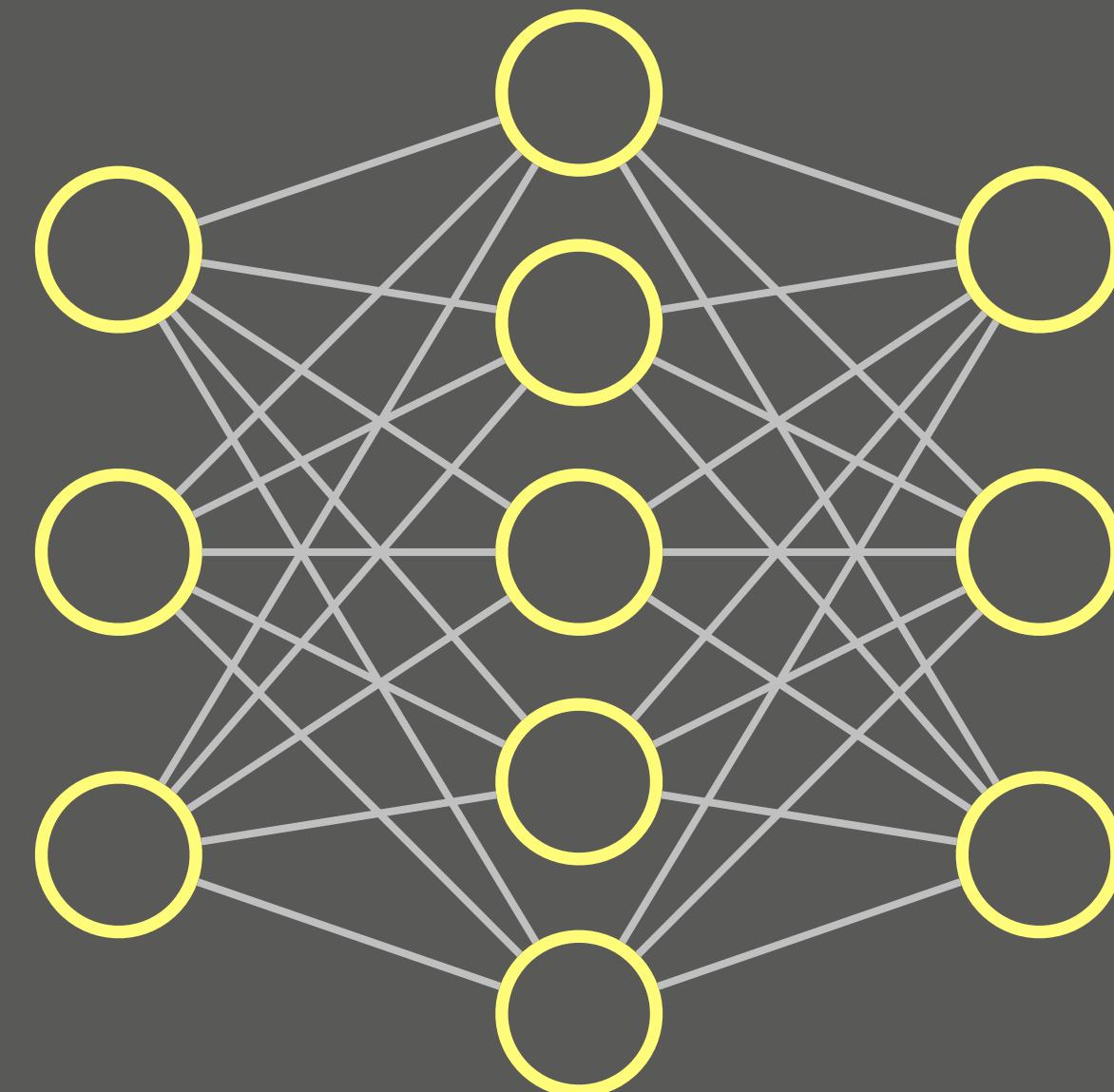


Machine Learning for Networking

What? With what? For what?



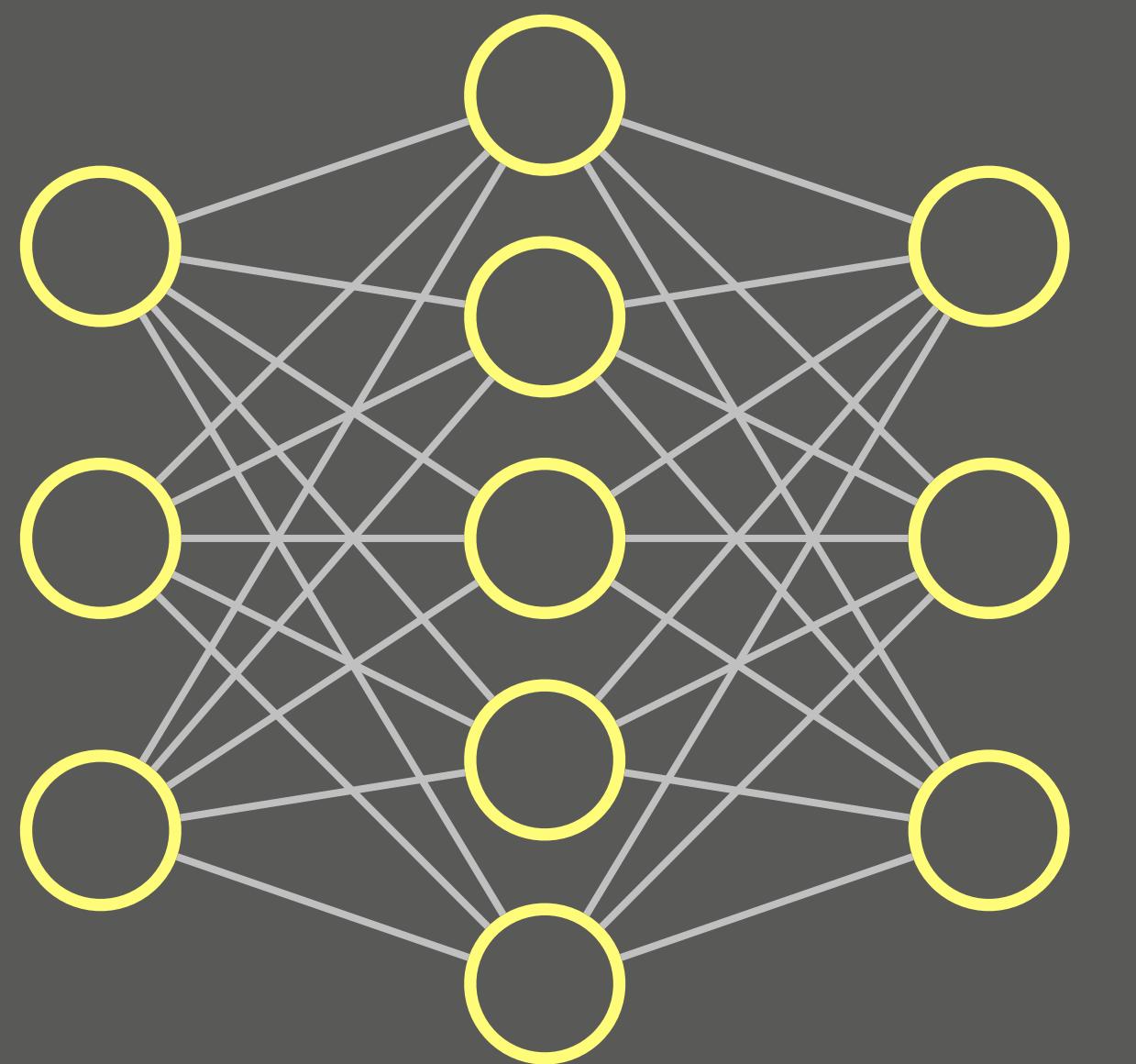
Ceci n'est pas un réseau.

Laurent Vanbever
nsg.ee.ethz.ch

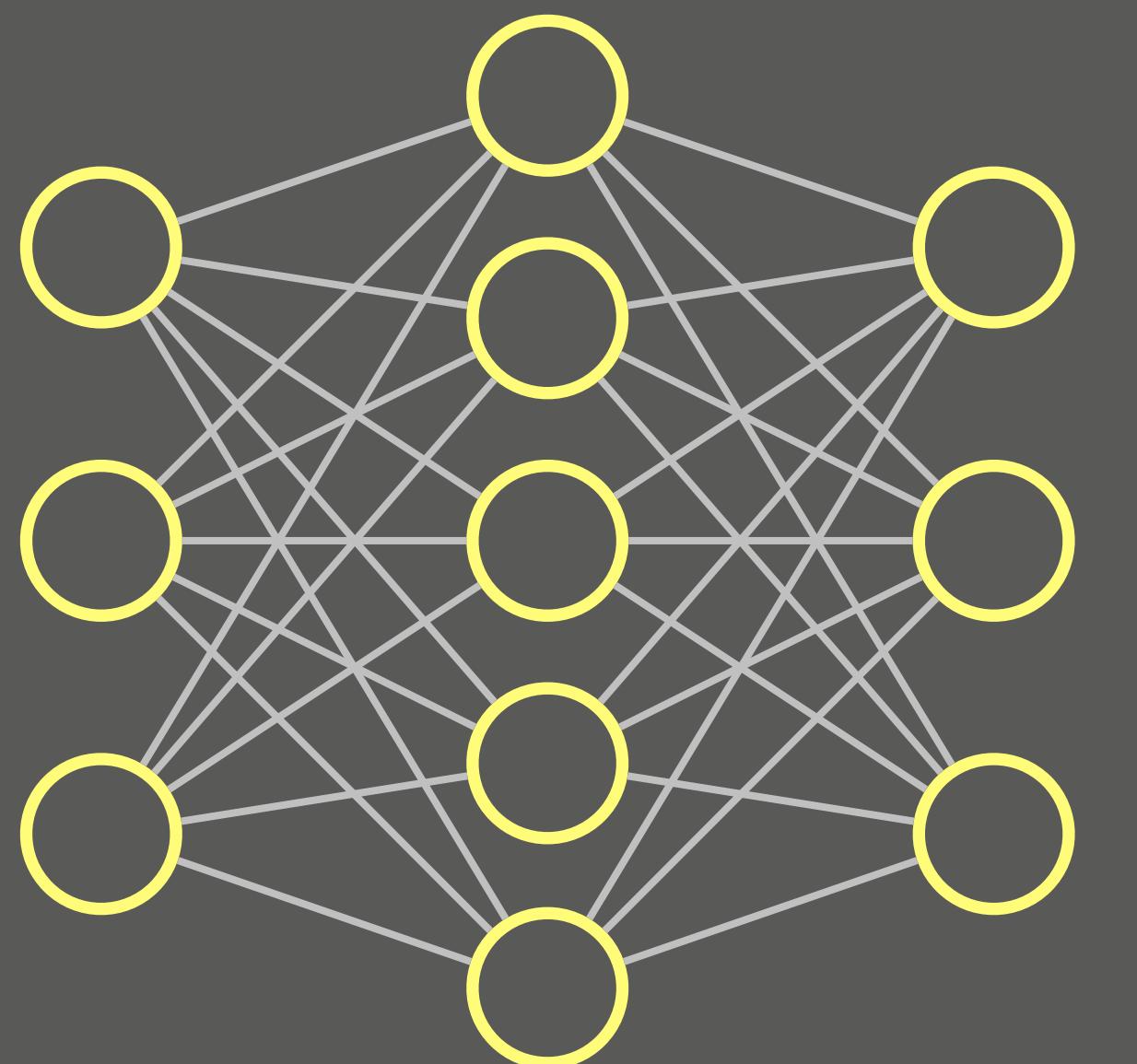
Google Networking Summit
Wed Oct 18 2023



René Magritte. La trahison des images. 1929.



René Magritte. La trahison des images. 1929.



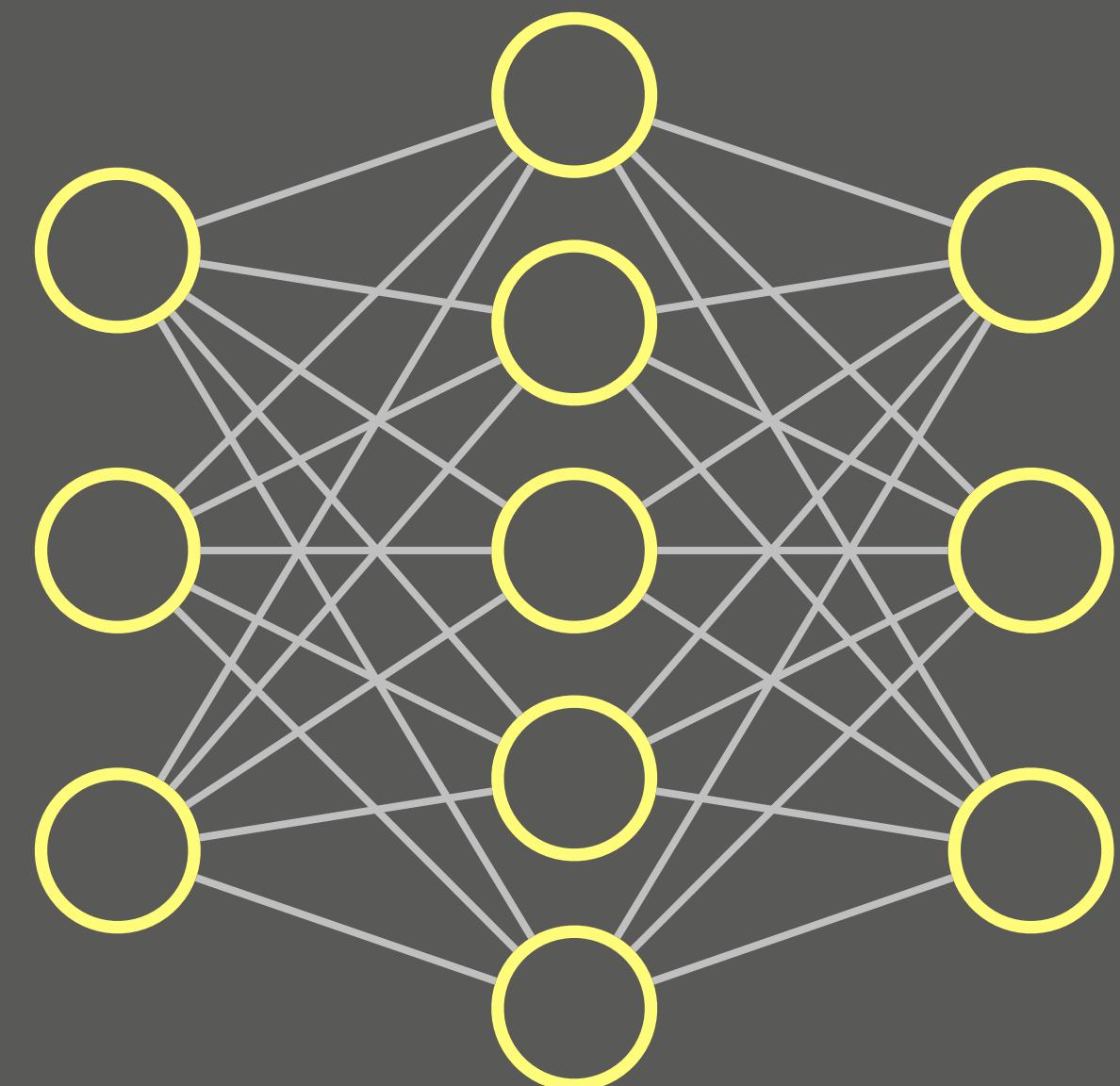
Ceci n'est pas un réseau.



René Magritte. La trahison des images. 1929.

Machine Learning for Networking

What? With what? For what?

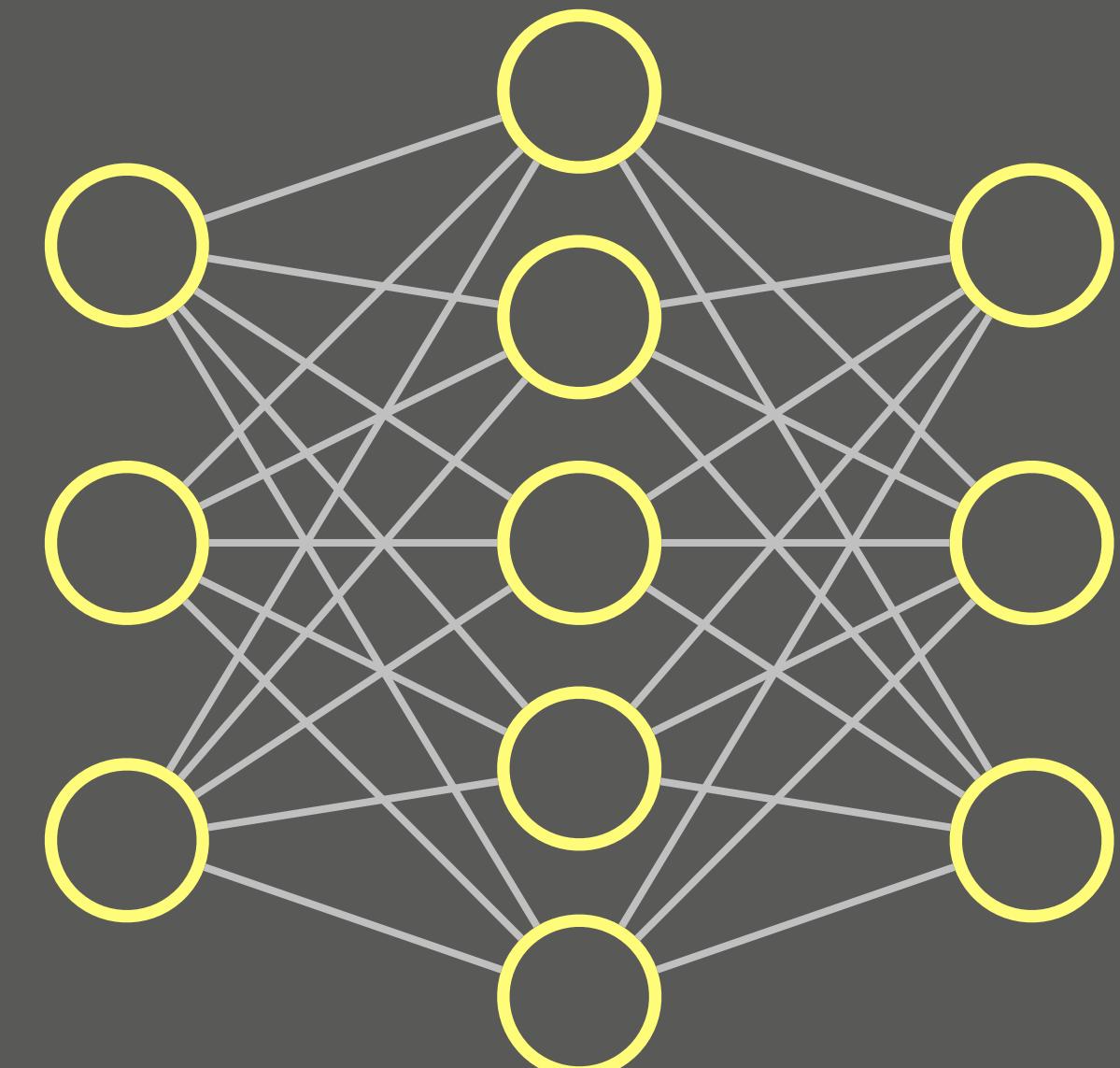


Ceci n'est pas un réseau.

- 1 What to train?
transformers to the rescue
 - 2 With what data?
big code × networking
 - 3 For what purpose?
fun with LLMs

Machine Learning for Networking

What? With what? For what?



Ceci n'est pas un réseau.

1

What to train?
transformers to the rescue

With what data?
big code × networking

For what purpose?
fun with LLMs

A New Hope for Network Model Generalization

Alexander Dietmüller* Siddhant Ray
ETH Zürich ETH Zürich

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

ACM Reference Format:

Alexander Dietmüller*, Siddhant Ray, Romain Jacob, and Laurent Vanbever. 2022. A New Hope for Network Model Generalization. In *The 21st ACM Workshop on Hot Topics in Networks (HotNets '22)*, November 14–15, 2022, Austin, TX, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3563766.3564104>

*The CRediT statement for this work is available at [19].

Romain Jacob Laurent Vanbever
ETH Zürich ETH Zürich

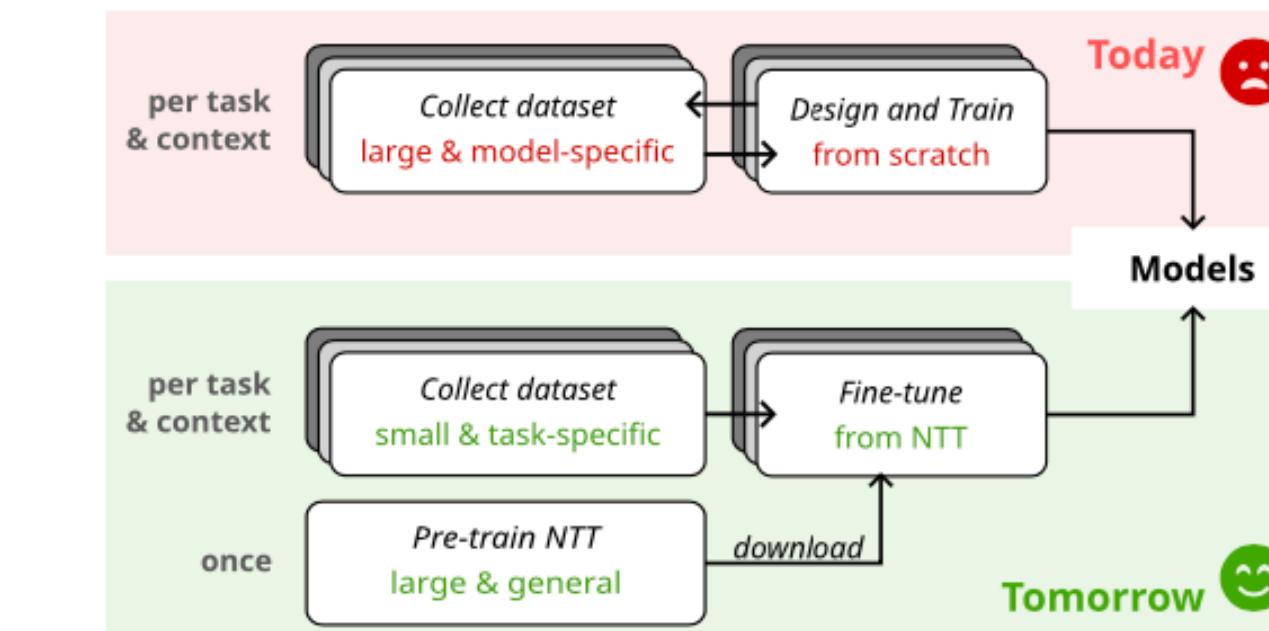


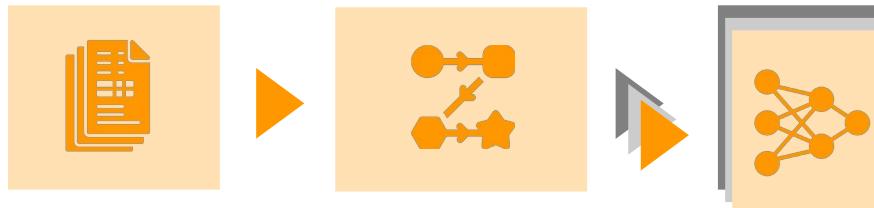
Figure 1: Could we collectively learn general network traffic dynamics *once* and focus on task-specific data collecting and learning for *all future models*?

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

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?

GENET: Automatic Curriculum Generation for Learning Adaptation in Networking

Zhenyu Xu^{1*}, Yafei Zhou², Francis Y. Yan³, Junchen Jiang¹

¹University of Chicago, ²Boston University, ³Microsoft Research

ABSTRACT

Deep reinforcement learning (RL) showcases its strengths in networking, its pitfalls are also evident to the public at large. This paper presents GENET, an automatic curriculum learning framework to improve the learning performance, whereas training on a narrow distribution of environments can easily lead to overfitting.

This work presents GENET as a new training framework for learning better RL-based network adaptation algorithms. GENET is built on top of the RL library Tensorforce. It addresses two major similar issues in other RL applications. At a high level, curriculum learning provides much more diverse environments to train the model and chooses them according to a number of metrics. However, these RL-based techniques face two challenges that are unique to networking.

- **Training in a wide range of environments:** When the training distribution spans a wide variety of network environments (e.g., different bandwidth patterns), an RL policy may overfit properly even if tested in the environment where the same distribution was used.
- **Generalization:** RL policies trained on one distribution of synthetic or trace-driven environments may have poor performance and even erroneous behavior when tested in a new distribution of environments.

Our analysis in Fig. 1 will reveal that, across three RL use cases in networking, these challenges can cause well-trained RL policies to underperform.

Video streaming, congestion control, and load balancing

SIGCOMM'22 [GENET]

Learning in situ: a randomized experiment in video streaming

Francis Y. Yan¹, Hudson Ayers², Chenzhi Zhu³, Sadiq Fouladi², Philip Levis², Keith Winstein¹

¹Stanford University, ²Tsinghua University

Abstract

We describe the results of a randomized controlled trial of video-streaming algorithms for buffer selection and network protection. Over the last year, we have seen over 38.6 years of video from 530 million unique sessions. Sessions are randomized in blinded fashion among algorithms.

We found that in this real-world setting, it is difficult for sophisticated or machine-learned control schemes to outperform In the academic literature, many recent ABR algorithms use statistical and machine-learning methods [4, 23, 38–40, 46], which often allow them to make better decisions given more input signals and to adapt quickly to a wide variety of environments. An ABR decision can depend on recent throughput, client-side buffer occupancy, delay, the experience of clients on similar ISPs or providers, and other factors. Making good decisions in seas of data and in a maturing field is a perennial lesson.

However, it is a perennial lesson that the performance of

Video streaming NSDI'20 [Puffer]

MimicNet: Fast Performance Estimates for Data Center Networks with Machine Learning

Qizhen Zhang, Kelvin K.W. Ng, Charles W. Kazer¹, Shen Yan¹, João Sedoc², and Vincent Liu³

¹University of Pennsylvania, ²Swarthmore College, ³Peking University, New York University

[✉]zhangqz@seas.upenn.edu, [✉]charlesk@seas.upenn.edu, [✉]yanshen@pku.edu.cn, [✉]jst13@nyu.edu, [✉]vincent.liu@nyu.edu

ABSTRACT

An end-to-end evaluation of new data center network innovations is becoming increasingly intractable. This is true for testbeds, where few, if any, can support a definitive full-scale replica of a data center. It is also true for simulation, where the state of the art is required for precisely this purpose, but has struggled to cope with the size of modern data centers.

This paper presents an approach for quickly obtaining accurate performance estimates for large data center networks. Our system, MimicNet, uses machine learning to learn the underlying data center-level simulation for a portion of the network while leveraging redundancy and recent advances in machine learning to provide an end-to-end performance estimate. The results show that the errors are not directly visible. MimicNet can provide over two orders of magnitude speedup compared to regular simulation for a data center with insufficient data for subsampling that could lead to misleading estimates and ultimately suboptimal decisions. In this paper, we shed light on such errors by identifying a promising path to address these pitfalls by leveraging parallels in causal inference, namely the Doubly Robust estimator.

Network simulation

SIGCOMM'21 [MimicNet]

Biases in Data-Driven Networking, and What to Do About Them

Mihovil Bartulovic¹, Junchen Jiang², Sivaraman Balakrishnan¹, Vyas Sekar¹, Bruno Sinopoli¹

¹Carnegie Mellon University, ²Microsoft Research/Carnegie Mellon University

ABSTRACT

Recent efforts highlight the promise of data-driven approaches to optimize network decisions. Many such efforts use trace-driven evaluation; i.e., performing offline analysis on network traces to evaluate the performance of a proposed policy before running them in practice. Unfortunately, such frameworks can have fundamental pitfalls (e.g., skew due to previous policy selection). We show that these pitfalls are particularly insidious for specific subpopulations that could lead to misleading estimates and ultimately suboptimal decisions. In this paper, we shed light on such errors by identifying a promising path to address these pitfalls by leveraging parallels in causal inference, namely the Doubly Robust estimator.

Figure 1: Trace-driven evaluation predicts the best policy using traces of empirical measurement data.

Figure 1 illustrates the trace-driven evaluation process as described in this paper. It shows a flowchart starting with 'Network traces' leading to 'Trace-driven policy selection' (which includes 'Policy' and 'Policy'), then to 'Decision making' (which includes 'e.g., server selection logic'), and finally to 'Real deployment (Datacenter system)'.

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

...

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.

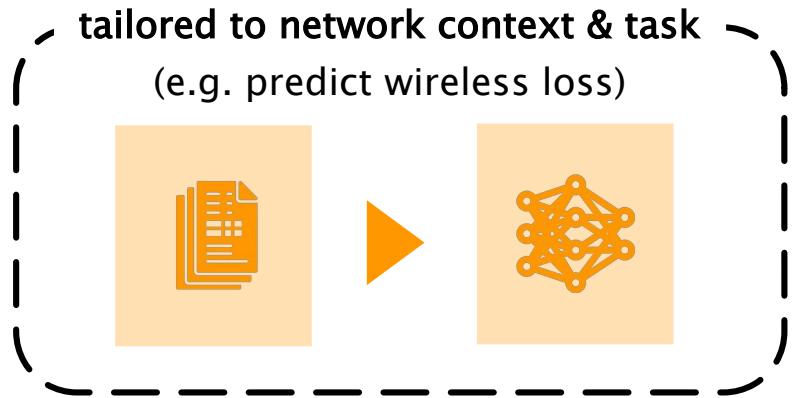
MimicNet packet (drop, latency, ECN)

Puffer transmission time

GENET bitrate for next chunk

...

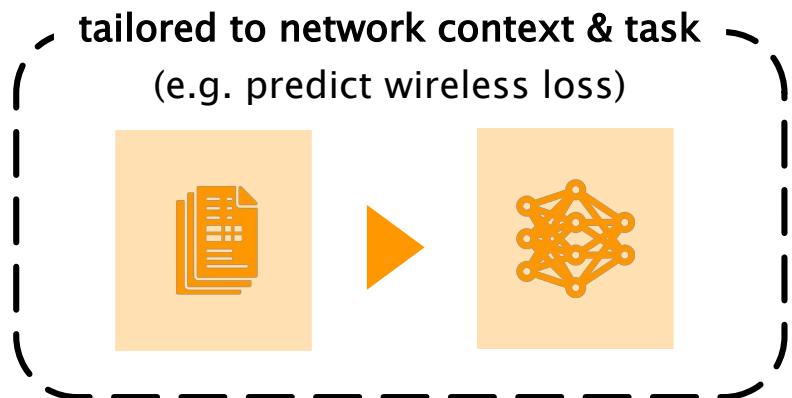
ML systems in networking do not generalize.



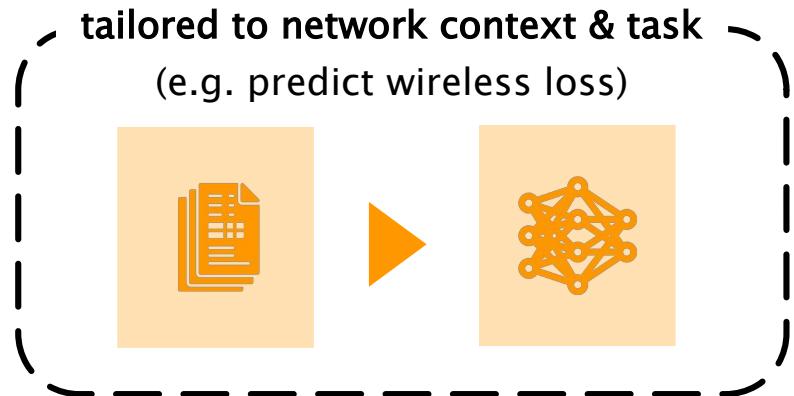
**SO
WHAT**

SO are the
WHAT 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.

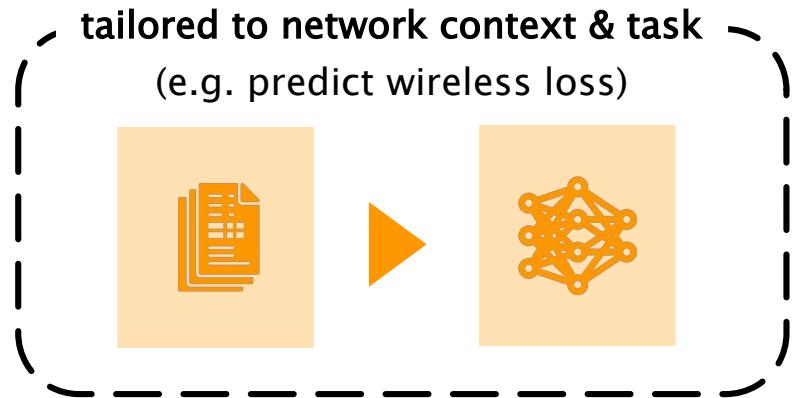


ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.



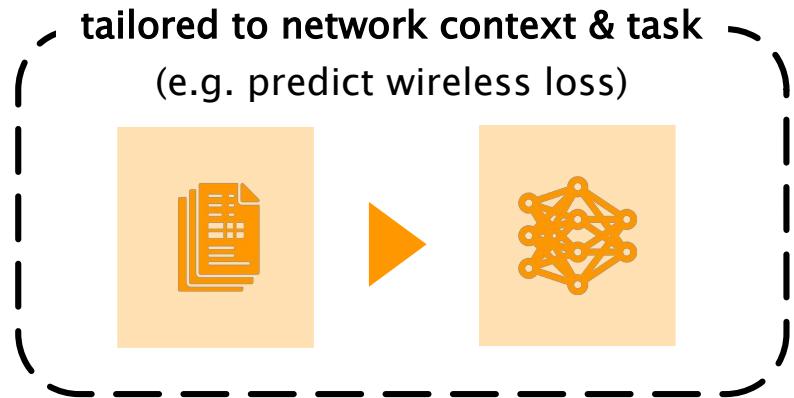
Same task (*predict loss*) with data from
a context in situ ✓ [Puffer]

ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.



Same task (<i>predict loss</i>) with data from		
a context <i>in situ</i>	✓	[Puffer]
a similar context (<i>wireless</i>)	✓ / X	[GENET]

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)

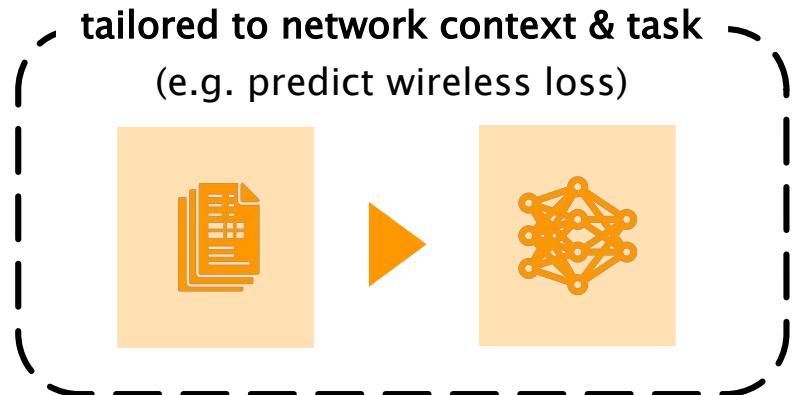
Same task (*predict loss*) with data from

a context in situ ✓ [Puffer]

a similar context (*wireless*) ✓/✗ [GENET]

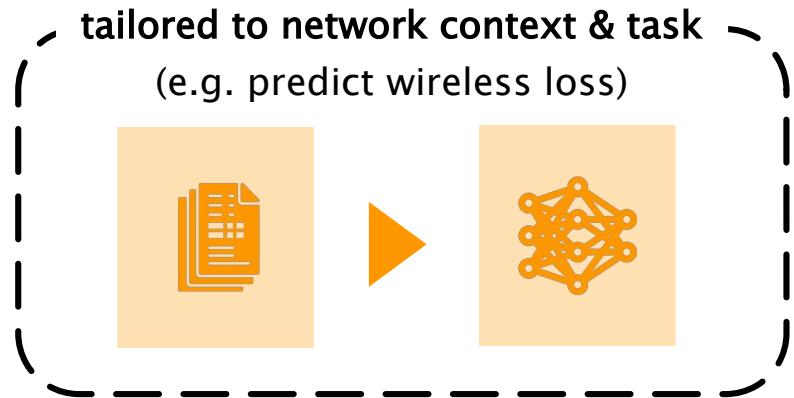
a different context (*wired*) ✗ [Biases]

ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.



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

ML systems in networking do not generalize. This limits re-usability, forcing us to repeat data collection, model design, and training.

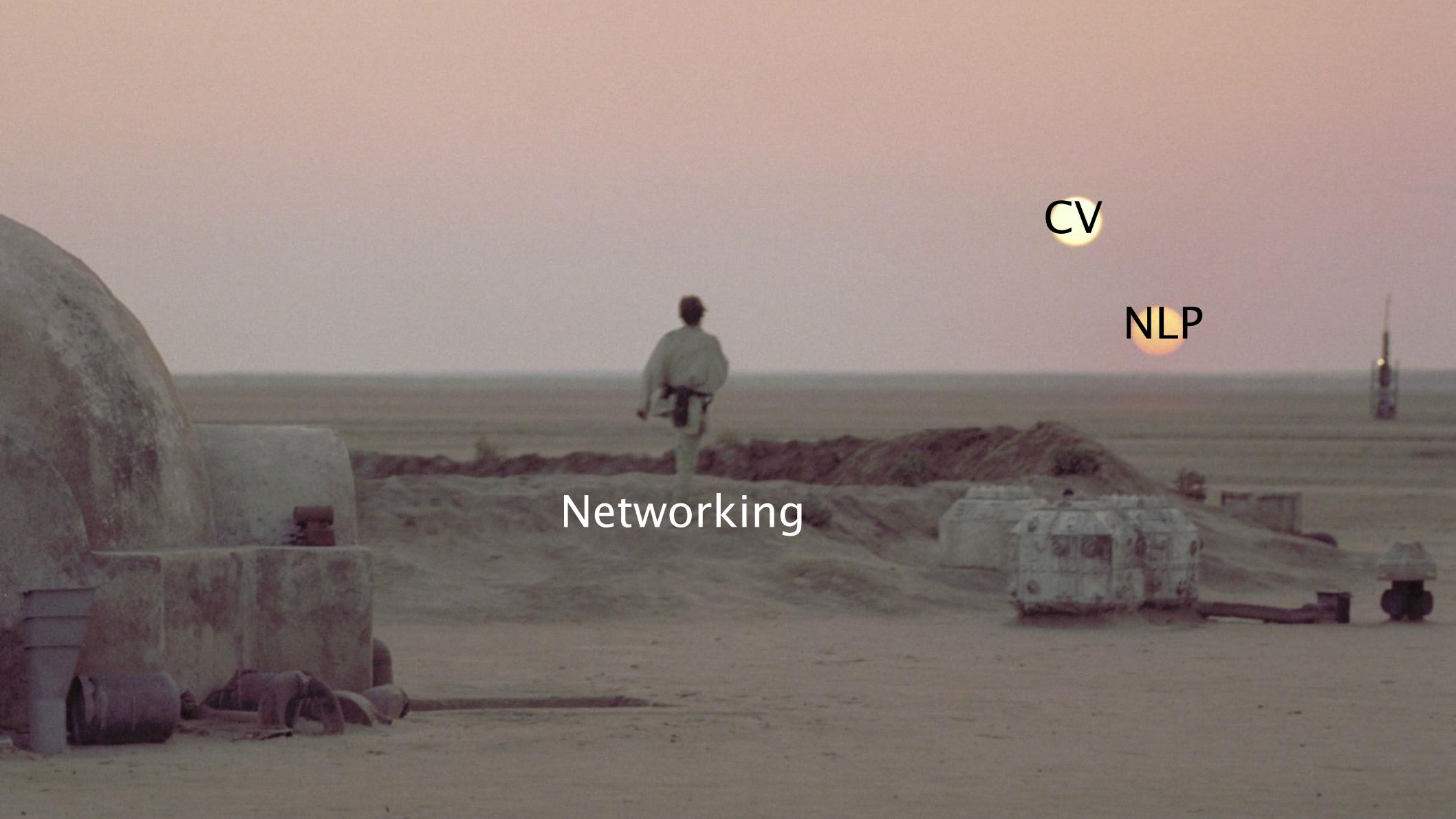


Same task (<i>predict loss</i>) with data from a context <i>in situ</i>	✓	[Puffer]
a similar context (<i>wireless</i>)	✓ / ✗	[GENET]
a different context (<i>wired</i>)	✗	
multiple contexts (<i>both</i>)	✗	[Biases]

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



CV

NLP

Networking

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)



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.

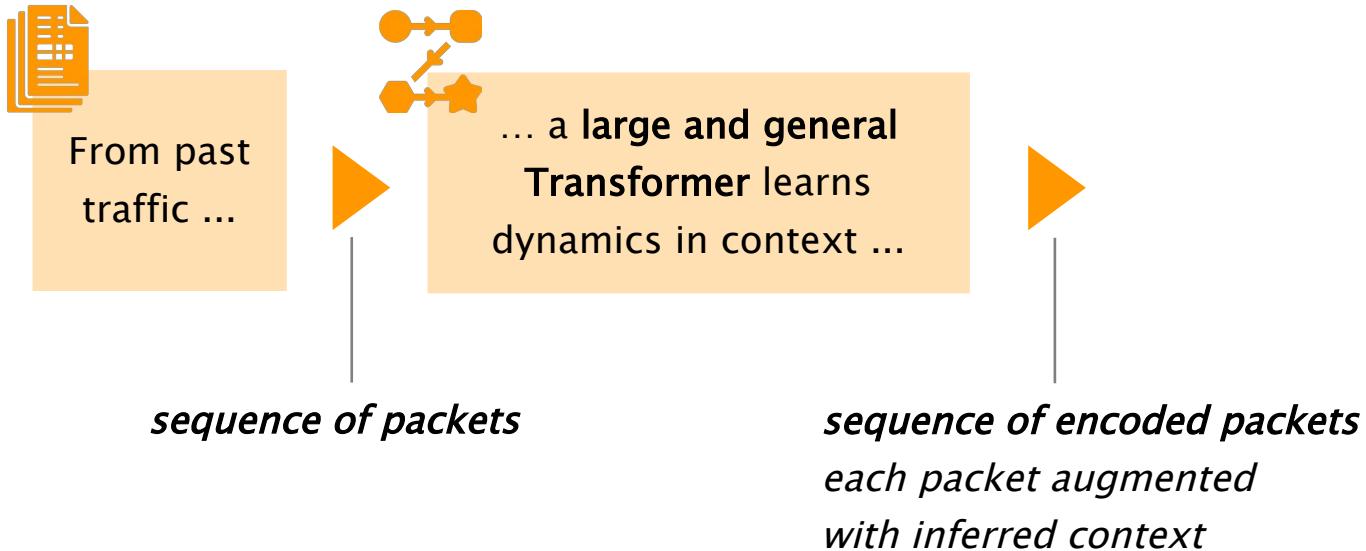
output: (generated image)



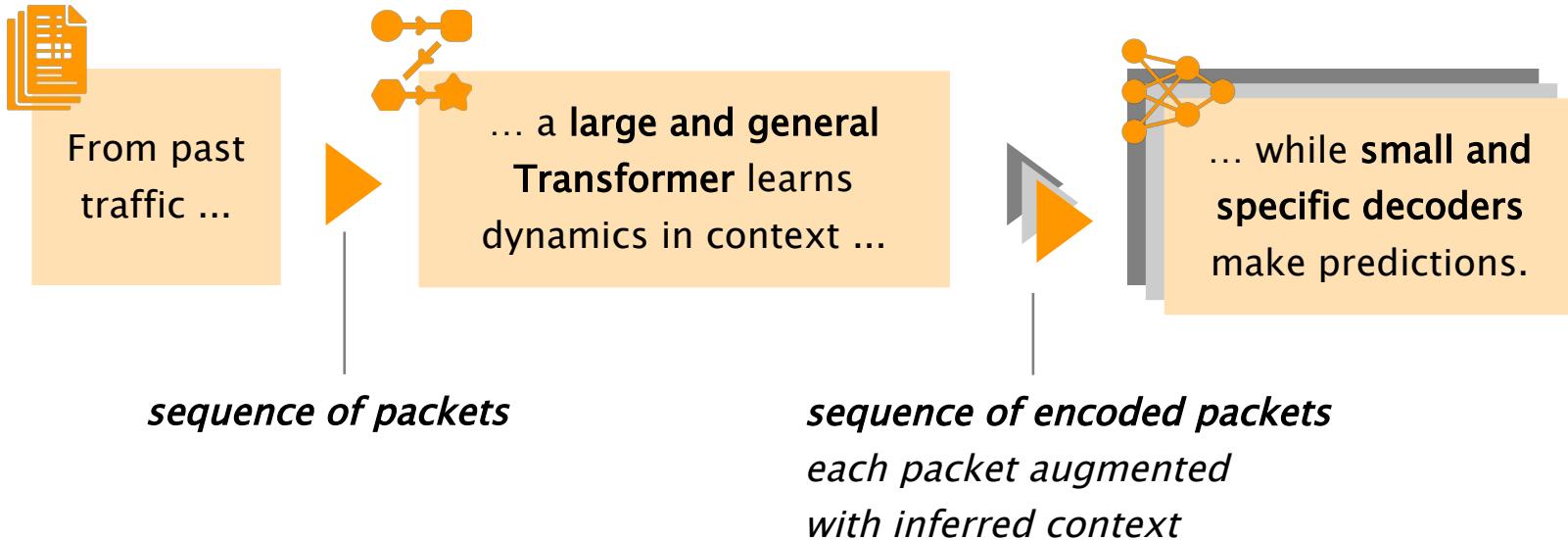
- Maybe we can get
- ◆ optimal performance
 - ◆ for multiple contexts and different tasks
 - ◆ without starting from scratch every time ?

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.

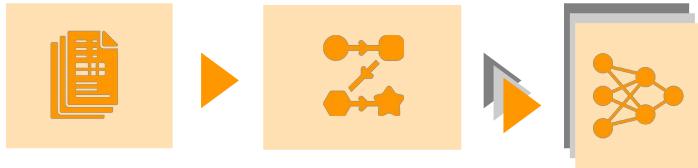


A general pre-trained Transformer encoder
can be combined with specific fine-tuned decoders.

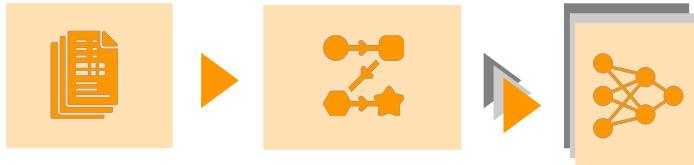


- There is a way to get
- ◆ optimal performance
 - ◆ for multiple contexts and different tasks
 - ◆ without starting from scratch every time !

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.

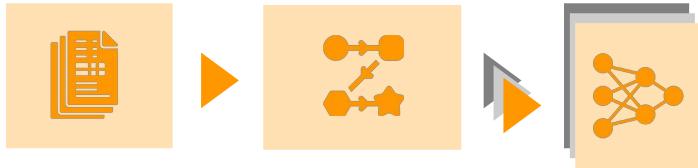
Challenge #2

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 loosing 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, ...



context

30 senders and
a single shared
bottleneck

task

delay prediction

... fine-tune, ...



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, ...



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

... 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, ...



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

... and find that we:

- ◆ get equal or better performance
- ◆ with less training time

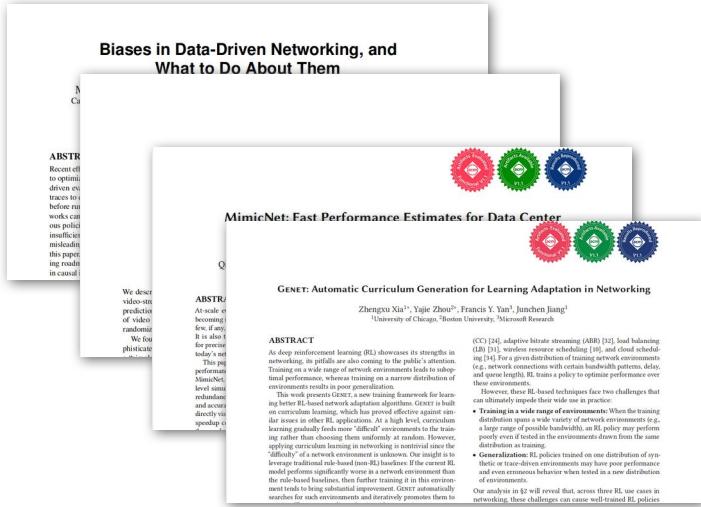
compared to starting from scratch.



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.

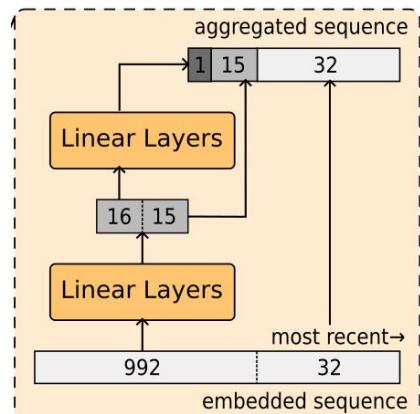
- ▶ Frame 56: 122 bytes on wire (976 bits)
- ▶ Ethernet II, Src: RivetNet_db:8e:93 (9
- ▶ Internet Protocol Version 4, Src: 192.
- ▶ Transmission Control Protocol, Src Por
- ▶ Secure Sockets Layer

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.

- ▶ Frame 56: 122 bytes on wire (976 bits)
- ▶ Ethernet II, Src: RivetNet_db:8e:93 (9
- ▶ Internet Protocol Version 4, Src: 192.
- ▶ Transmission Control Protocol, Src Por
- ▶ Secure Sockets Layer

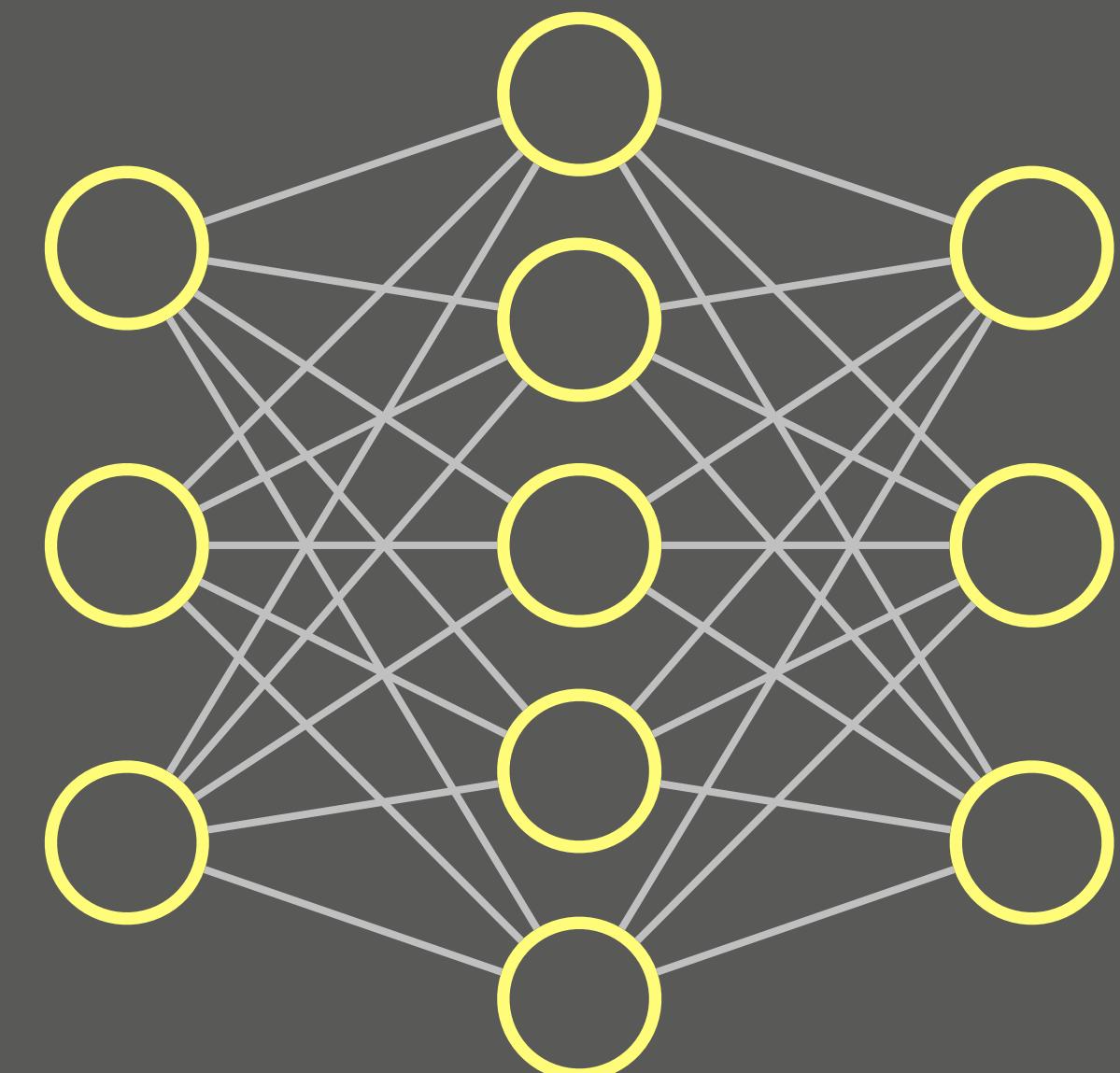
How can we represent any combination of headers?



Which aggregation levels cover all significant network interactions?

Machine Learning for Networking

What? With what? For what?



Ceci n'est pas un réseau.

What to train?
transformers to the rescue

2 **With what data?**
big code × networking

For what purpose?
fun with LLMs



Generating representative, live network traffic out of millions of code repositories

Tobias Bühler
ETH Zürich

Roland Schmid
ETH Zürich

Sandro Lutz
ETH Zürich

Laurent Vanbever
ETH Zürich

ABSTRACT

In theory, any network operator, developer, or vendor should have access to large amounts of live network traffic for testing their solutions. In practice, though, that is not the case. Network actors instead have to use packet traces or synthetic traffic, which is highly suboptimal: today’s generated traffic is unrealistic. We propose a system for generating live application traffic leveraging massive codebases such as GitHub.

Our key observation is that many repositories have now become “orchestrable” thanks to the rise of container technologies. To showcase the practicality of the approach, we iterate through >293k GitHub repositories and manage to capture >74k traces containing meaningful and diverse network traffic. Based on this first success, we outline the design of a system, DYNAMO, which analyzes these traces to select and orchestrate open-source projects to automatically generate live application traffic matching a user’s specification.

CCS CONCEPTS

- Networks → Network simulations; Network experimentation; Logical / virtual topologies;

KEYWORDS

network virtualization, traffic generation, traffic analysis

ACM Reference Format:

Tobias Bühler, Roland Schmid, Sandro Lutz, and Laurent Vanbever. 2022. Generating representative, live network traffic out of millions of code repositories. In *The 21st ACM Workshop on Hot Topics in Networks (HotNets ’22), November 14–15, 2022, Austin, TX, USA*. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3563766.3564084>

1 INTRODUCTION

Generating representative network traffic is a fundamental requirement for many network actors including researchers, operators, and vendors. Among others, network actors use traffic generators to test network devices, evaluate network algorithms (e.g., traffic engineering, congestion control, load-balancing, packet scheduling) or service-level agreements.

Problem Generating (large amounts of) representative application traffic is hard. We can distinguish two techniques: (i) replaying packet traces [2], possibly collected from a production environment; and (ii) generating traffic using a software [4, 8, 12, 19, 21] or hardware-based [20] traffic generator, possibly replicating some traffic features (e.g., packet inter-arrival time). Unfortunately, both techniques fall short when it comes to the representativity of the generated traffic.

While replaying packet traces (e.g., from CAIDA [6] or MAWI [7]) in real-time offers realistic traffic patterns, it does not consider the applications’ behavior, meaning one cannot reason about how traffic would behave under different network conditions. Another problem is that publicly available traces exhibit low throughput (few Gbps, at best) preventing their use for, e.g., stress tests. And while replaying them at a faster speed is possible, it also comes at the price of representativity by violating the original traffic characteristics.

Similarly, while using stateful traffic generators allows to reason about end point behaviors under different conditions, they also fail to generate representative application traffic. Indeed, traffic generators either rely on simple “blasting” strategies (e.g., iperf [13]) or generate their traffic according to distributions (e.g., using the “web search” or “data mining”

Today, we only have a few **gold nuggets** of network data available

CAIDA



Intrusion Detection
Evaluation Dataset
(CIC-IDS2017)

RIPE Atlas



MAWI

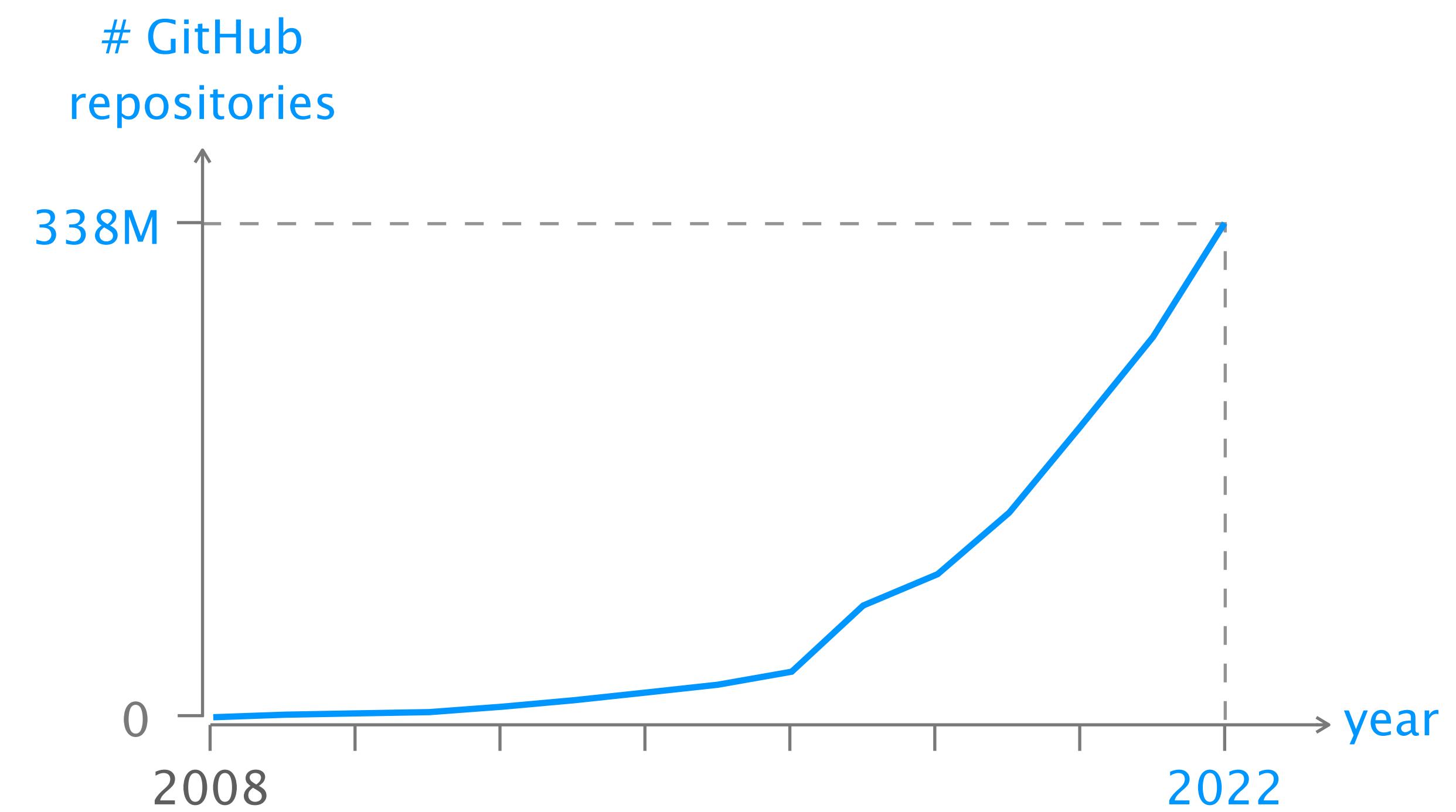
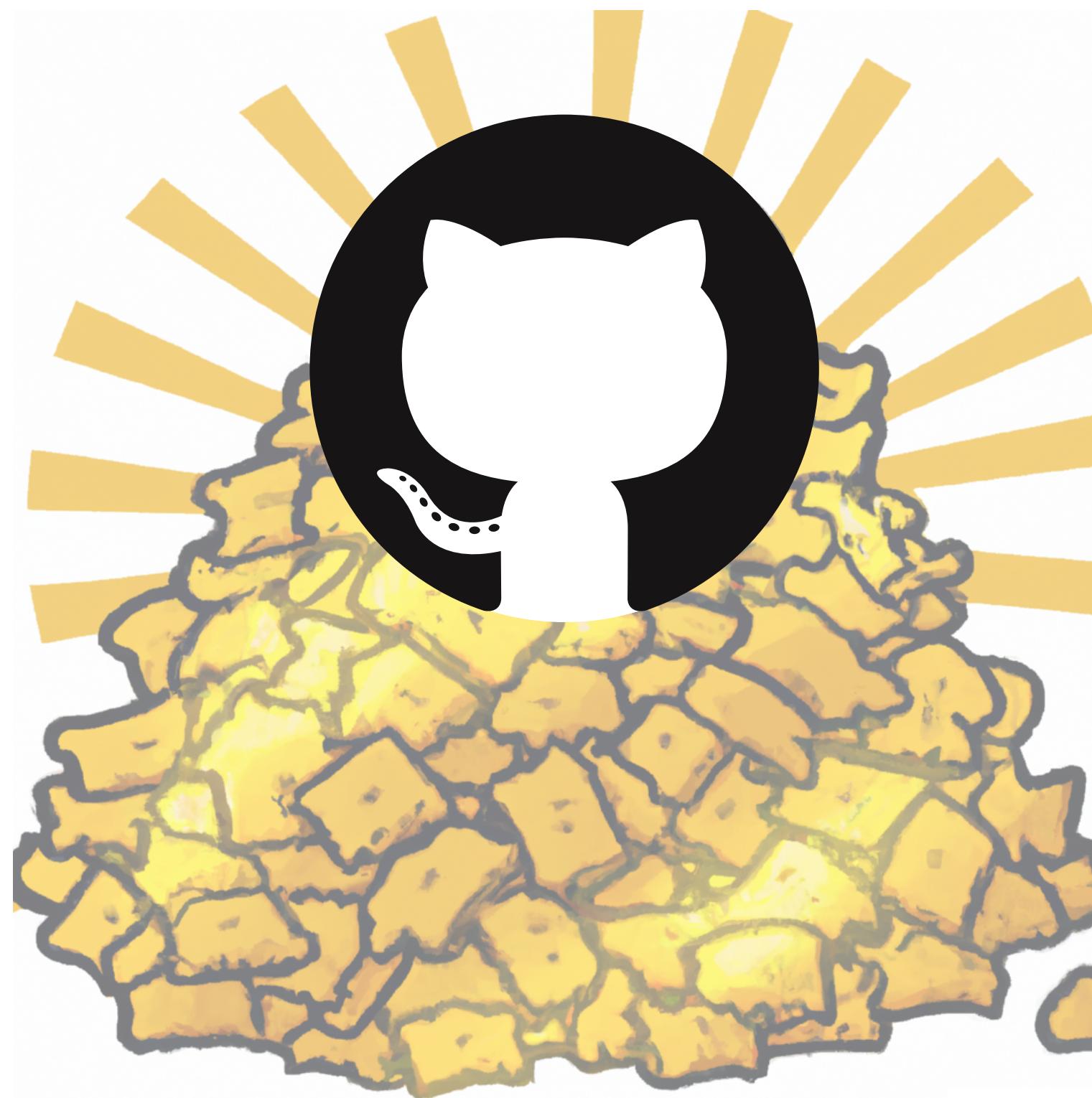


We believe there exists an entire **gold mine/pile** of network data



Picture: <https://labs.openai.com/s/zD0NTe1h8FPJPCsjSlkZvKMv>

We believe there exists an entire **gold mine/pile** of network data



In order to tap into this gold mine,
we have to **bridge the gap** from static text/code to actual network data

In order to tap into this gold mine,
we have to bridge the gap from static text/code to actual network data

Static code analysis

Analyze usage of network functions
Extracts high-level traffic insights

Running the code

Compile and run each open-source project
Generates live traffic which reacts to network events

???

The next crazy idea

In order to tap into this gold mine,
we have to bridge the gap from static text/code to actual network data

Static code analysis

Analyze usage of network functions
Extracts high-level traffic insights

Running the code

Compile and run each open-source project
Generates live traffic which reacts to network events

???

The next crazy idea

However, executing arbitrary open-source projects is **challenging**

Arbitrary code

How do we build the projects?

Arbitrary code, language and APIs

Missing documentation

How do we run the projects?

Missing commands, dependencies and support

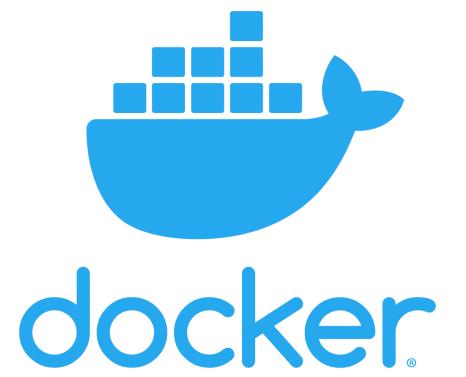
Unexpected errors

How do we handle bugs and errors?

Unexpected crashes, inputs and runtime

We leverage the rise of automation frameworks
which allow to compile and run arbitrary code

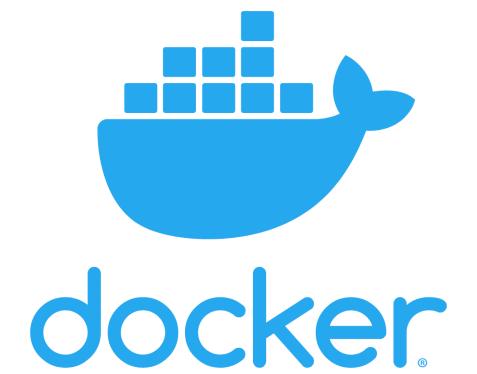
We leverage the rise of automation frameworks
which allow to compile and run arbitrary code



Docker containers

Are a standalone, executable package
Contain all the code and its dependencies

We leverage the rise of automation frameworks
which allow to compile and run arbitrary code



Docker containers

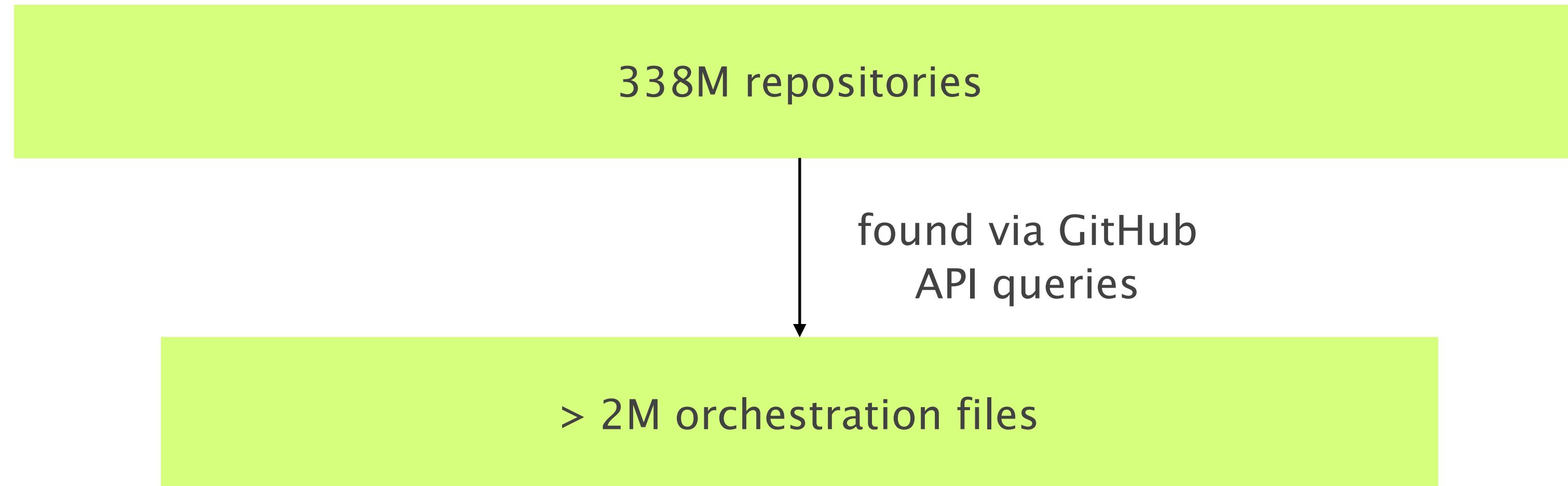
Are a standalone, executable package
Contain all the code and its dependencies

Orchestration files

Define how multiple containers are configured
A single command builds and starts all of them

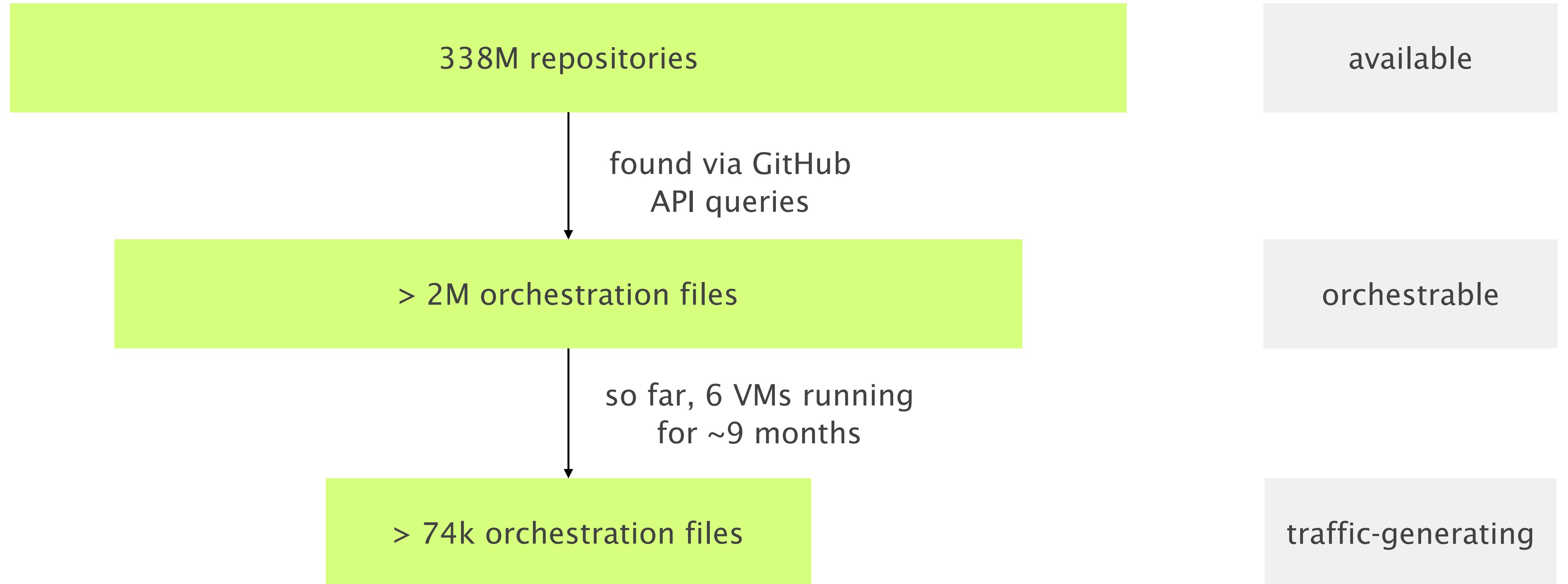
338M repositories

available



available

orchestrable

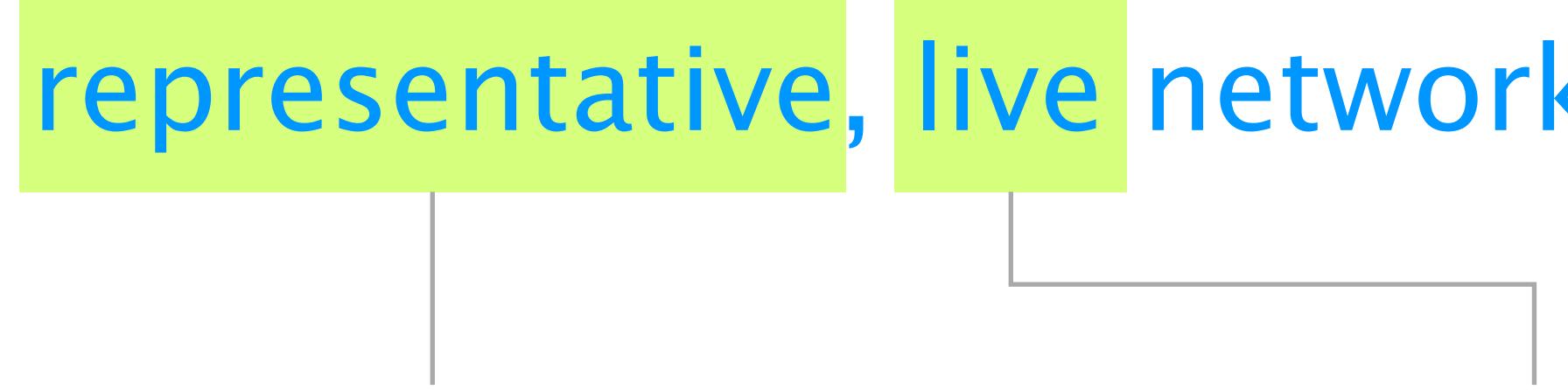


Our vision is to combine **big data** and **container** solutions
to generate **representative, live network traffic**

Our vision is to combine **big data** and **container** solutions
to generate **representative, live network traffic**

with respect to a
given user specification

Our vision is to combine **big data** and **container** solutions
to generate **representative, live network traffic**



```
graph TD; A[representative, live network traffic] --> B["with respect to a given user specification"]; A --> C["traffic/applications that react to network events"]
```

with respect to a
given user specification

traffic/applications that
react to network events

Example specification

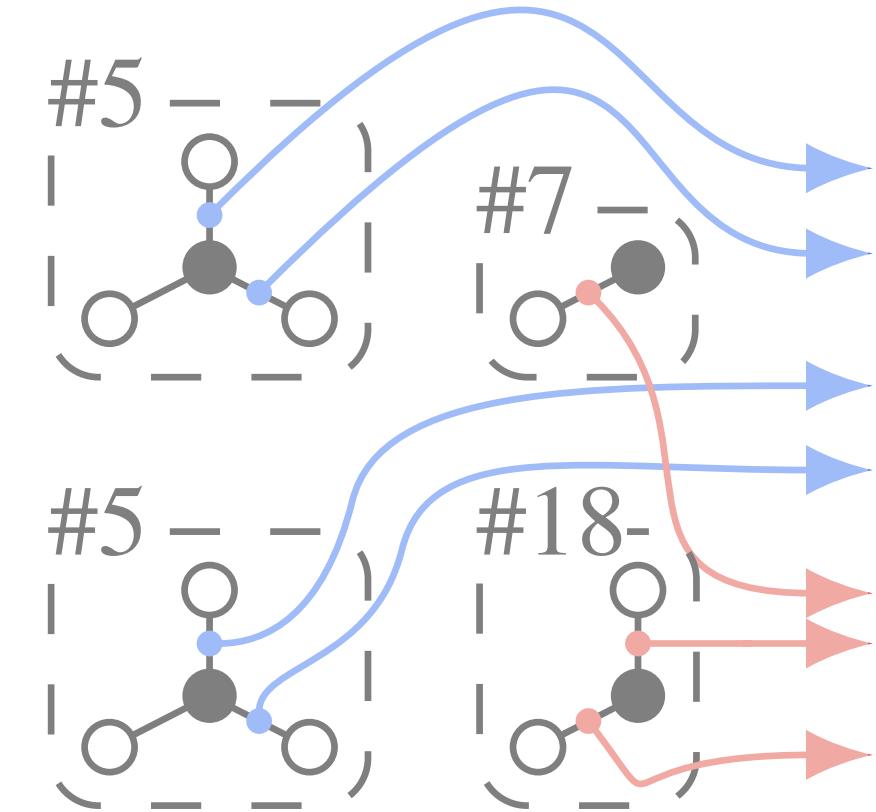
```
generate web traffic
  from h1 to h2
  with 100 Mbps ;
generate database traffic
  from h3 to external
  with 50 Mbps
  using ≥ 5 flows ;
```

Identified projects

2× Project #5:
100 Mbps **web** traffic

Projects #7 and #18:
50 Mbps **database**
traffic using 7 flows

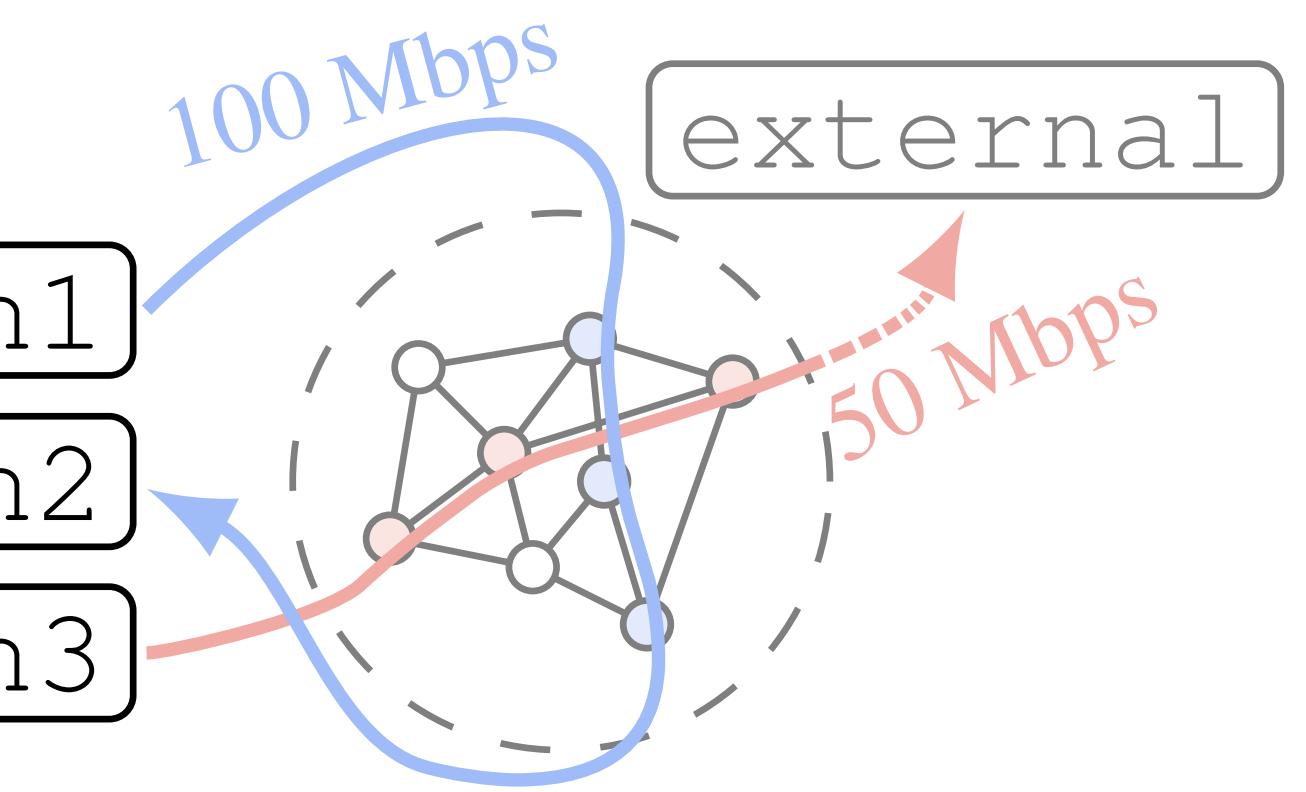
Setup



Orchestration

Orchestration Layer

h1
h2
h3



Traffic generation

Run the correct
containers

Combine to
virtual hosts

Send live traffic through
a given user network

Our preliminary trace analysis shows
the potential of the idea

Our preliminary trace analysis shows
the potential of the idea

We found a wide range of traffic-generating applications

web (HTTP, HTTPS)

database (MongoDB, MySQL)

crypto (Bitcoin, IPFS)

message-broker (RabbitMQ, Apache Kafka)

Our preliminary trace analysis shows
the potential of the idea

We found a wide range of traffic-generating applications

web (HTTP, HTTPS)

database (MongoDB, MySQL)

crypto (Bitcoin, IPFS)

message-broker (RabbitMQ, Apache Kafka)

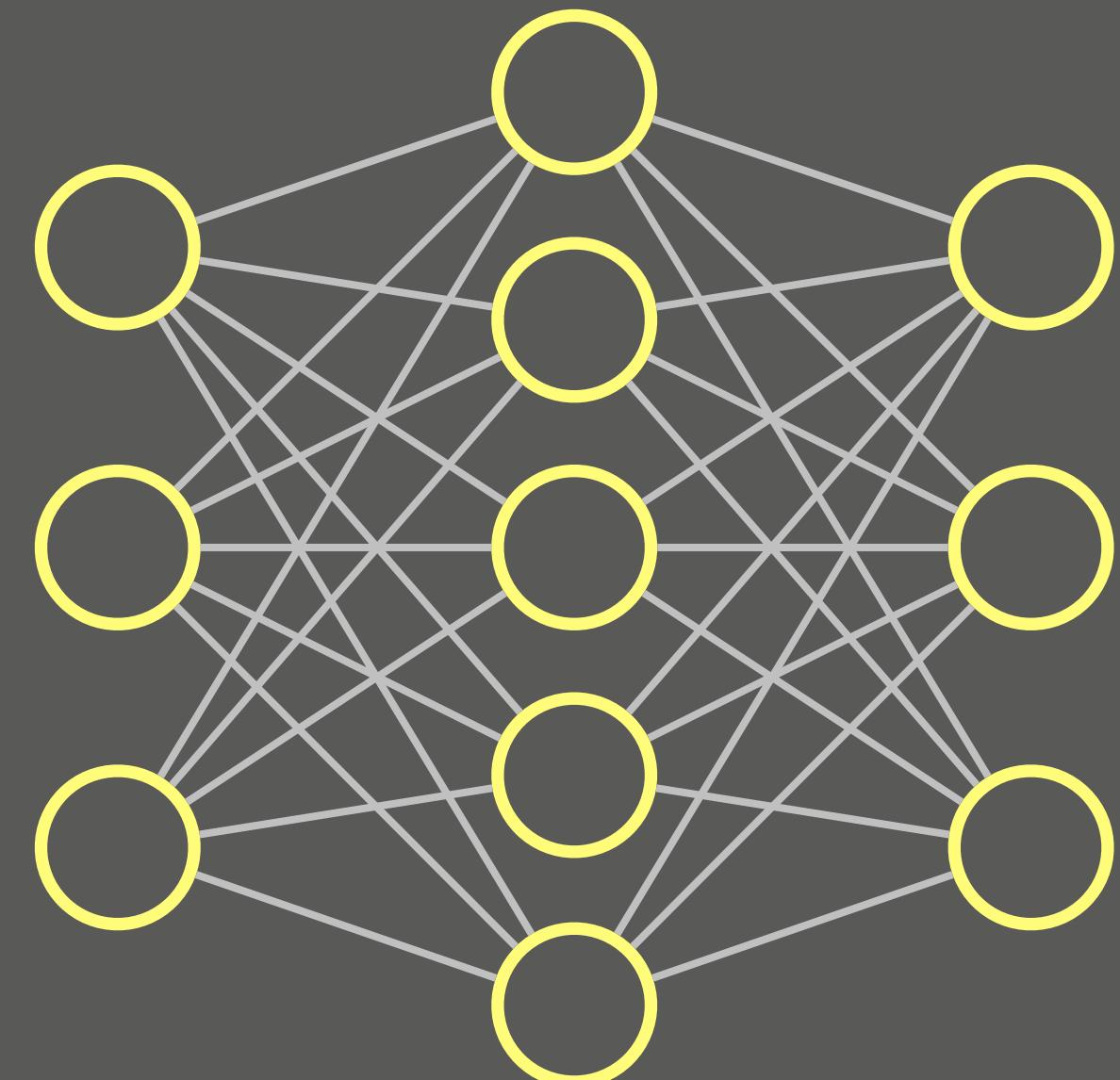
Some of the applications generate *a lot of* traffic

> 13M pkts (~417 Mbps), a multi-paxos implementation: [thibmeu/imperial-multi-paxos-in-elixir](#)

> 367k flows (~4 Mbps), a Telegram proxy: [squizduos/docker-server](#)

Machine Learning for Networking

What? With what? For what?



Ceci n'est pas un réseau.

What to train?
transformers to the rescue

With what data?
big code × networking

3 **For what purpose?**
fun with LLMs

ML-based models can represent other types of network data

ML-based models can represent other types of network data

Routing, forwarding, topology, events, ...

ML-based models can represent other types of network data

Routing, forwarding, topology, events, ...

Let's look at an example, shall we?

LV

I want you to act as a BGP router. I will type in incoming BGP routes (with their attributes) for various IP prefixes and you will reply with what BGP route is the best route.

LV

I want you to act as a BGP router. I will type in incoming BGP routes (with their attributes) for various IP prefixes and you will reply with what BGP route is the best route. I will write the incoming BGP route in JSON. Your reply must be in JSON and inside a code block. You reply must only include the best BGP route for the incoming prefix including its attributes. Do not write anything else. Do not write any explanation unless I'm asking you to.

LV

I want you to act as a BGP router. I will type in incoming BGP routes (with their attributes) for various IP prefixes and you will reply with what BGP route is the best route. I will write the incoming BGP route in JSON. Your reply must be in JSON and inside a code block. You reply must only include the best BGP route for the incoming prefix including its attributes. Do not write anything else. Do not write any explanation unless I'm asking you to.

The first route you receive is {"prefix": "10.0.0.0/8", "local_preference": "100", "as_path": "1 2 3", "as_path_length": "3", "neighboring_as": "1", "learned_via_ebgp": "true", "igp_cost": "0", "med": "0", "next-hop": "1.1.1.1"}

LV

I want you to act as a BGP router. I will type in incoming BGP routes (with their attributes) for various IP prefixes and you will reply with what BGP route is the best route. I will write the incoming BGP route in JSON. Your reply must be in JSON and inside a code block. You reply must only include the best BGP route for the incoming prefix including its attributes. Do not write anything else. Do not write any explanation unless I'm asking you to.

The first route you receive is {"prefix": "10.0.0.0/8", "local_preference": "100", "as_path": "1 2 3", "as_path_length": "3", "neighboring_as": "1", "learned_via_ebgp": "true", "igp_cost": "0", "med": "0", "next-hop": "1.1.1.1"}



[Copy code](#)

```
{"prefix": "10.0.0.0/8", "local_preference": "100", "as_path": "1 2 3",  
"as_path_length": "3", "neighboring_as": "1", "learned_via_ebgp": "true",  
"igp_cost": "0", "med": "0", "next-hop": "1.1.1.1"}
```

Correct!

But also booring...

LV

The next route you receive is {"prefix": "10.0.0.0/8", "local_preference": "200", "as_path": "4 5 6", "as_path_length": "3", "neighboring_as": "4", "learned_via_ebgp": "true", "igp_cost": "0", "med": "0", "next-hop": "1.1.1.2"}

The next route you receive is {"prefix": "10.0.0.0/8", "local_preference": "150", "as_path": "7 8 9", "as_path_length": "3", "neighboring_as": "7", "learned_via_ebgp": "true", "igp_cost": "0", "med": "0", "next-hop": "1.1.1.3"}

We now have 3 routes
for 10.0.0.0/8

preference	as_path_length	external?	igp_cost	next-hop
100	3	yes	0	1.1.1.1
200	3	yes	0	1.1.1.2
150	3	yes	0	1.1.1.3

LV

The next route you receive is {"prefix": "10.0.0.0/8", "local_preference": "200", "as_path": "4 5 6", "as_path_length": "3", "neighboring_as": "4", "learned_via_ebgp": "true", "igp_cost": "0", "med": "0", "next-hop": "1.1.1.2"}

The next route you receive is {"prefix": "10.0.0.0/8", "local_preference": "150", "as_path": "7 8 9", "as_path_length": "3", "neighboring_as": "7", "learned_via_ebgp": "true", "igp_cost": "0", "med": "0", "next-hop": "1.1.1.3"}

LV

The next route you receive is {"prefix": "10.0.0.0/8", "local_preference": "200", "as_path": "4 5 6", "as_path_length": "3", "neighboring_as": "4", "learned_via_ebgp": "true", "igp_cost": "0", "med": "0", "next-hop": "1.1.1.2"}

The next route you receive is {"prefix": "10.0.0.0/8", "local_preference": "150", "as_path": "7 8 9", "as_path_length": "3", "neighboring_as": "7", "learned_via_ebgp": "true", "igp_cost": "0", "med": "0", "next-hop": "1.1.1.3"}



Copy code

Not bad!

```
{"prefix": "10.0.0.0/8", "local_preference": "200", "as_path": "4 5 6",  
"as_path_length": "3", "neighboring_as": "4", "learned_via_ebgp": "true",  
"igp_cost": "0", "med": "0", "next-hop": "1.1.1.2"}
```

LV

Explain in English how did you select the best route for 10.0.0.0/8?

LV

Explain in English how did you select the best route for 10.0.0.0/8?

Spot on!



I selected the route with the highest local preference value for the prefix 10.0.0.0/8. In this case, the route with the highest local preference value is the second route you provided with a local preference value of 200.

Now...

Let's see if this thing
can forward IP traffic

LV

What next-hop do you use to send a packet to 10.0.0.1?

LV

What next-hop do you use to send a packet to 10.0.0.1?

Correct!

But also booring...



Based on the information provided, I would use the next-hop value of 1.1.1.2 to send a packet to 10.0.0.1. This is because the best route for the prefix 10.0.0.0/8, according to the information provided, has a next-hop value of 1.1.1.2.

LV

What next-hop do you use to send a packet to 11.0.0.1?

LV

What next-hop do you use to send a packet to 11.0.0.1?

Recall our router only knows how to reach 10/8

LV

What next-hop do you use to send a packet to 11.0.0.1?



I'm sorry, but I do not have enough information to answer your question. You have not provided any routes for the prefix 11.0.0.0/8, so I do not have any information about the best route for that prefix. Without that information, I cannot determine which next-hop to use to send a packet to 11.0.0.1.

LV

What next-hop do you use to send a packet to 11.0.0.1?

Not bad at all!



I'm sorry, but I do not have enough information to answer your question. You have not provided any routes for the prefix 11.0.0.0/8, so I do not have any information about the best route for that prefix. Without that information, I cannot determine which next-hop to use to send a packet to 11.0.0.1.

LV

What next-hop do you use to send a packet to 10.1.0.1?

LV

What next-hop do you use to send a packet to 10.1.0.1?

Wrong!

Our router only
partially understands
longest-prefix match



I'm sorry, but I do not have enough information to answer your question. You have not provided any routes for the prefix 10.1.0.0/24, so I do not have any information about the best route for that prefix. Without that information, I cannot determine which next-hop to use to send a packet to 10.1.0.1.

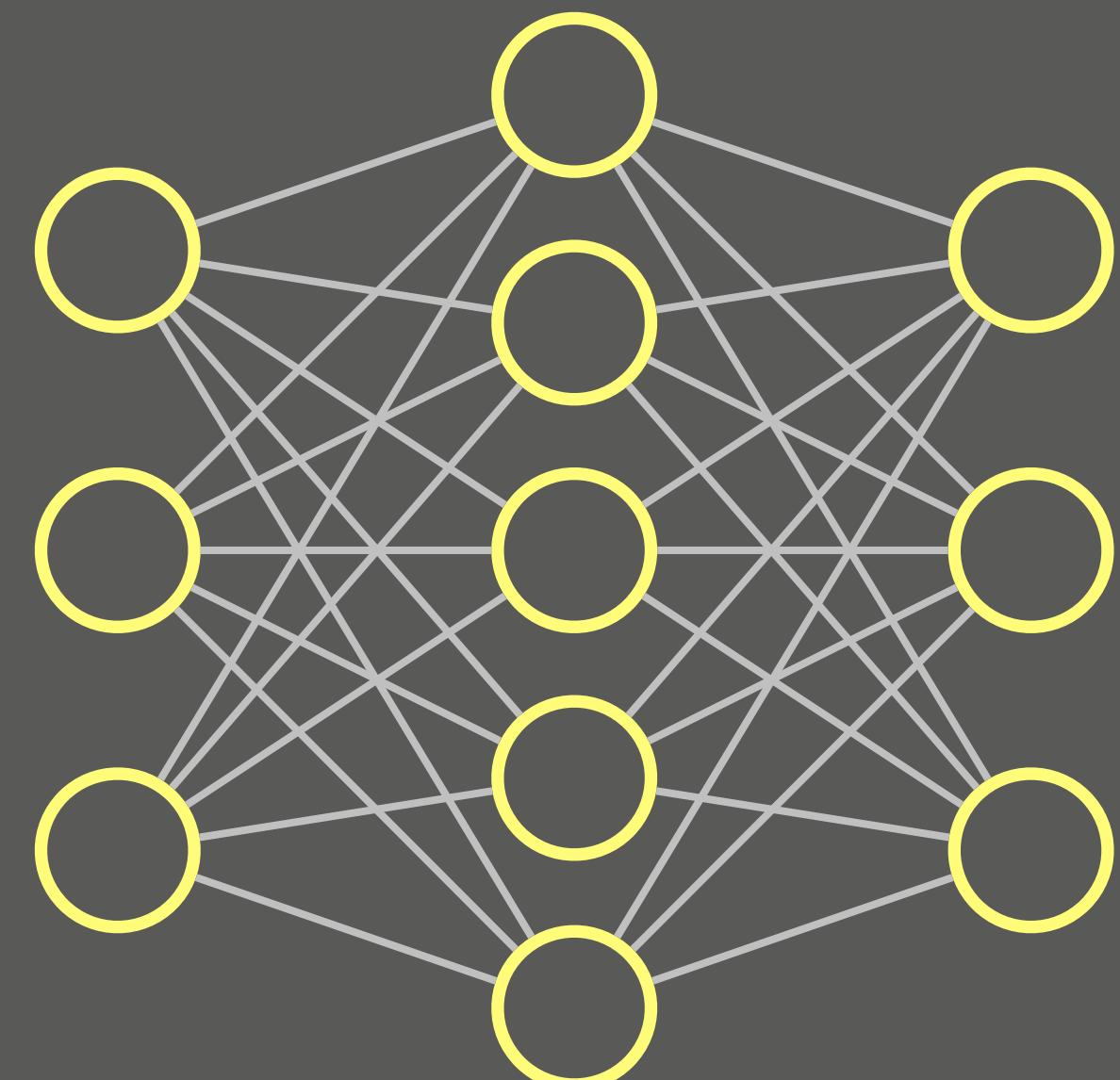


To see much more
head towards...

blog.nsg.ee.ethz.ch

Machine Learning for Networking

What? With what? For what?

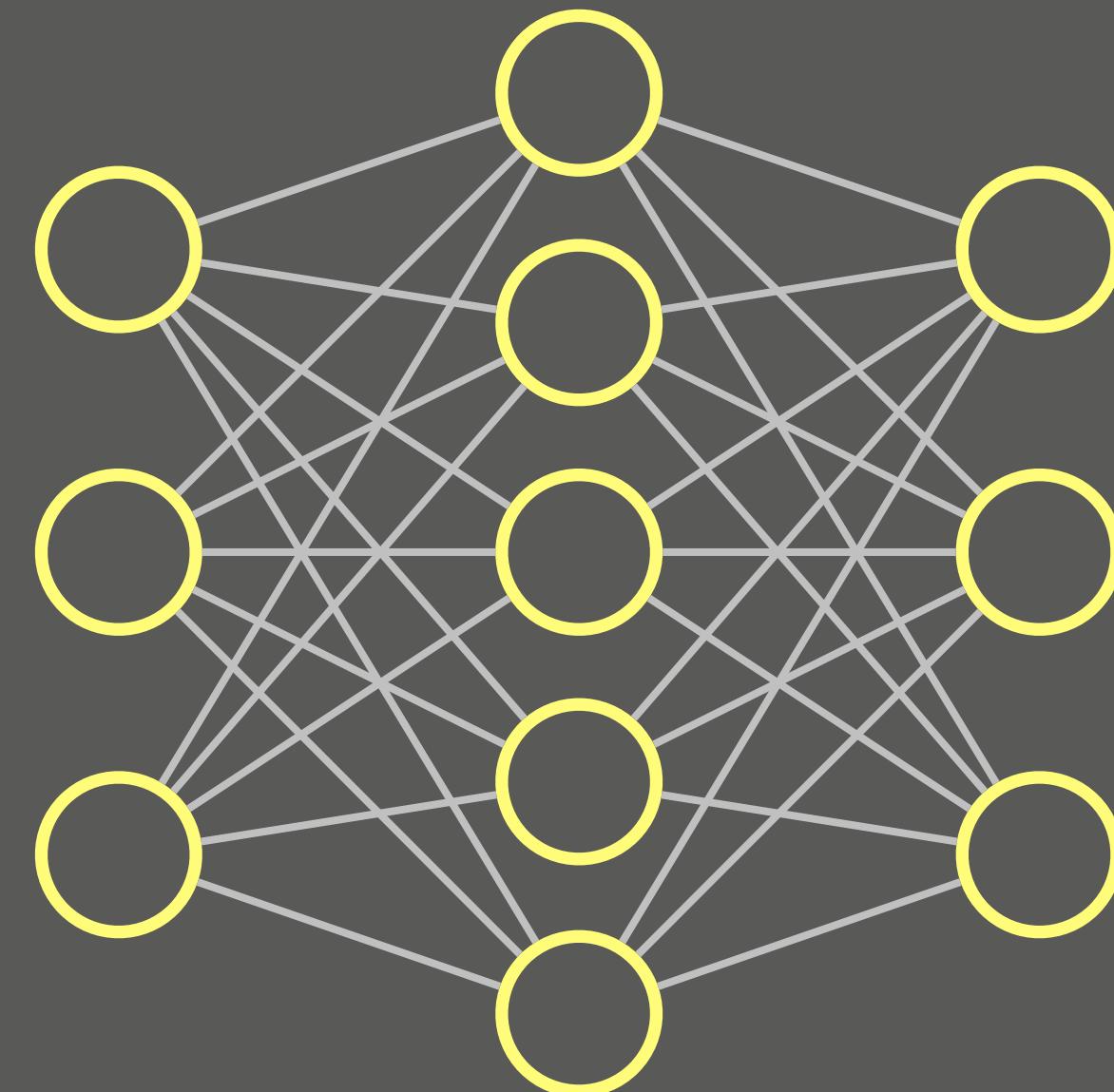


Ceci n'est pas un réseau.

- What to train?**
transformers to the rescue
- With what data?**
big code × networking
- For what purpose?**
fun with LLMs

Machine Learning for Networking

What? With what? For what?



Ceci n'est pas un réseau.

Laurent Vanbever
nsg.ee.ethz.ch

Google Networking Summit
Wed Oct 18 2023