A New Hope for
Network Model Generalization

ACM HotNets 2022

Alexander Dietmüller
Siddhant Ray
Romain Jacob
Laurent Vanbever

ETH Zürich
What do these systems have in common?

Video streaming, congestion control, and load balancing
SIGCOMM’22 [GENET]

Network simulation
SIGCOMM’21 [MimicNet]

Video streaming
NSDI’20 [Puffer]

Data-driven networking
HotNets’16 [Biases]
What do these systems have in common?
They have the same problem setting.

From past traffic ...

... an ML system estimates the state of the network to make a prediction.

MimicNet packet (drop, latency, ECN)
Puffer transmission time
GENET bitrate for next chunk
...

3
What do these systems have in common?
They have the same problem setting. But that’s about it.

From past traffic ...

... an ML system estimates the state of the network to make a prediction.

MimicNet    packet (drop, latency, ECN)
Puffer       transmission time
GENET        bitrate for next chunk
...

4
ML systems in networking do not generalize.

tailored to network context & task
(e.g. predict wireless loss)
SO
WHAT
SO
WHAT
are the consequences for ML in networking?
ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.

tailored to network context & task
(e.g. predict wireless loss)
ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.

- **tailored to network context & task** (e.g. predict wireless loss)
- **Same task (predict loss) with data from** a context in situ ✓ [Puffer]
ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.

- **Same task** (predict loss) with data from
  - a context in situ ✓ [Puffer]
  - a similar context (*wireless*) ✓ / ✗ [GENET]

*tailored to network context & task (e.g. predict wireless loss)*
ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.

Same task *(predict loss)* with data from:
- a context in situ: ✓ [Puffer]
- a similar context *(wireless)*: ✓ /[GENET]
- a different context *(wired)*: ✗ [Biases]
ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.

Same task (predict loss) with data from:
- a context in situ ✓ [Puffer]
- a similar context (wireless) ✓/✗ [GENET]
- a different context (wired) ✗ [Biases]
- multiple contexts (both) ✗
ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.

**Same task** *(predict loss)* with data from
- a context in situ ✓ [Puffer]
- a similar context *(wireless)* ✓ / ✗ [GENET]
- a different context *(wired)* ✗ [Biases]
- multiple contexts *(both)* ✗

**Different task** *(e.g. predict delay)*
- ✗ (requires a completely new model and data)
Is there no way to get

- optimal performance
- for multiple contexts and different tasks
- without starting from scratch every time?
A New Hope for
Network Model Generalization
In NLP and CV, Transformer-based architectures generalize by learning to infer sequence context.
In NLP and CV, Transformer-based architectures generalize by learning to infer sequence context.

Dall-E 2
input: (text)

output: (generated image)
In NLP and CV, Transformer–based architectures generalize by learning to infer sequence context.

Dall–E 2
input: (text)  
Hand me that stick!  
Stick to that hand.

output: (generated image)
In NLP and CV, Transformer–based architectures generalize by learning to infer sequence context.

Dall–E 2
input: (text) Hand me that stick! Stick to that hand.

output: (generated image)

NLP: Natural Language Processing; CV: Computer Vision; Images generated by OpenAI Dall–E 2.
In NLP and CV, Transformer–based architectures generalize by learning to infer sequence context.

Dall–E 2
input: (text)  Hand me that stick!  Stick to that hand.
output: (generated image)
Maybe we can get

- optimal performance
- for multiple contexts and different tasks
- without starting from scratch every time?
A general pre-trained Transformer encoder can be combined with specific fine-tuned decoders.
A general pre-trained Transformer encoder can be combined with specific fine-tuned decoders.

- From past traffic ...
- ... a large and general **Transformer** learns dynamics in context ...

- **sequence of packets**
- **sequence of encoded packets**
  - each packet augmented with inferred context
A general pre-trained Transformer encoder can be combined with specific fine-tuned decoders.

From past traffic ... … a large and general Transformer learns dynamics in context ... … while small and specific decoders make predictions.

*sequence of packets*

*sequence of encoded packets*  
each packet augmented with inferred context
There is a way to get

- optimal performance
- for multiple contexts and different tasks
- without starting from scratch every time
We cannot just copy an NLP Transformer: a Network Traffic Transformer (NTT) must handle network challenges!
We cannot just copy an NLP Transformer: a Network Traffic Transformer (NTT) must handle network challenges!

**Challenge #1**
Avoid packet features tailored to a specific task.

**Challenge #2**
Process long sequences without losing detail.

**Challenge #3**
Learn contextual dynamics during pre-training.
We cannot just copy an NLP Transformer: a Network Traffic Transformer (NTT) must handle network challenges!

Challenge #1
Avoid packet features tailored to a specific task.
→ learning features

Challenge #2
Process long sequences without losing detail.
→ aggregate past packets hierarchically

Challenge #3
Learn contextual dynamics during pre-training.
→ pre-train to predict end-to-end delay
In simulation, we observe first evidence that networking could benefit from pre-trained models as well.

We pretrain, ...

**context**
30 senders and a single shared bottleneck

**task**
delay prediction
In simulation, we observe first evidence that networking could benefit from pre-trained models as well.

*We pretrain, ...*  
... fine-tune, ...

**context**  
30 senders and a single shared bottleneck

**task**  
delay prediction

**with different contexts**  
indep. bottlenecks with unobserved cross-traffic
In simulation, we observe first evidence that networking could benefit from pre-trained models as well.

*We pretrain, ...*

**context**
30 senders and a single shared bottleneck

**task**
delay prediction

... fine-tune, ...

**with different contexts**
indep. bottlenecks with unobserved cross-traffic

**with another task**
predict message completion time
In simulation, we observe first evidence that networking could benefit from pre-trained models as well.

We pretrain, ...  ... fine-tune, ...

context
30 senders and a single shared bottleneck

task
delay prediction

with different contexts
indep. bottlenecks with unobserved cross-traffic

with another task
predict message completion time

... and find that we:

- get equal or better performance
- with less training time compared to starting from scratch.
In simulation, we observe first evidence that networking could benefit from pre-trained models as well.

We pretrain, ...

... fine-tune, ...

... and find that we:

- get equal or better performance
- with less training time

We pretrain, tasks with different contexts:
- indep. bottlenecks with unobserved cross-traffic
- with another task, predict message completion time

... and find that we:
NTT *may* generalize.

What next?
Our simulation results are promising, and it is time to use and evaluate NTT-based models in the real-world.

Re-create existing models based on NTT, collecting new data where needed.
Our simulation results are promising, and it is time to use and evaluate NTT-based models in the real-world.

Re-create existing models based on NTT, collecting new data where needed.

Create new models based-on NTT.
Real-world applications will reveal all limits, but there are clear steps to refine the NTT design.
Real-world applications will reveal all limits, but there are clear steps to refine the NTT design.

How can we represent any combination of headers?
Real-world applications will reveal all limits, but there are clear steps to **refine the NTT design.**

How can we represent any combination of headers?

Which aggregation levels cover all significant network interactions?
Transformer models like NTT extract and compress information, facilitating sharing and collaboration.
Transformer models like NTT extract and compress information, facilitating sharing and collaboration.

\[
\frac{1}{100} \text{ in NLP!}
\]

- **Train from Large Traces Every Time**
- **Download a Pre-trained Model**
Transformer models like NTT extract and compress information, facilitating sharing and collaboration.

- train from large traces every time
- download a pre-trained model
- share private data
- combine insights via federated learning

private data $\xrightarrow{}$ NTT$_1$

private data $\xrightarrow{}$ NTT$_2$

private data $\xrightarrow{}$ NTT$_3$

combined public NTT
A New Hope for Network Model Generalization

ACM HotNets 2022

Pretraining

Transformers

Networking

Alexander Dietmüller
Siddhant Ray
Romain Jacob
Laurent Vanbever

ETH Zürich