Independent Study on Decentralized Internet Topology Learning

Zev Wilson
Hampshire College, UMass
Amherst, Massachusetts
zw22@hampshire.edu, zevwilson@umass.edu

***Abstract*—In order to create an internet more controlled by users than by corporations, we need to enable services to run directly on end-user devices in a peer-to-peer network. Core to this problem is the issue of finding nodes in *topologically ideal* places, meaning selections that optimize latency, reliability, bandwidth, anonymity, or storage locality. These are fundamentally optimization problems that require predicting network properties between arbitrary nodes in a network in order to optimize node selection.**

***Index Terms*—Transformers, Network Topology, Decentralization**

Contents

I. Background

Much research has been done in this area over the years, but as of the author's knowledge, no one has trained a medium-sized transformer model to learn a general prediction algorithm that can not only predict latency between two nodes, but also predict other topological features for a variety of applications, including conditioning on timestamp. This independent study investigates training a transformer directly on latency measurements in order to learn a variety of potentially useful prediction modes for node selection, in addition to reviewing the literature on how such a transformer could be trained in a decentralized fashion.

II. Proposed Objectives

The objective of this independent study is to work towards creating a system with *all* of the following properties:

1) Ability to take into account time when predicting link properties (e.g. diurnal patterns and outages).
2) Ability to predict link properties even without a connection to the network (offline, cold-start inference).
3) Ability to cheaply calibrate the predictive model with additional local measurements (few-shot adaptation).
4) Ability to detect and resist significant node failures or malicious behavior (e.g. poisoning, Sybil, Byzantine).
5) Ability for nodes to limit training data discovery (membership or relationship inference from weights or outputs).
6) Ability to scale prediction accuracy based on end-user hardware capabilities (distilled tiers, adaptive context/compute).

A transformer model trained on heterogeneous measurement sequences and in-context learning could potentially satisfy properties 1-3. It is unclear how properties 4-6 could be easily satisfied, however. Thus the goal of this independent study is to train a transformer to address goals 1-3 and then do a literature review to map out how properties 4-6 could be achieved.

III. Design

A. Transformer

The transformer trained is a decoder-only transformer using Google's `MaxText` library. A decoder-only solution is used because this application only has one sequence type and is expected to be used autoregressively from the first measurement.

Architecturally, the target model is a medium-sized network (on the order of 80-100M parameters) with:

- An embedding dimension of approximately 640 and a vocabulary size of 267 (11 role tokens + 256 byte tokens) shared between input and output embeddings.
- 20 transformer decoder layers with multi-head self-attention (around 10 heads of width 64) and MLP blocks of width roughly 2048 (a 3.2x expansion over the embedding size).
- Rotary positional embeddings (RoPE), using MaxText defaults.
- A context window of 1024 tokens, which corresponds to roughly 25-60 measurements depending on IPv4/IPv6 mix and timestamp encoding under the tokenization scheme described below.

This design intentionally favors depth over width and uses a smaller-than-usual MLP ratio so that the model learns general routing and topology rules instead of memorizing individual (src, dst, time) triples.

B. Data

The data to be trained on is roughly 200M measurements directly downloaded and parsed from the RIPE Atlas project's 'daily dumps' page split 90/10 into train/test sets. This 200M measurement dataset is a random sample drawn from a much larger raw corpus (35 billions measurements comprising roughly a month of RIPE Atlas operations). [1]

The RIPE Atlas is a project that contains tens of thousands of probes and a smaller set of anchors. Probes run measurements (including pings) to other probes, anchors, and user-specified targets, while anchors are a well-provisioned subset of probes intended as stable reference endpoints for measurement campaigns.

For this project, the raw JSON data is preprocessed into a Parquet snapshot. Only the following fields are actually used:

```
event_time: timestamp       # Event timestamp (unix epoch
seconds)
src_addr: string            # Source IP address (IPv4 or IPv6)
dst_addr: string            # Destination IP address
ip_version: int64           # 4 or 6
rtt: double                 # Round-trip time in milliseconds
(-1.0 = failed/filtered probe in preprocessing)
```

The resulting dataset has the following characteristics (for the current snapshot):

- Approximately 200M measurements (2.1 GB on disk) sampled at random from about a month of measurement data (1TB).
- Date range on the order of late June to late July (snapshot-specific), so the model sees both short-term and medium-term temporal variation.
- Mixed IPv4/IPv6 coverage with a roughly 60%/40% split between IPv4 and IPv6 rows.

C. Tokenization

In order to create a flexible prediction engine for internet measurements, we need to create a language that can encode sequences of measurements and allow for the transformer to learn various conditional slices of a given measurement. We have a base token language, and a way of composing them to encode sequences of measurements in a way that allows the transformer to learn how to predict from various slices.

The token language is defined as follows:

Token	::=	<MeasurementStart>	Measurement boundary
	|	<SrcIPv4> | <SrcIPv6>	Source IP markers
	|	<DstIPv4> | <DstIPv6>	Destination IP markers
	|	<TimestampAbs>	Absolute timestamp marker
	|	<TimestampDelta1> | <TimestampDelta4>	Delta timestamp markers
	|	<RttStart> | <ThroughputStart>	Result value markers
	|	<Failed>	Connection failed marker
	|	<Byte0>, ..., <Byte255>	Data bytes (11-266)
U8	::=	<Byte0> | ... | <Byte255>	Any single byte
Float16	::=	U8 U8	Mantissa and Exponent
SrcIp	::=	<SrcIPv4> U8^4	IPv4 Address
	|	<SrcIPv6> U8^{16}	IPv6 Address
DstIp	::=	<DstIPv4> U8^4	IPv4 Address
	|	<DstIPv6> U8^{16}	IPv6 Address
Timestamp	::=	<TimestampAbs> U8^8	Absolute (First measurement)
	|	<TimestampDelta1> U8	Delta $< 256s$
	|	<TimestampDelta4> U8^4	Delta $\geq 256s$
Result	::=	<RttStart> Float16	Successful probe (RTT)
	|	<ThroughputStart> Float16	Throughput (Future)
	|	<Failed>	Failed probe
Field	::=	SrcIp | DstIp | Timestamp | Result	Shuffled per measurement
Meas	::=	<MeasurementStart> $\text{Field}^{3..4}$	3 or 4 fields per record
Context	::=	Meas^+	Sequence of measurements

In the implementation, this grammar is realized as a compact 267-token vocabulary:

- 11 role tokens that mark measurement structure (`<MeasurementStart>`, IP family and direction, timestamp/RTT markers, and failure/throughput markers).

- 256 byte tokens (`<Byte0>`-`<Byte255>`) used to encode all numeric values as big-endian byte sequences.

Each measurement is serialized as `<MeasurementStart>` followed by three or four field blocks (source IP, destination IP, RTT or failure indicator, and optionally a timestamp). During training, these field blocks are shuffled per measurement so the model is forced to learn the full joint distribution over fields rather than relying on a fixed field order. The `<ThroughputStart>` marker is reserved for future measurement types; the current dataset uses RTT or `<Failed>` only.

Under this scheme, a typical IPv4 measurement with a timestamp uses about 16-23 tokens depending on whether the timestamp can be encoded as a 1-byte delta from the previous timestamp; IPv6 measurements are longer but follow the same pattern. Compared to the earlier, more verbose tokenization, this reduces sequence length by roughly a factor of 2-3, allowing a 1024-token context window to cover many more measurements and enabling stronger in-context "network localization" from recent observations.

The RTT value is encoded as an unsigned 2-byte exponent/mantissa in microseconds (5-bit exponent, 11-bit mantissa), giving a wide dynamic range with sub-percent relative precision for typical RTT values. RTTs are converted from milliseconds to microseconds during tokenization to preserve resolution. Timestamps are delta-encoded in seconds: the first measurement uses an absolute 64-bit timestamp, then deltas use 1 byte for gaps under 256 seconds and 4 bytes otherwise (unsigned). All numeric values are stored as big-endian bytes.

D. Rationale

The goal of training this transformer is to create a sort of "foundation model" to predict various useful properties of internet links. Some useful properties might be:

- Given `<SrcIp>` and `<DstIp>`, and optionally `<Timestamp>`, what is the estimated distribution of `<Rtt>` or `<Throughput>` for this potential connection?
- Given `<SrcIp>` and `<Rtt>`, which `<DstIp>` values are most likely to be closest?
- Given an IP prefix, which IPs are most likely to be online?
- What is the IPv6 / IPv4 counterpart that a given `<SrcIp>` likely corresponds to?
- What is the distribution of `<DstIp>` values that a `<SrcIp>` is most likely to talk to?
- Given `<SrcIp>`, `<DstIp>`, and `<Rtt>` (or `<Throughput>`), what is the most likely `<Timestamp>`?
- What `<SrcIp>` / `<DstIp>` / `<Timestamp>` combinations are most common?

- Given `<Rtt>` and `<Timestamp>`, what connections (`<SrcIp>` + `<DstIp>` pairs) are most likely to fit these properties?

Multiple usages of these kinds of predictions could enable not only more efficient routing, but if further trained on the properties of full anonymous paths, this might allow for both path selection and estimation of path anonymity, by either sampling paths from the model, or given a path estimating how likely it is to have been generated.

All the above prediction modes can be achieved by varying which fields are present and by randomizing field order within each measurement when encoding sequences.

a) *Privacy Concerns:*

Because the model is trained on real-world network measurements, there is a risk of memorizing rare or sensitive traffic patterns. The architectural choices above (medium model size, smaller MLP ratio, and emphasis on learning aggregate routing structure) are partly motivated by a desire to generalize rather than memorize individual flows, but a more thorough privacy analysis (including membership inference and extraction risks) is left to the second half of this paper discussing the space of decentralized training algorithms.

b) *Related Work (Brief):*

Latency prediction and topology learning have a long history in network coordinate systems (embedding nodes into low-dimensional spaces), matrix completion or low-rank approaches for RTT estimation, and more recent ML models that incorporate ASN/geo features or graph structure. This work differs by training an autoregressive transformer directly on measurement sequences, enabling multi-field conditional predictions (latency, timestamp, failure likelihood) from a single model.

E. Size & Training

The model is in the 100M parameter range, which should ideally be large enough to capture rich IP and routing structure but small enough to be trainable on a single modern accelerator.

Training is formulated as standard next-token prediction over tokenized measurement sequences with a context length of 1024. Tokenization is done at runtime using a loader built on Google's `grain` library. The dataset is pre-grouped by source IP into probe-centric rows. Each row is sampled into windows (window size drawn from a log-uniform distribution) so the model sees both short- and long-range temporal patterns.

F. Model Architecture

The model is a decoder-only transformer whose primary goal is to learn the joint distribution of a measurement `(src_ip, dst_ip, rtt, timestamp)` conditional on past measurements. Key architectural choices are:

- A relatively deep stack of decoder layers (20) to support multi-step reasoning tasks such as "RTT -> IP search" and hierarchical IP structure learning (from coarse prefixes down to specific subnets).
- A moderate embedding size (640) with standard 64-dimensional attention heads to keep the model expressive without oversizing the representation for a 267-token vocabulary.
- A smaller MLP expansion ratio (3.2x) than is typical in general-purpose language models, which reduces pure memorization capacity and encourages the network to represent reusable routing and geography patterns.
- Rotary positional embeddings (RoPE), matching the MaxText configuration used for training.
- A context window of 1024 tokens so the model can condition on tens of recent measurements from the same vantage point and thereby infer location, connection type (residential vs datacenter), and other latent properties via in-context learning.

These are mostly arbitrary choices based on intuition since I don't have any data as to what degree parameters affect performance on this dataset.

IV. Runtime Data Processing

The goal is to train a transformer in a somewhat reminiscent manner as to how it might be trained in a federated environment. The dataset also needs to be preprocessed into a format that can efficiently feed a high-speed GPU like an A100/H100/B200.

To do this, we use DuckDB to group by source IP address, sort by time, and serialize probe-centric rows into ArrayRecord. Rows are capped at roughly 8 MB so large probes split across multiple rows. Each row stores compact measurements plus metadata (time span and first/last timestamps), enabling random access and high reuse per I/O.

During training-time sampling, each row yields K contexts where K = min(ceil(n/30), 16). Small rows use all measurements, while large rows sample a log-uniform time window and then subsample measurements to hit the target length while preserving temporal order. Field order within each measurement is randomized.

Training alternates between three timestamp modes (roughly 40/30/30 weighting):

- **Full timestamp:** all measurements in a window include timestamps and are ordered temporally, allowing the model to learn diurnal and long-range temporal patterns with efficient delta encoding.
- **No timestamp:** timestamps are omitted and measurements are randomly shuffled, forcing the model to learn atemporal network topology (e.g., geographic and ASN structure) that can be applied when timestamps are unavailable.
- **Mixed timestamps:** 10-90% of measurements drop timestamps; timestamped measurements stay ordered while untimestamped measurements are shuffled and then interleaved, teaching the model to use temporal information when available but remain robust to missing or partial timing data.

V. Training Details

Training uses the AdamW optimizer with a cosine learning rate schedule and a brief warmup period. A representative configuration is:

- Learning rate on the order of `1e-4` with a brief warmup (e.g., 20 steps in pilot runs; longer runs will scale warmup with total steps) followed by cosine decay.
- Per-device batch size 128 with sequence length 1024; sequence packing is enabled when supported to reduce padding waste.
- Dropout of about 0.1 and weight decay of about 0.01 for regularization.
- bfloat16 for parameters and activations to match modern accelerator hardware.

The primary metric during training is token-level cross-entropy on held-out measurements, with downstream evaluation focusing on latency prediction accuracy and generalization to unseen IP pairs and network conditions.

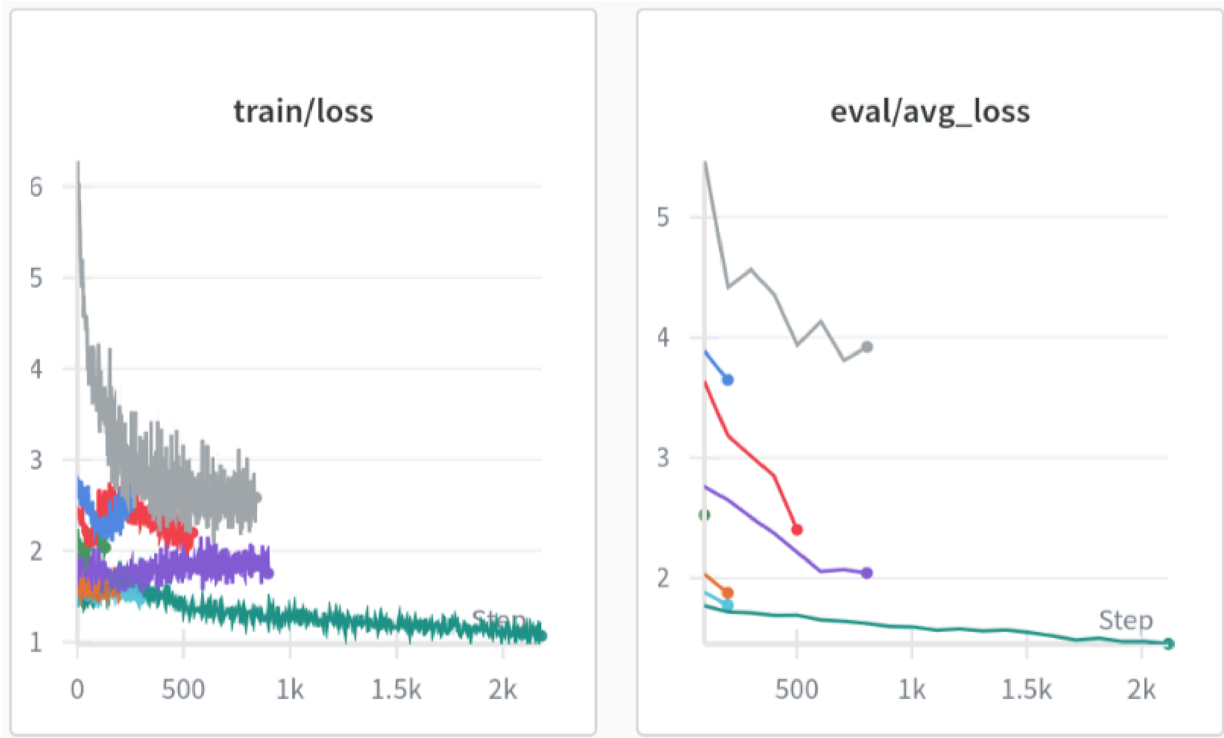

Fig. 1: Multiple rounds of training, continuing from previous model checkpoints

VI. Evaluation & Metrics

Evaluation focuses on three core checks that should intuitively communicate the utility of the transformer approach. This includes a visualization of the hypothesis that including timestamps makes for better predictions, and a comparison of model predictions to real-world sampling from a held-out source IP not seen in training. Additional baselines, metrics, and robustness tests are part of the evaluation plan and ongoing work.

Timestamp vs no-timestamp RTT likelihood: compare the log-probability of the correct RTT tokens with and without timestamp context. This is meaningful because it isolates whether temporal information actually sharpens the RTT distribution without changing the IP conditioning.

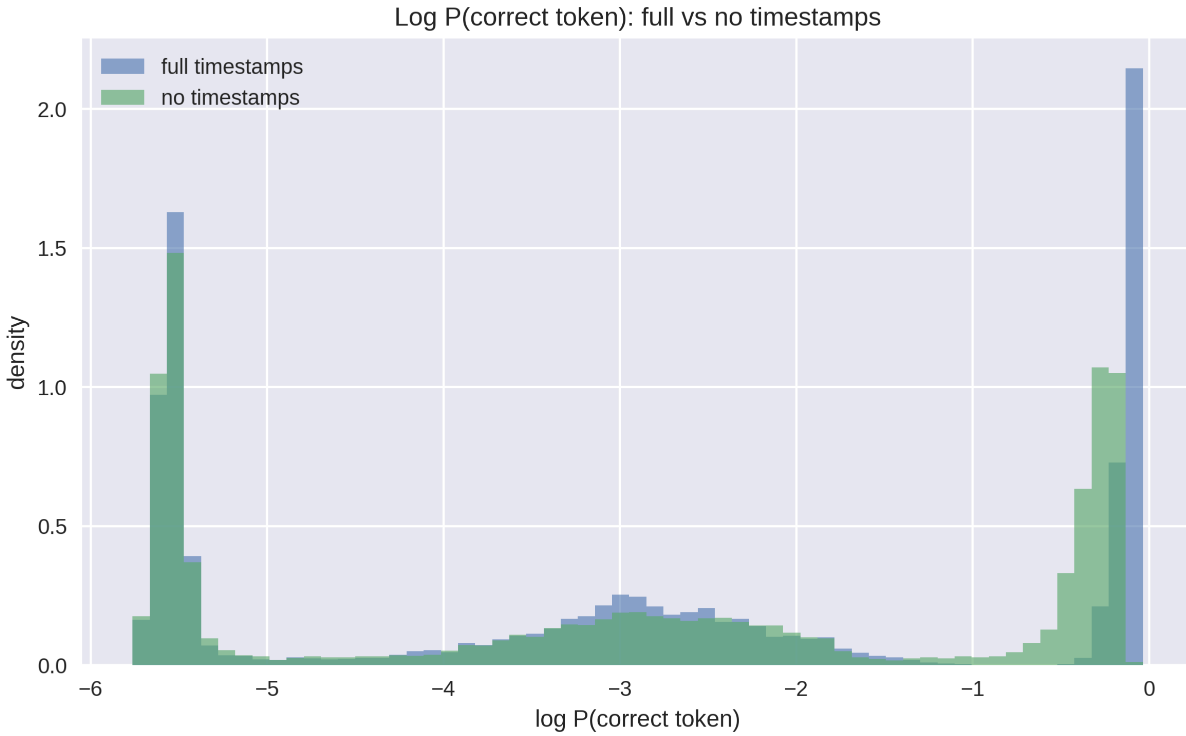

Fig. 2: Log P(correct RTT tokens) for full timestamps vs no timestamps.

Live ping distribution match: This eval compares real RTT histograms against model samples for a fixed set of targets and report KL divergence. It gives a good sense as to how calibrated the model is in predicting specific latency for both anchor and non-anchor nodes.

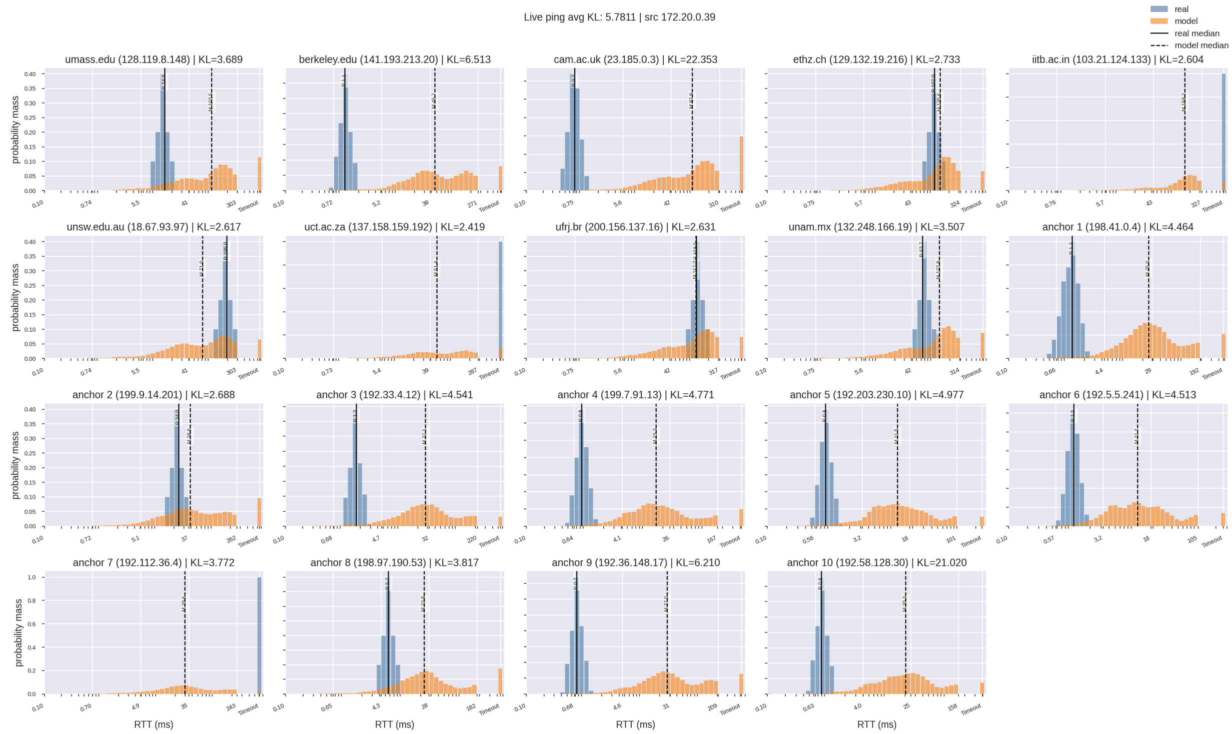

Fig. 3: Live ping RTT distributions for real vs model samples with per-target KL.

Destination IP Sampling: This eval samples a selection of IP addresses given a specific latency target and check to see how accurate the model's prediction of IP addresses was to real latency measurements. As we can see, the current model isn't very good at this just yet. This is confirmed in Figure 5 as destination IPs are the worst the network is at predicting. (This feels unsurprising given that there are not very many destination IPs in this dataset, they are all anchors).

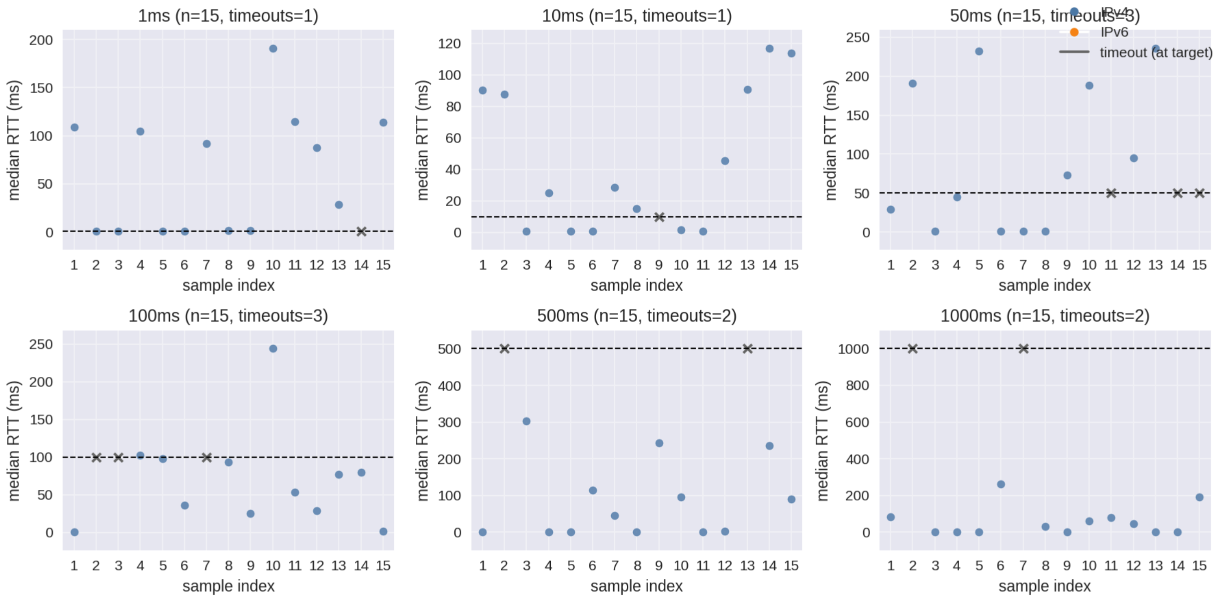

Fig. 4: Real measured latency differences from target latency of sampled destination IPs

Prediction-mode accuracy summary: This eval rotates which field is predicted last and measure mean correct-token probability by group. This shows which fields are "easier" to learn than others / have less average entropy, at least at this model size, training time, and dataset size.

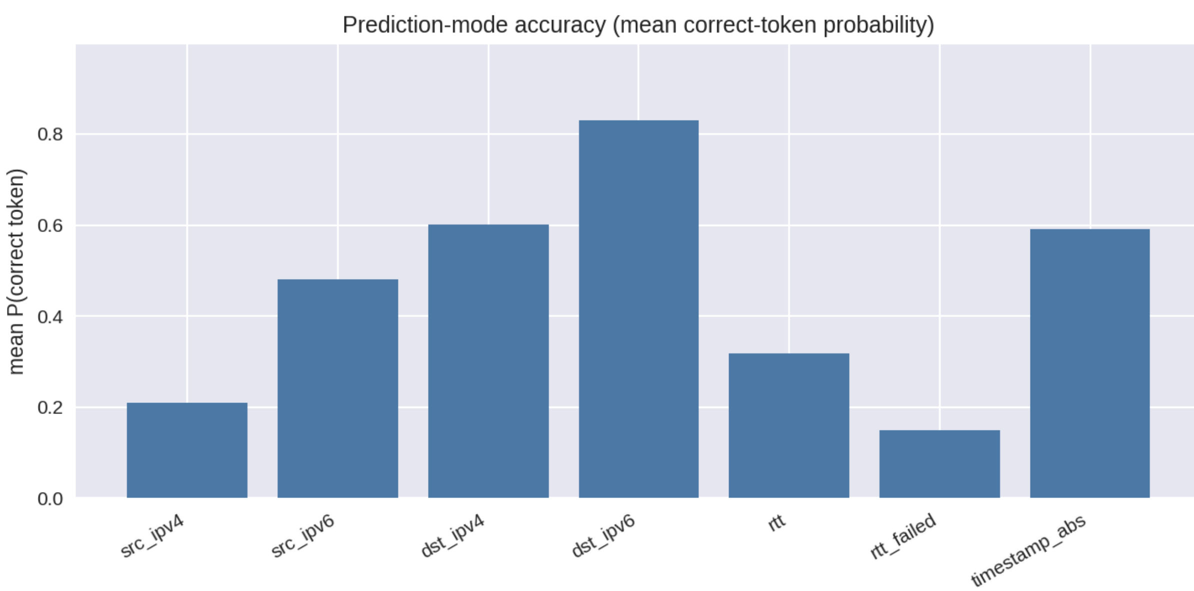

Fig. 5: Prediction-mode accuracy by token group (mean P(correct token)).

Potential additional evaluation ideas:

- Compare against simple, established methods (nearest neighbor using IP prefix, autonomous system, or geography; basic regression; and classic network coordinate systems) to show whether the transformer adds value.
- Report error size in everyday terms (average absolute error, root-mean-square error, and percentile errors), and also check whether predicted distributions are well calibrated rather than just sharp.
- Treat failed probes as rare-event prediction and report how often the model misses failures vs raises false alarms (precision, recall, and summary scores for receiver operating characteristic and precision-recall curves).
- Report results separately for IPv4 vs IPv6 and for cold-start (no recent context) vs in-context use, so it is clear where the model helps.
- Run controlled ablations that change one design choice at a time (timestamp handling, field order randomization, context length, model size, numeric precision) to see what actually matters.

A. *Implementation Appendix*

This appendix summarizes the concrete training pipeline so the reader can map the conceptual model to the actual data flow.

```
Raw measurements
  -> group by source, sort by time, cap rows (~8 MB)
  -> ArrayRecord rows + metadata
  -> K contexts per row (K = min(ceil(n/30), 16))
  -> sample windows (log-uniform) + timestamp modes (full/
partial/none)
  -> tokenization (role + byte tokens; RTT exponent/mantissa;
timestamp deltas)
  -> pad/pack to 1024-token crops
```
Listing 1: Pipeline overview (probe-centric rows and runtime tokenization).

A few implementation details worth noting:

- Measurements are always sorted by time before delta encoding, even when partial or no-timestamp modes are applied.
- Randomized field order plus timestamp masking induces many conditional prediction tasks without changing the underlying data.

- The same row yields multiple contexts per epoch, providing strong data augmentation without duplicating stored measurements.
- Source IP is used as a proxy for probe identity; probe IDs are intentionally excluded because they are dataset-specific and not part of the model inputs.

VII. Reflections on Decentralized Learning

Truly decentralized learning, where a model is updated locally and gradients are shared, often with differential privacy applied to the gradients or various other techniques to avoid leaking too much about private training data to the rest of the network, is a very active area of research. The design space is vast, with variations on how nodes connect, how updates are exchanged, how data is made private (differential privacy, multi-party computation) among many other dimensions.

Looking at existing research, no solution that I can see so far solves *all* the following requirements that would be required for this to be practical in a ungoverned peer-to-peer network. Most of them don't fully solve just the categories they are in either. The list below is illustrative rather than exhaustive.

1) Resilience to Sybil attacks
    - No clear solutions identified in this review
2) Capable of keeping information that needs to be private, or at least providing a reasonable tradeoff between privacy and performance.
    - P4: Towards private, personalized, and Peer-to-Peer learning [2]
    - Exposing the Vulnerability of Decentralized Learning to Membership Inference Attacks Through the Lens of Graph Mixing [3]
    - Privacy-Preserving Decentralized Federated Learning via Explainable Adaptive Differential Privacy [4]
    - Low-Cost Privacy-Preserving Decentralized Learning (Zip-DL) [5]
    - Robust peer-to-peer learning via secure multi-party computation [6]
3) Capable of dealing with adversarial actors who spread bad information for specific ends.
    - Backdoor Attacks in Peer-to-Peer Federated Learning [7]
    - Robust Peer-to-Peer Machine Learning Against Poisoning Attacks [8]
    - Byzantine-Robust Decentralized Federated Learning (BALANCE) [9]
    - GRANITE: a Byzantine-Resilient Dynamic Gossip Learning Framework [10]
4) Can deal with free riders / is well-incentivized.
    - No clear solutions identified in this review

5) Capable of scaling model size to node capabilities.
   - P4: Towards private, personalized, and Peer-to-Peer learning [2]
   - Privacy-Preserving Decentralized Federated Learning via Explainable Adaptive Differential Privacy [4]
   - Low-Cost Privacy-Preserving Decentralized Learning (Zip-DL) [5]
   - A Tale of Two Learning Algorithms: Multiple Stream Random Walk and Asynchronous Gossip [11]

How in such a dynamic system like the internet is a static protocol supposed to deal with all the ways adversaries can possibly disrupt or measure a network? Not only for the purposes of figuring out if your connection is sufficiently obfuscated, but even for modeling the internet itself, or incentivizing nodes to cooperate with each other, the sheer amount of complexity and things that could go wrong / be exploited seems insurmountable.

A. Thought Experiment: Privacy-Preserving Decentralized Network Design Powered by a Universal Optimizer

One idea I've been thinking about for some time is recursively improving AI models. This is probably somewhat dangerous, but if you assume you have an AI that can take a formal specification and produce something that satisfies it under resource constraints, it allows a designer to focus on the specification rather than the problem itself. In practice such an AI doesn't exist yet, but thinking about the specification often helps one understand what implementations even need to do, what the space of implementations looks like, and where an "ideal" implementation might be hiding.

For this thought experiment, we realize that there is already an existing decentralized peer-to-peer network that is self-regulating and incentivizing: human society. How do humans naturally collaborate, specialize, and build complex protocols over time? I suspect the secret is a continually growing knowledge base embedded into our culture (science) that allows us to make increasingly accurate predictions about what will happen given a certain mechanism or initial conditions for collaboration. Can this be replicated in decentralized networks?

It feels like the root of the problem here is *world simulation*. We somehow need to collect large amounts of data, and then be able to reproduce that data as accurately as possible with as small a program as we can. If the resulting program is simple and accurate enough, then it seems likely to be a program that can simulate counterfactual cases. This paper about network topology LLMs is an example of this world-modeling in action, but if we had a universal optimizer, we could make it more optimal.

The key to creating this simulator is not just to have it be small and accurate, but also to require it to have enough structure to enable swapping the internal code running on each simulated node. I imagine this would look something like a base data structure defining a way for programs associated with nodes in a graph to talk to each other, i.e. describing the baseline of a UDP/TCP graph where two programs on two different nodes can dial each other. You then use the optimizer to find a simulation program that infers this base structure (including the simple program on each node), while simultaneously being able to simulate various levels of fidelity of this base structure to both match a dataset and to be useful for search. The "multiple levels of fidelity" part is especially interesting and is related to emergence and whether (or to what degree) you can simulate the results of a program on average, and how good of an inference that gives you.

Once you have a good simulator, you can start doing RL to find policies. We do not even need to run the simulator with special algorithms in it directly; we can just use the universal optimizer to say "find me a protocol that optimizes the collective utility among many nodes running that protocol." In theory, this should re-invent simple fully trusted protocols like unencrypted communication or Paxos. Then, to make it resilient, you use the universal optimizer again to optimize a different utility function related to exploitation, denial of service, and surveillance, among other things. Then use the optimizer to do a joint mutual-improvement loop, where the protocol designer is optimized with respect to the adversary, which is optimized with respect to the protocol design.

This may work somewhat in theory, but the devil is in the details, specifically what exactly the local utility metric is defined as. There are some obvious things to optimize for: latency to connect to nodes, speed at which data can be retrieved, likelihood of correlating traffic to source/destination from various attacker vantage points, etc. Assuming you can define a *reasonable* utility metric (like an easier version of the alignment problem), the entire protocol might be able to pop out anything from cryptocurrencies for proper incentivization to onion routing to cryptographic protocols. I suppose the only way to figure out is to try. For me at least, I'm not sure how much I want to try to make the FOOM machine, but it is definitely attractive from an applications perspective to be able to skip the decades of research needed to make distributed systems practical and jump straight to the "best" solution AI can find.

VIII. Conclusion

Ping-LLM was a fun project. The hardest part was getting the data and actually getting started trying to run and iterate on an LLM. It was helpful to switch to a cloud provider like Modal as the Unity cluster is a little clunky to use. (I did rack up at least 40$ in compute costs, though.) From this pilot project I think overall the idea of using a transformer network to predict network properties seems solid. I want to run a few more evaluations. The next step is to scale up, train for longer and on more diverse data (the whole RIPE Atlas is on the table, as well as any bandwidth datasets I can get my hands on). I think it may also be possible to scale up to 1B parameters. Overall, this feels like a promising research direction and I plan to continue down this path in the future. Future research will focus on scaling this approach to larger models and better datasets, as well as actually testing out training in a distributed, adversarial, and privacy-preserving environment.

The code can be accessed at this repository: https://github.com/zontasticality/ping-llm. The ingesting code for processing RIPE Atlas data is here: https://github.com/zontasticality/ping-ingest

References

[1] RIPE Network Coordination Centre, "RIPE Atlas Daily Dumps." RIPE Network Coordination Centre, 2025.

[2] M. M. Maheri, S. Siby, A. Shamsabadi, S. Abdollahi, A. Borovykh, and H. Haddadi, "P4: Towards private, personalized, and Peer-to-Peer learning." [Online]. Available: https://arxiv.org/abs/2405.17697

[3] O. Touat, J. Brunon, Y. Belal, J. Nicolas, M. Maouche, and S. Ben Mokhtar, "Exposing the Vulnerability of Decentralized Learning to Membership Inference Attacks Through the Lens of Graph Mixing." [Online]. Available: https://arxiv.org/abs/2412.12837

[4] F. J. Piran, Z. Chen, Y. Zhang, J. Tang, and F. Imani, "Privacy-Preserving Decentralized Federated Learning via Explainable Adaptive Differential Privacy." [Online]. Available: https://arxiv.org/abs/2509.10691

[5] S. Biswas *et al.*, "Low-Cost Privacy-Preserving Decentralized Learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2025, no. 3, pp. 451–474, 2025, doi: 10.56553/popets-2025-0108.

[6] Y. Luo, W. Luo, R. Zhang, H. Zhang, and Y. Shi, "Robust peer-to-peer learning via secure multi-party computation," *Journal of Information and Intelligence*, vol. 1, pp. 341–351, 2023, doi: 10.1016/j.jiixd.2023.08.003.

[7] G. Syros, G. Yar, S. Boboila, C. Nita-Rotaru, and A. Oprea, "Backdoor Attacks in Peer-to-Peer Federated Learning." [Online]. Available: https://arxiv.org/abs/2301.09732

[8] M. Bouhaddi and K. Adi, "Robust Peer-to-Peer Machine Learning Against Poisoning Attacks," in *Proceedings of the 22nd International Conference on Security and Cryptography (SECRYPT 2025)*, 2025, pp. 539–546. doi: 10.5220/0013640600003979.

[9] M. Fang *et al.*, "Byzantine-Robust Decentralized Federated Learning," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*, Salt Lake City, UT, USA, 2024. doi: 10.1145/3658644.3670307.

[10] Y. Belal, M. Maouche, S. Ben Mokhtar, and A. Simonet-Boulogne, "GRANITE: a Byzantine-Resilient Dynamic Gossip Learning Framework." [Online]. Available: https://arxiv.org/abs/2504.17471

[11] P. Gholami and H. Seferoglu, "A Tale of Two Learning Algorithms: Multiple Stream Random Walk and Asynchronous Gossip." [Online]. Available: https://arxiv.org/abs/2504.09792