ABSTRACT
Generalizing machine learning (ML) models for network traffic dynamics tends to be considered a lost cause. Hence for every new task, we design new models and train them on model-specific datasets closely mimicking the deployment environments. Yet, an ML architecture called Transformer has enabled previously unimaginable generalization in other domains. Nowadays, one can download a model pre-trained on massive datasets and only fine-tune it for a specific task and context with comparatively little time and data. These fine-tuned models are now state-of-the-art for many benchmarks.

We believe this progress could translate to networking and propose a Network Traffic Transformer (NTT), a transformer adapted to learn network dynamics from packet traces. Our initial results are promising: NTT seems able to generalize to new prediction tasks and environments. This study suggests there is still hope for generalization through future research.

CCS CONCEPTS
• Networks → Network dynamics; • Computing methodologies → Neural networks;

KEYWORDS
Transformer, Packet-level modeling

1 INTRODUCTION
Modeling network dynamics is a sequence modeling problem: From a sequence of past packets, estimate the current state of the network (e.g., Is it congested?), then predict the state’s evolution and future traffic’s fate—or which action to take next. This is a notoriously complex task, and the research community is increasingly turning to Machine Learning (ML) for solutions in many applications, including congestion control [4, 20, 28, 36], video streaming [5, 25, 38], traffic optimization [11], routing [34], flow size prediction [15, 29], MAC protocol optimization [21, 40], and network simulation [42].

Problem Today’s models do not generalize well; i.e., they often fail to deliver outside of their original training environments [7, 8, 16, 38, 39]; generalizing to different tasks is not even considered. Recent work argues that, rather than hoping for generalization, one obtains better results by training in-situ, i.e., using data collected in the deployment environment [38]. Thus, today we tend to design and train models from scratch using model-specific datasets (Fig. 1, top). This process is repetitive, expensive, and time-consuming. Moreover, the growing resource requirements to even attempt training these models is increasing inequalities in networking research and, ultimately, hindering collective progress.

Vision We argue there is still hope for generalization in networking. Even if the networking contexts (topology, network configuration, traffic, etc.) are very diverse, the underlying dynamics remain similar; e.g., when buffers fill up, queuing disciplines delay or drop packets. These dynamics can be learned with ML, and there is no need to relearn everything every time, e.g., how congestion looks like.

We envision a generic network model trained to capture the shared dynamics underpinning any network—once—which can be fine-tuned for many different networking tasks and contexts.
Tackling this challenge would benefit the entire community. Starting from such a model, one would only need to collect a small task-specific dataset for fine-tuning (Fig. 1, bottom), assuming that the pre-trained model generalizes well.

Existing approaches—while not performing optimally outside of their training environments—provide evidence for generalization. The congestion control algorithm Aurora [20] performs adequately in environments with bandwidth more than an order of magnitude higher than during training. Models trained with Genet [37] on simulation data perform well in several real-world settings. But truly “generic” models—able to perform well on a wide range of tasks and networks—remain unavailable, as mixing different contexts is unpredictable. In some cases, more diverse training data has been shown to provide benefits without consequences, e.g., training over a wide range of propagation delays in [32]. Yet in other cases, mixing contexts can decrease performance, e.g., varying numbers of senders in [32] or wired and wireless traces in [8], if the model is not able to tell these contexts apart.

**Game-changer** A few years ago, a new architecture for sequence modeling was proposed: the Transformer [35]. This architecture is designed to train efficiently, enabling learning from massive datasets and unprecedented generalization across multiple contexts. In a pre-training phase, the transformer learns contextual sequential “structures,” e.g., the structure of a language from a large corpus of texts. Then, in a much quicker fine-tuning phase, the final stages of the model are adapted to a specific prediction task. Today, transformers are among the state-of-the-art in natural language processing (NLP [33]) and computer vision (CV [17, 24]).

Transformers generalize well because they can learn to distinguish different contexts during pre-training; they learn rich contextual representations [13] where the representation of the same element, e.g., a word, depends on its context, inferred from the sequence. Consider two input sequences: *Stick to it!* and *Can you hand me this stick?* The transformer output for each *stick* is different as it encodes the word’s context. This contextual output is an efficient starting point for fine-tuning the model to diverse downstream tasks, e.g., question answering, text comprehension, or sentence completion [33]. We can draw parallels between networking and NLP: packet metadata alone (headers, delay, etc.) provide limited insights into the network state—we also need the context, i.e., the history of past packets. For example, increasing latency indicates congestion, and loss patterns or ACK batching may allow for differentiating wired and wireless connections.

We believe a transformer can learn many such network-specific contexts. If it does, it could pave the way for generalization in networking, as it did for NLP and CV.

**Challenges** Naively transposing NLP or CV transformers to networking fails, unsurprisingly. We must adapt them to the peculiarity of networks. In particular, “sequences” must be carefully defined: While text snippets and images are relatively self-contained, any packet trace only gives a partial view of the network. Moreover, generalizing the diversity and dynamism of protocol interactions is far from trivial. Ultimately, we identify three main open questions.

1. How to adapt transformers for networking?
2. Which pre-training task would allow the model to generalize, and how far can we push generalization?
3. How to assemble a dataset large and diverse enough to allow useful generalization?

**Contributions** After a short background on transformers (§2), we begin to answer these questions and present NTT: our proof-of-concept *Network Traffic Transformer* (§3, [30]). Preliminary simulations (§4) provide first evidence that NTT can learn network traffic dynamics and generalize to new tasks and environments. This opens a broad research agenda (§5).

## 2 BACKGROUND ON TRANSFORMERS

In this section, we introduce attention, the mechanism behind Transformers; detail the idea of pre-training and fine-tuning; and present insights from adapting Transformers to CV.

**Sequence modeling with attention** Transformers are built around the attention mechanism, which maps an input sequence to an output sequence of the same length. Every output encodes its own information and its context, i.e., information inferred from related elements in the sequence. For a detailed explanation, we refer to [35] and excellent online guides [6, 18]. Computing attention is efficient as all elements in the sequence can be processed in parallel with matrix operations that are highly optimized on most hardware.

While attention originated as an improvement to recurrent neural networks (RNNs), Vaswani et al. [35] realized that it could replace them entirely. The authors propose an architecture for translation tasks that contains: an embedding layer mapping words to vectors; a transformer encoder encoding the input sequence; and a transformer decoder generating an output sequence based on the encoded input (Fig. 2a). Each transformer block alternates between attention and linear layers, i.e., between encoding context and refining features.

**Pre-training and fine-tuning** Transformers are used for a wide range of NLP tasks, and the prevailing strategy is to use pre-training and fine-tuning. We explain this approach on the example of BERT [13], one of the most widely used transformer models. BERT uses only the transformer encoder, followed by a small and replaceable decoder. Usually, a multilayer perceptron (MLP) with a few linear layers; this decoder is often called the ‘MLP head’.
a task that requires learning language structure. Concretely, a fraction of words in the input sequence is masked out, and the decoder is tasked to predict the original words from the encoded input sequence (Fig. 2b). Conceptually, this is only possible if the encoding includes sufficient context to infer the missing word. Afterward, the unique pre-trained model can be fine-tuned to many different tasks by replacing the small decoder with task-specific ones, e.g., language understanding, question answering, or text generation [12, 13, 41]. The model has already learned to encode language context and only needs to learn to extract the task-relevant information from this context. This requires far less data compared to starting from scratch: BERT is pre-trained from text corpora with several billion words and fine-tuned with \(\sim100\) thousand examples per task. Furthermore, BERT’s pre-training task is unsupervised, i.e., it requires only “cheap” unlabelled data for masking and reconstruction. “Expensive” labeled data, e.g., for text classification, is only needed for fine-tuning.

Vision transformers Following their success in NLP, Transformers gained traction in CV as well, with two notable distinctions: (i) input aggregation; and (ii) a domain-specific pre-training task. While attention is efficient to parallelize, it needs to compare each element in the sequence with each other element to encode context. Consequently, the required computation scales quadratically with the input sequence length, and using sequences of individual pixels does not scale to images of high resolution. As a solution, the Vision Transformer (ViT, [9, 14]) aggregates pixels into \(16\times16\) patches and applies the embedding and transformer layers to the resulting sequence of patches, using an architecture similar to BERT (Fig. 2c). However, using a classification task to pre-train ViT delivered better results than a reconstruction task. This shows the importance of domain-appropriate pre-training: it may be possible to reconstruct a patch by only considering neighboring ones, but classification requires understanding the whole image, i.e., the context of the entire sequence.

3 NETWORK TRAFFIC TRANSFORMER

Given the success of Transformers in NLP and CV and the similarities between the underlying sequence modeling problems, we postulate that transformers could also generalize network traffic dynamics. This section presents our proof-of-concept: the Network Traffic Transformer (NTT, Fig. 3).

(1) Packets are more complex than words or pixels. Which packet features are helpful and which are necessary to learn network dynamics?

(2) The fate of a packet may depend on much older ones. As the sequence length is practically limited (§2), how can we capture both short- and long-term network dynamics in the input sequence?

(3) Which pre-training task enables the model to learn general network patterns effectively?

Learning feature extraction Packets carry a lot of information that could be used as model features, e.g., header fields. Today, we typically use domain knowledge to manually extract and aggregate features and feed these into off-the-shelf ML architectures. We argue this is sub-optimal for two reasons: (i) we select features for a specific task and dataset, which limits generalization; (ii) since the features are not learned from data, they may end up sub-optimal for the task.
Instead, we propose to let the model learn useful features from raw data. To learn traffic dynamics from a sequence of packets, we must provide the model with information about the packets as well as their fate in the network. Since we do not want to define a priori how important the individual pieces of information are, we feed them all into the first embedding layer (Fig. 3). It is applied to every packet separately.

In our proof-of-concept, we use minimal information: timestamp, packet size, receiver ID, and end-to-end delay. These enable learning embeddings with temporal (delays over time) and spatial (impact of packet size on delay) patterns. We discuss the challenge of embedding more information in §5.

**Learning packet aggregation** Packet sequences must be sufficiently long to capture more than short-term traffic dynamics. But as the training time of Transformers scales quadratically with the sequence length, we face practical limitations.

We address this problem by using a hierarchical aggregation layer (Fig. 3). We aggregate a long packet sequence into a shorter one while letting the model learn how to aggregate the relevant historical information, similar to the pixel patch aggregation in ViT [14]. However, we aim to both aggregate and retain recent packet-level details. To achieve this, we keep the most recent packets without aggregation and the longer traffic is in the past, the more we aggregate, as details become less relevant to predict the current traffic dynamics.

In our proof-of-concept, we set the input sequence length to 1024 packets, enough to cover the number of in-flight packets in our experiments. We aggregate this sequence into 48 elements in two stages: the most recent packets are kept as-is; less recent packets are aggregated once; and the least recent twice (Fig. 3). Our multi-timescale aggregation is easy to adapt to a larger history without sacrificing recent packet details. We show in §4 that this aggregation is beneficial, but it is unclear which sequence length and levels of aggregation generalize best; we discuss this further in §5.

**Learning network patterns** Finally, we need a training task that allows NTT to learn network dynamics: in our proof-of-concept, we use end-to-end delay prediction. We aim to pre-train NTT to generalize to a large set of fine-tuning tasks. Consequently, we need a pre-training task that is generic enough to be affected by many network effects. As almost everything in a network affects packet delays (e.g., path length, buffer sizes), a delay prediction task seems a rational choice.

To pre-train NTT, we mask the delay of the most recent packet in the sequence and use a decoder with linear layers to predict the actual delay. During training, the NTT must learn which features are useful (embedding layer), how to aggregate them over time (aggregation layer), and to infer context from the whole sequence (transformer encoder layers).

---

3An IP addresses proxy, as we do not want to learn IP address parsing (yet).

---

### Figure 4: Dataset generation setup.

During fine-tuning, one can update or replace the decoder (Fig. 3) to adapt NTT to a new environment (e.g., same decoder in a different network) or to new tasks (e.g., predicting message completion times). This is efficient as the knowledge accumulated by NTT during pre-training generalizes well to the new task, as we demonstrate in the next section.

## 4 PRELIMINARY EVALUATION

Our preliminary evaluation of NTT in simulation shows that:

1. **NTT is able to learn some network dynamics**;
2. **Pre-training helps to generalize**;
3. **Networking-specific design helps generalization**.

Importantly, we do not aim to show that NTT outperforms existing specialized models (yet\(^4\)). Here, we focus on assessing the potential of our approach.

**Datasets** We use ns-3 [31] and the setup in Fig. 4 to generate several datasets; one for pre-training, and several for fine-tuning. From each, we reserve a fraction for testing.

In the pre-training dataset, 60 senders generate 1Mbps of messages each, following real-world traffic distributions [26]. They send messages over a bottleneck link with 30Mbps bandwidth and a queue size of 1000 packets. We run 10 simulations for 1 minute each with randomized application start times. This dataset contains about 1.2 million packets.

For the fine-tuning datasets, we add cross-traffic (case 1) and additionally extend the network topology (case 2). Cross-traffic is modeled as 20Mbps of TCP flows. Note that the datasets do not contain the cross-traffic packets, only those from the senders. For each case, we generate a dataset containing roughly as many packets as the pre-training dataset and a “smaller” dataset containing about 10% of the packets.

**Models** We compare several versions of NTT. The pre-trained models first learn from the pre-training and then one fine-tuning dataset, while the from-scratch versions only learn from one fine-tuning dataset. We also pre-train ablated versions of NTT: we compare our multi-timescale aggregation of 1024 packets into 48 aggregates (see §3) with no aggregation (using only 48 individual packets) and fixed aggregation (using

\(^4\)The research on Transformers in CV showed that large datasets are required for transformers to outperform the state-of-the-art.
fixed aggregation

In addition, we pre-train one model without delay and one without packet size information in the input sequences. Finally, we consider two naive baselines: one always returns the last observed output value; another returns an EWMA.\(^5\)

**Tasks**  We evaluate our models on two prediction tasks. The first is to predict the delay of the last packet of the sequence; this task is also used for pre-training. The second task is to predict the message completion times (MCTs), i.e., the time until the final packet of a message is delivered. This flow-level prediction task uses a decoder with two inputs: the NTT outputs for the past packets and the message size. We report the mean-squared error (MSE) for both tasks and process MCTs on a logarithmic scale to limit the impact of outliers.\(^6\)

### Case #1 – Generalization on the same topology

We first consider the fine-tuning case 1, where we add unseen cross-traffic on the same topology (see Fig. 4, Tables 1 and 2).

First, we confirm that the pre-trained NTT beats all baselines (Table 1). While this is no breakthrough (the baselines are basic), it suggests that NTT indeed learns sensible values.

Second, we observe that pre-training is beneficial: on both fine-tuning tasks, the pre-trained NTT outperforms the from-scratch version (Table 1); it generalizes to a new context (i.e., unseen cross-traffic) and a new task (i.e., MCT prediction).

Third, we observe the benefits of hierarchical aggregation and the mix of network and traffic information in the raw data (Table 1). With no aggregation, the model has little history available; we observe that, perhaps surprisingly, this affects the delay predictions but not the MCT ones. Conversely, with a fixed aggregation, the model loses packet-level details.

### Case #2 – Generalization on a larger topology

We now consider the fine-tuning case 2, with several cross-traffic sources on a larger topology (Fig. 4). In this setting, packets toward different receivers experience different path delays and different levels of congestion from cross-traffic.

### Table 1: Mean Squared Error for all models and tasks.
The pre-trained NTT outperforms the from-scratch version and benefits from our design choices (see §3).

<table>
<thead>
<tr>
<th></th>
<th>Pre-training</th>
<th>Fine-tuning (10%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delay</td>
<td>Delay</td>
</tr>
<tr>
<td>NTT Pre-trained</td>
<td>0.072</td>
<td>0.097</td>
</tr>
<tr>
<td>From scratch</td>
<td>-</td>
<td>0.313</td>
</tr>
<tr>
<td>Baselines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Last observed</td>
<td>0.142</td>
<td>0.121</td>
</tr>
<tr>
<td>EWMA</td>
<td>0.259</td>
<td>0.211</td>
</tr>
<tr>
<td>NTT (Ablated)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No aggregation</td>
<td>0.258</td>
<td>0.430</td>
</tr>
<tr>
<td>Fixed aggregation</td>
<td>0.055</td>
<td>0.134</td>
</tr>
<tr>
<td>Without packet size</td>
<td>0.001</td>
<td>8.688</td>
</tr>
<tr>
<td>Without delay</td>
<td>15.797</td>
<td>10.898</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Layers trained</th>
<th>MSE(Delay)</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-trained</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine-tuning (full)</td>
<td>0.033</td>
<td>8h45</td>
</tr>
<tr>
<td>Fine-tuning (10%)</td>
<td>0.037</td>
<td>3h45</td>
</tr>
<tr>
<td>From scratch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine-tuning (full)</td>
<td>0.036</td>
<td>26h</td>
</tr>
<tr>
<td>Fine-tuning (10%)</td>
<td>0.118</td>
<td>8h40</td>
</tr>
</tbody>
</table>

Table 2: On a simple setting, pre-training saves training resources: fine-tuning data and computing power.

but has access to a longer history; this seems sufficient to predict delays but not MCT. More generally, this initial result suggests that both recent packet-level information and an aggregated history are useful to generalize to a large set of tasks. Considering the NTT versions without packet size and without delay information, we observe that neither generalize. Without packet size, the model overfits the pre-training dataset and performs poorly on predicting delay for fine-tuning. Without delay information, the model can logically not produce any sensible prediction related to packet delays or MCTs.

Finally, one can argue that the pre-trained NTT has an unfair advantage as it trained on about ten times more data than the from-scratch version. To put things into perspective, Table 2 compares the delay MSE and training time for NTT versions fine-tuned on different datasets. We observe that fine-tuning on a full dataset from scratch yields about the same performance as the on the 10% dataset after pre-training.\(^7\) However, fine-tuning on the full dataset also requires almost seven times as much training time (26h vs. 3h45). In practice, collecting fine-tuning data is often expensive; it is thus beneficial to require less. Finally, fine-tuning from scratch may just not work in more complex settings, as shown next.

### Table 3: On a larger topology, fine-tuning from scratch no longer works, even if using a large dataset.

\(^5\)Exponentially Weighted Moving Average; we used \(\alpha = 0.01\).

\(^6\)MCT mean: 0.2s; 99.9th percentile: 23s

\(^7\)Tables 1 and 2 were obtained with different "10% fine-tuning datasets". The data allows comparison within each table but not across the two tables.
As evident from Table 3, pre-training is essential for NTT to learn basic congestion dynamics first, then generalize later on to the topology’s specifics during fine-tuning.\(^8\) When fine-tuning from scratch, even the full dataset is not enough to learn; performance is worse than the baselines (MSE of 11.2 and 4.0—not shown). Without addressing information, NTT cannot differentiate between the receivers and thus cannot predict the packet delay accurately (MSE of 2.8—not shown).

5 CONCLUSION & FUTURE RESEARCH

Our initial results are promising: they show that NTT effectively learns, that the pre-training knowledge generalizes to new tasks and contexts, and that its specific design benefits overall performance. Nevertheless, it merely validates that NTT may work. There is a lot more research to assess whether our vision can indeed become a reality.

Does the premise hold? We showed some potential of pre-training and fine-tuning with small-scale simulations. However, real networks are undeniably more complex than this environment. Real topologies include many paths where many different applications, transport protocols, queuing disciplines, etc. coexist. There are also many more fine-tuning tasks to consider, e.g., flow classification for security or anomaly detection. Testing our NTT prototype in real, diverse environments and with multiple fine-tuning tasks would provide invaluable insights into the strengths and weaknesses of our architecture and the ‘learnability’ of network dynamics in general. A next step would be experiments to analyze real-world datasets from Caida [1], M-LAB [3], or Crawdad [2].

How does the NTT hold up with more diverse environments and fine-tuning tasks? Which aspects of network dynamics are easy to generalize to, and which are difficult?

Advancing NTT  Our prototype architecture [30] needs enhancements to be helpful in more diverse environments. We see three directions for improvement: (i) packet headers; (ii) network telemetry; and (iii) sequence aggregation. Considering packet headers may be essential to learning the behavioral differences of transport protocols or network prioritization of different traffic classes. However, raw headers are challenging inputs for an ML model, as they may appear in many combinations and contain values that are difficult to learn, like IP addresses [42]. Research from the network verification community on header space analysis [23] may provide valuable insights on header representations and potential first steps in this direction. In addition, we may collect telemetry data like packet drops or buffer occupancy. This may help to learn, but not every trace will contain all telemetry, and future research will need to address this potential mismatch. Finally, we base our prototype aggregation levels on the number of in-flight packets, i.e., whether packets in the sequence may share some fate, usually determined by buffer sizes. The further packets are apart, the less likely they do, and the more we aggregate. We believe matching individual aggregation levels to typical buffer sizes (e.g., flow and switch buffers) may be beneficial. Still, future research needs to put this hypothesis to the test and determine the best sequence sizes and aggregation levels across multiple networks.

Collaborative pre-training  Transformers in NLP and CV truly outshine their competition only when pre-trained with massive amounts of data. We envision this could require a previously unseen collaboration across the networking industry. We see two main challenges: (i) training data volume; and (ii) privacy concerns preventing data sharing. One can also see these challenges as opportunities: First, ML models effectively compress data. For example, GPT-3 [10], one of the largest current Transformer models, consists of 175 Billion parameters or roughly 350 Gigabytes. However, it contains information from over 45 Terabytes of text data: Sharing a pre-trained model is much more feasible than sharing all the underlying data, not to mention the savings in training resources. Second, sharing models instead of data could overcome privacy barriers via federated learning [22]: Organizations could keep their data private and only share pre-trained models, which can be combined into a final collectively pre-trained model.

Can we leverage pre-training and federated learning to learn from previously unavailable data?

Continual learning  A cat remains a cat, but the Internet is an evolving environment. Protocols, applications, etc., change over time. We conjecture that underlying network dynamics change less frequently than specific environments; thus, the same NTT may be used for several updates of the same fine-tuned model. Nevertheless, even a pre-trained model may become outdated. It is already difficult to determine when to re-train a specific model [38]; it might be even more difficult for a model supposed to capture a large range of environments.

At which point should we consider an NTT outdated? When and with what data should it be re-trained?

Acknowledgements  We thank our anonymous reviewers for their helpful comments and feedback. This work was partially supported by ETH Research Grant ETH-03 19-2.
REFERENCES


