin, G. Fanti, V. Sekar, Practical gan-based synthetic ip header trace generation using netshare, in: Proceedings of the ACM SIGCOMM 2022 Conference, 2022, pp. 458–472.

[84] J. Ho, A. Jain, P. Abbeel, Denoising diffusion probabilistic models, Adv. Neural Inf. Process. Syst. 33 (2020) 6840–6851.

[85] S. Zhang, T. Li, D. Jin, Y. Li, NetDiff: A service-guided hierarchical diffusion model for network flow trace generation, Proc. ACM Netw. 2 (CoNEXT3) (2024) 1–21.

[86] N. Sivaroopan, D. Bandara, C. Madarasingha, G. Jourjon, A.P. Jayasumana, K. Thilakarathna, Netdiffus: Network traffic generation by diffusion models through time-series imaging, Comput. Netw. 251 (2024) 110616.

[87] X. Jiang, S. Liu, A. Gember-Jacobson, A.N. Bhagoji, P. Schmitt, F. Bronzino, N. Feamster, Netdiffusion: Network data augmentation through protocol-constrained traffic generation, Proc. ACM Meas. Anal. Comput. Syst. 8 (1) (2024) 1–32.

[88] D.K. Kholgh, P. Kostakos, PAC-GPT: A novel approach to generating synthetic network traffic with GPT-3, IEEE Access (2023).

[89] H.Y. He, Z.G. Yang, X.N. Chen, PERT: Payload encoding representation from transformer for encrypted traffic classification, in: 2020 ITU Kaleidoscope: Industry-Driven Digital Transformation, ITU K, IEEE, 2020, pp. 1–8.

[90] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, J. Yu, Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification, in: Proceedings of the ACM Web Conference 2022, 2022, pp. 633–642.

[91] X. Meng, C. Lin, Y. Wang, Y. Zhang, Netgpt: Generative pretrained transformer for network traffic, 2023, arXiv preprint arXiv:2304.09513.

[92] R. Zhao, M. Zhan, X. Deng, Y. Wang, Y. Wang, G. Gui, Z. Xue, Yet another traffic classifier: A masked autoencoder based traffic transformer with multi-level flow representation, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, 2023, pp. 5420–5427, 4.
[93] Z. Shi, N. Luktarhan, Y. Song, G. Tian, BFCN: A novel classification method of encrypted traffic based on BERT and CNN, Electronics 12 (3) (2023) 516.
[94] Z. Hang, Y. Lu, Y. Wang, Y. Xie, Flow-MAE: Leveraging masked AutoEncoder for accurate, efficient and robust malicious traffic classification, in: Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses, 2023, pp. 297–314.
[95] J. Devlin, Bert: Pre-training of deep bidirectional transformers for language understanding, 2018, arXiv preprint arXiv:1810.04805.
[96] D. Wu, X. Wang, Y. Qiao, Z. Wang, J. Jiang, S. Cui, F. Wang, NetLLM: Adapting large language models for networking, in: Proceedings of the ACM SIGCOMM 2024 Conference, 2024, pp. 661–678.
[97] E.J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, 2021, arXiv preprint arXiv:2106.09685.
[98] T. Wang, X. Xie, L. Zhang, C. Wang, L. Zhang, Y. Cui, ShieldGPT: An LLM-based framework for DDoS mitigation, in: Proceedings of the 8th Asia-Pacific Workshop on Networking, 2024, pp. 108–114.
[99] Q. Wang, C. Qian, X. Li, Z. Yao, G. Zhou, H. Shao, Lens: A foundation model for network traffic, 2024, arXiv preprint arXiv:2402.03646.
[100] S. Guthula, R. Beltiukov, N. Battula, W. Guo, A. Gupta, netFound: Foundation model for network security, 2023, arXiv preprint arXiv:2310.17025.
[101] R. Bommasani, D.A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M.S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al., On the opportunities and risks of foundation models, 2021, arXiv preprint arXiv:2108.07258.
[102] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F.L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., Gpt-4 technical report, 2023, arXiv preprint arXiv:2303.08774.
[103] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, et al., Sam 2: Segment anything in images and videos, 2024, arXiv preprint arXiv:2408.00714.
[104] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, I. Sutskever, Zero-shot text-to-image generation, in: International Conference on Machine Learning, PMLR, 2021, pp. 8821–8831.
[105] G. Mai, W. Huang, J. Sun, S. Song, D. Mishra, N. Liu, S. Gao, T. Liu, G. Cong, Y. Hu, et al., On the opportunities and challenges of foundation models for geospatial artificial intelligence, 2023, arXiv preprint arXiv:2304.06798.
[106] M. Moor, O. Banerjee, Z.S.H. Abad, H.M. Krumholz, J. Leskovec, E.J. Topol, P. Rajpurkar, Foundation models for generalist medical artificial intelligence, Nature 616 (7956) (2023) 259–265.
[107] A. Vaswani, Attention is all you need, Adv. Neural Inf. Process. Syst. (2017).
[108] S.-A. Chen, C.-L. Li, N. Yoder, S.O. Arik, T. Pfister, Tsmixer: An all-mlp architecture for time series forecasting, 2023, arXiv preprint arXiv:2303.06053.
[109] V. Ekambaram, A. Jati, N. Nguyen, P. Sinthong, J. Kalagnanam, Tsmixer: Lightweight mlp-mixer model for multivariate time series forecasting, in: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2023, pp. 459–469.
[110] M. Seufert, R. Schatz, N. Wehner, P. Casas, Quicker or not?-an empirical analysis of quic vs tcp for video streaming qoe provisioning, in: 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN, IEEE, 2019, pp. 7–12.
[111] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, P. Tran-Gia, A survey on quality of experience of HTTP adaptive streaming, IEEE Commun. Surv. Tutor. 17 (1) (2014) 469–492.
[112] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, N. Zilberman, In-network machine learning using programmable network devices: A survey, IEEE Commun. Surv. Tutor. 26 (2) (2023) 1171–1200.
[113] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, S. Shenker, Revisiting network support for RDMA, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 313–326.
[114] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, M. Watson, A buffer-based approach to rate adaptation: Evidence from a large video streaming service, in: Proceedings of the 2014 ACM Conference on SIGCOMM, 2014, pp. 187–198.
[115] N. Wehner, T. Karagioules, E. Halepovic, F. Simonovski, T. Hossfeld, M. Seufert, To cap or not to cap: Bandwidth capping effects on network interactions and QoE of competing short video streams, in: Proceedings of the 16th ACM Multimedia Systems Conference, 2025, pp. 90–100.