City Research Online

City St George's, University of London

Citation: Barry, I-J., Gashi, I., Salako, K., Marques, P. & Madhyastha, P. (2026). Establishing Zero-Shot LLM Performance at Network Tomography. In: UNSPECIFIED (pp. 98-106). IEEE. ISBN 979-8-3195-3244-2 doi: 10.1109/dsn-w70714.2026.00036

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/37990/

Link to published version: https://doi.org/10.1109/dsn-w70714.2026.00036

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

Establishing Zero-Shot LLM Performance at Network Tomography

1st India-Jane Barry
Department of Computer Science
City St George's, University of London
London, UK
0009-0007-9527-2293

2nd Ilir Gashi
Department of Computer Science
City St George's, University of London
London, UK
0000-0002-8017-3184

3rd Kizito Salako
Centre for Software Reliability
City St George's, University of London
London, UK
0000-0003-0394-7833

4th Pedro Marques
British Telecommunications
Ipswich, UK
0000-0002-5997-3144

5th Pranava Madhyastha
Department of Computer Science
City St George's, University of London
London, UK
0000-0002-4438-8161

***Abstract*—This paper investigates the zero-shot capabilities of LLMs to reconstruct the origin network's underlying structure from the produced NetFlow data. We conduct a comprehensive empirical analysis of thirteen open-weight models and a closed-weight, large-scale model, evaluating their ability to reconstruct topological and application-level features from limited NetFlow samples. We demonstrate that off-the-shelf LLMs can infer certain topological features from NetFlow metadata. However, we identify a clear boundary to these capabilities: while structural inference is strong, models struggle to generate plausible values for dynamic, unbounded numerical features. This study provides a critical baseline for LLM-based traffic analysis, highlighting the fragility of metadata anonymisation and the security implications of deploying LLMs in network-aware applications.**

I. Introduction

An awareness of a network's structure, i.e. the network devices, the communication channels between them, the routing protocols and firewalls, etc., enables or enhances a number of use-cases, such as synthetic data generation and network management, while also revealing potential vulnerabilities. The capacity of Large Language Models (LLM) for high-dimensional pattern recognition suggests they could potentially identify the underlying originating structures that produce network traffic, and offers a novel paradigm for gaining such awareness from previously overlooked data sources. However, this capability introduces a critical paradox: the same capabilities that enable this inference of the network's topology also presents a significant, unquantified privacy risk.

Traditional reconnaissance methods are split between 'active' (high detail, low stealth) and 'passive' (low detail, high stealth). LLM-based analysis of traditionally unprivileged information could disrupt this standard. By analysing existing data artifacts rather than probing the live network, this technique avoids generating detectable traffic and is thus invisible to conventional network monitoring. Yet, as we demonstrate in this paper, the inferred network configuration information can be surprisingly complete. A successful *inference attack* could potentially provide an adversary with a detailed map of the target network, revealing critical intelligence for reconnaissance and attack planning, effectively bypassing perimeter defences.

The scarcity of highly realistic synthetic network traffic data presents a persistent bottleneck in cybersecurity research. LLMs-based generation of traffic which faithfully reflect the implicit constraints of underlying network topology and applications could present a solution to this obstacle.

Despite the possible applications, applying LLMs to the highly structured, non-language based domain of network traffic data - in particular NetFlow data - is not a straightforward task. A primary obstacle stems from the fundamental mismatch between the training corpora for the LLMs and the nature of the task. LLMs are predominantly trained on vast datasets of natural language, whereas comma-separated NetFlow records represent a highly structured, tabular format that may prove difficult for them to parse and interpret [1].

Beyond mere semantic understanding, a more profound challenge lies in learning the implicit structural and topological constraints inherent in the data. Furthermore, an ideal model must develop a nuanced, context-aware generation strategy, distinguishing between those fields that demand memorisation and repetition of static values (e.g. a fixed router interface) and those that require the generation of novel, yet plausible, approximations (e.g. timestamps).

These challenges, and the recent evolution of LLM training corpora to include vast amounts of source code and mathematical text, motivate a structured inquiry into their latent capabilities for this task and our primary research questions:

RQ1: Are LLMs able to infer the structural and topological features of a network from limited NetFlow data?

Expectation Models will fail to infer the underlying structural and topological features of a network from NetFlow data, resulting in outputs with no statistical significance and poor distributional similarity to the ground truth

RQ2: What features in a training dataset impact a models' ability to infer network topology and application behaviour?

Expectation: Including structured data (e.g. code or maths) in a training dataset will enhance an LLM's ability to model logical and topological constraints of NetFlow data, resulting in improved performance. However, regardless of the presence of structured data, models with larger training datasets will outperform models with smaller training datasets.

RQ3: Do LLMs exhibit any distinct difference in ability when generating more or less structured data types?

Expectation: LLMs will struggle to accurately generate dynamic unbounded numerical features, exhibiting significantly worse performance than on static, rule-based features.

RQ4: What is the baseline capability of LLMs for generating semantically valid data that demonstrates a basic conceptual understanding of networking and NetFlow data?

Expectation: Models will generate semantically invalid NetFlow data, producing random or malformed outputs that do not adhere to the expected data types and formats, and will show little understanding of networking and NetFlow data.

To investigate these questions, we perform a comprehensive empirical analysis of modern LLMs and their ability to infer and replicate network topology and application behaviour from NetFlow data. We evaluate a diverse set of thirteen open weight 6-9B parameter LLMs. Our evaluation assesses how well these models can reconstruct distinct structural concepts, from device and application fingerprinting to logical network segmentation and end-to-end path construction. Our primary contributions in this paper are

- **Establishes The Potential of LLMs to Infer Network Information**. We evaluate 14 LLMs and establish their viability for this task, and a zero-shot baseline for what network information is most vulnerable to inference, against which future research can be compared.
- **A Novel Security Analysis of LLMs on Presumed-Anonymised Network Data**. A comprehensive security analysis of the risk of LLM network topology inference from limited NetFlow samples. We developed a framework for mapping NetFlow fields to exploitable topological and application-level data.

Our analysis reveals that models fine-tuned for code and logical reasoning are more adept at inferring network structure; static configurations (e.g., router interfaces) are more easily inferred, than dynamic data (e.g., traffic volumes, specific paths); and even small, incomplete data snippets can lead to accurate inference if they contain key identifiers (flows from various subnets, etc). Code and data is available in an affiliated GitHub repository, along with a in-depth technical report containing detailed information about the experimental set-up.

II. Background

A. LLMs for Network Traffic Generation

Foundation models are machine learning models that have been pre-trained [2] on a large corpus of general (often web-scraped) data [3]–[8].

Foundation models can undergo post-training into fine-tuned variants, such as task or domain-specific models. This involves further training a model- using the weights of said model as a starting point- on a smaller domain specific dataset [2].

There are multiple post-training methods, including Supervised Fine-Tuning, and Direct Preference Optimization [9].

Alternatively, domain-specific models may instead be pre-trained 'from scratch' on a domain-specific corpus, creating a model that is specialised for that domain [10]–[12].

A 'frontier model' is here defined as a production-quality, general LLM which matches or exceeds the capabilities of the current-most advanced models.

When training LLMs for tabular data generation the presence of source code in the training dataset instils a strong inductive bias towards structural consistency and symbolic reasoning, improving structured data generation performance [13] and improves performance due to its rigid syntax and hierarchical dependencies, attributes it shares with tabular formats and data serialization schemas [14].

Completing missing values (infilling) requires models to condition on both the prefix and suffix. Standard models can only predict tokens based on the prefix. One method to enable infilling, is to train models on fill-in-the-middle data [15].

The models used in this study are foundation models- and fine-tuned variants of these trained on code and mathematics data, among others- as well as 'from-scratch' domain-specific models with the same architecture, and an (at the time) 'frontier model': Gemini-2.0-Flash.

B. NetFlow Data and Network Topology

Network Flow (or NetFlow) data summarises unidirectional flows of packets that share a set of common properties: the "characteristic 5-tuple" (source & destination IPs, the source & destination port numbers, and the Protocol) among other fields. When collected by nfdump- [16]- and output in CSV format, 48 distinct fields are generated.

NetFlow acts as a lossy compression of network state, reflecting the graph structure of the network. Every flow record defines an edge in the network graph. Therefore predicting NetFlow fields is not only a data completion task; it is a topological inference task [17], [18]. A model's ability to predict NetFlow fields can serve as a proxy for the network's routing logic and physical layout.

III. Experimental Design

A. Models

To ensure a meaningful comparison, our study focuses on a set of models in the 6-9B parameter, consisting of thirteen open-weight transformer-based decoder-only models (see Table I for more details). We also include Gemini-2.0-Flash, a frontier model, as a performance benchmark [19].

The models evaluated include several prominent families. The LLaMa family consists of: LLaMa2, a foundation model; CodeLLaMa, a code-trained LLaMa2 fine-tuned variant; LLaMa3.1, another foundation model in the family,

TABLE I
A SUMMARY OF THE THIRTEEN OPEN-WEIGHT MODEL ARCHITECTURES AND THEIR ASSOCIATED TRAINING CORPORA CHARACTERISTICS.

Model	Params	Activation	Math	Code	FIM	Attn.[1]	Layers	Heads[2]	Context Length	Tokens	Training[3]	Tokenizer[4]
LLaMa2	7B	SwiGLU	N	Y($\ll$)	N	MHA	32	32/32	4K	2T	N	SP[5]/32K
CodeLLaMa	7B	SwiGLU	N	Y(85%)	Y	MHA	32	32/32	4K (16K LC)	+500B	P	LLaMa2 +4CT[6]
LLaMa3.1	8.1B	SwiGLU	Y (25%)	Y(17%)	N	GQA	32	24/8	8K	15T	N	TT[7]/128K
Tulu3	8B	SwiGLU	Y (+)	Y(+)	N	GQA	32	24/8	8K	+$\sim$1B	P	LLaMa3.1
Qwen2.5	7B	SwiGLU	Y	Y	N	GQA	28	28/4	128K	18T	N	TT/152K/22CT
Qwen2.5-Coder	7B	SwiGLU	Y (+10%)	Y(70%)	Y	GQA	28	28/4	128K	5.2T	R	Qwen2.5+7CT
Gemma	7B	GeGLU	N	Y	N	MHA	28	16/16	8K	6T	N	SP/256K
CodeGemma	7B	GeGLU	Y	Y(+80%)	Y	MHA	28	16/16	8K	+500B	P	SP/256K
DeepSeek-LLM	7B	SwiGLU	N	N	N	MHA	30	32/32	4K	2T	N	SP/100K/15CT
DeepSeek-BC	7B	SwiGLU	Y (+7%)	Y(+70%)	N	MHA	30	32/32	4K	+2T	P	SP/100K/15CT
DeepSeek-Coder	6.7B	SwiGLU	N	Y(+)	Y	MHA	30	32/32	16K	2T	R	HF[8]/32K/22CT
StarCoder2	7B	NR	Y	Y	Y	GQA	32	36/4	4K (16K LC)	3.5T	N	SC2[9]/49K/34CT
Olmo2	7B	SwiGLU	Y (0.6%)	Y	N	MHA	32	32/32	4K	3.9T	N	TT/100K

[1] Attention mechanism: MHA (Multi-Head Attention) / GQA (Grouped-Query Attention)
[2] Query/Key-Value heads used.
[3] Post-trained variant (P) / Retrained 'from scratch' on a specialised dataset (R)/ Neither (N)
[4] Tokenizer used/vocabulary size.
[5] SentencePiece tokenizer- Byte-Pair-Encoding (BPE) [20]) & unigram language model [21]
[6] Control tokens
[7] TikToken tokenizer (BPE)
[8] HuggingFace Tokenizer (BPE)
[9] Custom tokenizer (BPE & GPT-2 pre-tokenizer regex-splitter)

trained on a much larger dataset than LLaMa2 [5], [6]; and Tulu3, a code-trained LLaMa3.1 fine-tuned variant [22].

The Qwen model family, represented by: foundation model Qwen2.5, and Qwen2.5-Coder: an architecturally identical code-specific model [7], [8], [12].

The Gemma model family: Gemma, a foundation model, and CodeGemma, a variant fine-tuned on code [4], [23].

The DeepSeek family is represented by DeepSeek-LLM- a LLaMa-style foundation model- and two code-focused variants: DeepSeek-Base-Coder-v1.5, fine-tuned from DeepSeek-LLM; and DeepSeek-Coder, an architecturally identical model trained on a code-centric dataset [3], [11].

Starcoder2 is a model trained on a code-heavy, highly filtered dataset [10]. The Olmo2 7B model is pre-trained on web-data, then 'mid-trained' on a high-quality dataset, with the final model being an average of models produced from multiple mid-training runs with different data orders [24].

B. Dataset Construction

Our experiments utilise a dataset of realistic simulated NetFlow records provided by BT as CSV files, generated using Keysight Ixia's BreakingPoint traffic simulator. A wide range of use cases (video streaming, corporate traffic, etc.) are simulated. Simulated network traffic was collected and processed using nfdump [25]. NetFlow data characteristics are mirrored in our dataset: several topological fields in our dataset are highly constrained, mirroring how the majority of network traffic is funnelled through a few key interfaces.

C. Prompt Construction

Models were managed and responses generated via the Ollama framework. A prompt defined each model's role as a network expert tasked with infilling missing CSV data.

For each experimental scenario, 100 distinct inputs were given to each model. Models were all given identical system and user prompts, and identical constraints on their outputs, to ensure a fair test and comparison of like-for-like results. These are available in the associated artifact.

As shown in 1, user prompts consisted of a number of complete, consecutive NetFlow records and one partial record with a number of features removed- either preceding or following the complete records- which the model had to complete.

The length of the contextual records provided before the missing or partial record was 4 records for the majority of the experiments, however an additional set of experiments designed to measure the effect of varying context lengths varied the contextual length for the models for 100 inputs each, with the last record removed.

A JSON schema constrained the model responses. Further information about the exact data structures involved are available in the associated technical report.

D. Evaluation and Metrics

Evaluating synthesised tabular data is inherently difficult, and there is no single widely accepted evaluation metric [26]. Instead there exists a large number of methods to evaluate the statistical fidelity of synthetic data compared to the corresponding real data [27]. We employed a wide range of metrics categorised by the type of data being evaluated.

We chose to utilise both common feature-wise metrics, such as "Pearson's correlation", and individualised custom metrics.

We track the number of generated values that are both semantically valid and in range ("**Possible**"), and the precision - the number of values generated that match the ground truth ('**Correct Value**'), as well as the number of values generated which occur elsewhere in the prompt ("**Repeats**").

For IPs, we also compute the number of correct values generated for each octet in the IP Address, and the mean of these: the **Mean Partial Matches** (the mean of the ratio of correct to incorrect octets generated for each IP Address).

For numerical fields we calculate metrics such as the **median percentage difference** (the median of the set of

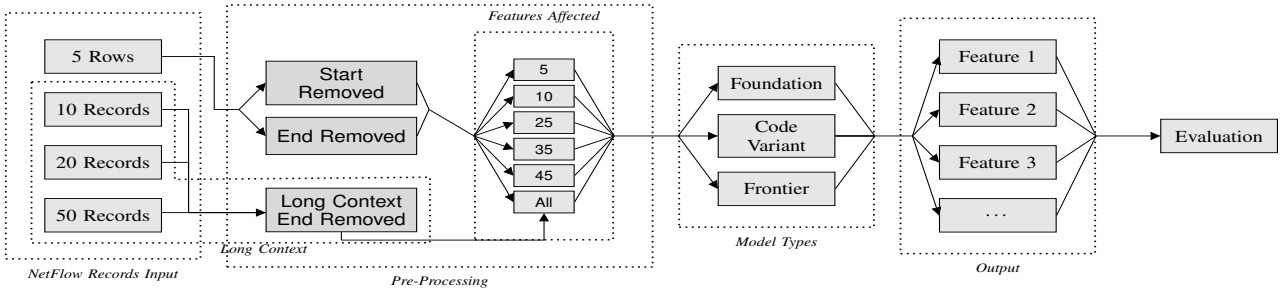

Fig. 1. A schematic representation of the experimental pipeline employed for assessing zero-shot topological inference. The workflow illustrates the stratification of input data by context window size (ranging from 5 to 50 rows of NetFlow records) and the subsequent feature ablation process, wherein specific fields (e.g., 'Start Removed', 'End Removed') are masked to simulate incomplete network telemetry.

percentage differences - the difference between the ground truth and the generated value as a percentage of their average).

For categorical fields we calculate the **total variation complement**. (The complement of the Total Variation Distance).

For 'typed' fields such as source & destination ports we also track the percentage of values that belong to the same 'group' as the ground truth- '**Correct Type**' (CT). For ports, a CT signifies that the model generated a port within the same IANA-defined range (system (0-1023), a registered (1024-49151), or dynamic port (49152–65535)) as the true port, while for TCP flags a CT indicates that the flags generated were appropriate for the protocol generated [28].

The statistical significance of our findings were evaluated against null hypotheses (H_0) where models are presumed to select values uniformly at random from the pool of previously observed values for that field (or type of field).

IV. Evaluation

This section presents an overview of LLMs' performance. Available in the associated technical report is an in-depth analysis of LLM performance at individual feature generation, as well as potential causes for variations on performance.

A. Device and Application Inference

A primary goal of network traffic analysis is to identify the specific devices on the network and the applications they are running. By strategically removing fields, we indirectly measure whether the model has inferred routing logic and topology. We evaluate models' ability to generate identifiers, service-specific port numbers, protocol information, and traffic volumes (among other features) which serve as indirect indicators of how well models are able to recognise key components and activities within the network infrastructure.

1) Device-Level Mapping: The Router Address was constant in all prompts. The challenge was to recognise this field and apply a constant rule. The retrained models, Qwen2.5-coder and DeepSeek-Base-Coder, appear better at detecting invariant patterns, such as constant Router Address values, possibly due to previous exposure from placeholders in structured data, supporting our expectation for RQ2 (see Table V).

The Input and Output Interface fields provide a more dynamic and realistic test of device-level mapping. As discussed in (III-B), these fields were highly constrained. This transforms the generation task into a context-aware classification challenge, testing a model's ability to learn which interface a flow should traverse based on its other characteristics.

This task demonstrated the variable impact of code specialisation: models like CodeLLaMa and DeepSeek-Base-Coder - either retrained from scratch or fine-tuned on code data - significantly outperformed their general-purpose base models as shown in Tables III and VI. However, this performance enhancement was not universal. In some cases, specialised models like Tulu3 and CodeGemma underperformed compared to their base models (LLaMa3.1 and Gemma, respectively).

This establishes that training on structured data like code can both enhance and impair a model's ability to infer a router's internal connectivity logic. This both supports and contradicts our expectation for RQ2, and contradicts our expectation for RQ1. A possible explanation for this contrasting behaviour is over-fitting; the base models are pre-trained on significant amounts of code, and the additional fine-tuning may have made them too specialised, hindering their ability to generalise to the novel structure of NetFlow data.

2) Application and Service Inference From Categorical Data: Protocol showed very high accuracy across many models (see Table VII) indicating they can easily distinguish flow types. When only the first 5 values were removed- a scenario in which the complex TCP flags value was present in the damaged record but the protocol value was not- several models exhibited a performance collapse, suggesting the models are able to recognise the relationship between the protocol used and the TCP flags present, contradicting our expectation for RQ4 and potentially suggesting a conceptual understanding of networking. This is contrasted by models' strong performance in a pure forecasting scenario (entire record removed), which suggests that the initial confusion may be context-specific.

Models were seemingly able to consistently learn the rule that TCP flags are only appropriate for the layer 4 protocol, even when they did not generate the exact correct flag combination (see Table VIII). Furthermore, the number of

correct flags and correctly typed flags tended to exhibit relative stability across most scenarios for the majority of models. This implies the association between protocol and flag is learned with considerable robustness by these models, or that they have an understanding of the information the data is conveying.

Performance in generating TCP Flag values was less dependent on the amount of context and more on the type of missing context (initial vs. terminal values). Retraining on code-centric datasets conferred a significant performance advantage. A key success was that the proportion of TCP flags field values generated which were valid for the generated protocol remained high and stable across scenarios, indicating that models were able to robustly learn the fundamental rule that TCP flags only apply to the TCP protocol, a key step in correctly fingerprinting application behaviour.

The source & destination ports are also strong indicators of specific applications. Models were significantly more successful at predicting destination ports, (see Table IX). This may be due to the tendency for destination ports to often correspond to well-known service ports, which are thus more predictable- again refuting our expectations for RQ4.

Furthermore, the number of ports generated of the right type was frequently significantly higher than the number of exact correct values, indicating that models often grasp the semantic category of the port, demonstrating a capacity for application-level inference even when the port number is either not perfectly recalled, or needs to be generated unseen.

The Source & Destination Type of Service fields are highly useful for fingerprinting application behaviours. This is a relatively simple task for most models, which achieve high accuracy across all scenarios, with performance remaining stable even with significant context removal, mostly due to models learning a simple, dominant pattern. All but one correctly generated value for source type of service (generated by LLaMa2 when the first 25 values were removed) was a '0'.

Model performance on the destination type of service generation task was nearly perfect, confirming their ability to master simple, rule-based data generation. The few minor errors observed demonstrate the inherent stochastic nature of generative models. While this demonstrates models can easily learn a network's default traffic profile, the lack of diverse QoS markings in the dataset prevented a more robust test of their ability to infer application types based on specific values.

3) Application and Service Inference From Numerical Data: In a finding that defines the boundaries of current zero-shot capabilities, models demonstrated a significant performance gap between static and dynamic features. Combined with section IV-A2 this supports our expectations for RQ3.

Models struggled to generate plausible dynamic numerical features, with all models exhibiting a performance collapse. For all four "flow size" fields ('ipkt', 'ibyt', 'opkt', 'obyt'), the median percentage difference was frequently -200%, or close to for most models. This seems to support our initial expectation for RQ4, that models will be unable to generate semantically correct data, in the case of numerical fields.

B. Logical Network Structure and Path Construction

1) Subnet Masks and IP Addresses: A core test of a model's understanding of logical network structure is its ability to generate valid IPs that conform to the correct subnet mask. The subnet mask defines the boundaries of the logical network segment, while the IP Address is an endpoint within that segment. Success in this area requires not just pattern matching but an application of the fundamental rules of IP Addressing.

Our evaluation shows that code-trained models are particularly adept at generating the correct subnet masks, supporting our prediction for RQ2 and contradicting our prediction for RQ1. For instance, with 25 values removed from the prompt, CodeLLaMa and DeepSeek-Base-Coder dramatically outperformed their respective general-purpose base models.

While generating the correct IP proved more challenging, the results indicate a strong structural understanding, further refuting our RQ1 expectation. Mean Partial Matches scores for IPs were consistently high, demonstrating that models often correctly generated the initial octets (placing hosts in the correct subnet) even when the full IP Address was not an exact match (Tables II & XII).

This indicates that models may grasp the "category" of the value (e.g., local subnet, external network, etc.). A similar pattern emerged for destination addresses, where models were significantly more capable of correctly generating the initial octets of the destination address (see Tables XI & XIII). For instance, in the "5 Values Removed" scenario, Gemini-2.0-Flash had a 96% match rate for the first octet but only a 4% rate for exactly correct values of the full address.

A model's success at applying the correct network rule (the mask) and subsequently generating an IP Address which reflects node positioning suggests that more advanced models might build an outline of the network's logical topology.

2) End-to-End Path Construction: A model's ability to model end-to-end network paths is most directly tested by its performance on fields such as Output Interface and Next Hop, which determine a flow's egress path. In our dataset, these tasks were simplified into pattern recognition challenges, as both fields had a constrained set of only four unique values.

There is a significant capability gap between 'Gemini-2.0-Flash' and the open-source models. The frontier model demonstrated a sophisticated ability to infer the correct path—specifically, to predict Input and Output Interfaces; the Source & Destination AS Numbers; and the Next Hop and Next Hop BGP values—when given the strong contextual anchors of source & destination IPs. It predicted the correct Output Interface with 89% accuracy when the initial record fields, including the source & destination IPs, where available. However, when the task shifted from infilling to forecasting, its accuracy dropped to 42%. A nearly identical performance drop was observed for the Next Hop field (see IV for more detail).

This stark difference between high-confidence infilling and low-confidence forecasting was unique to the frontier 'Gemini-2.0-Flash' model. The smaller open-source models were unable to capitalise on the partial context to the same degree, showing consistently low performance across all scenarios.

This suggests advanced models may possess an ability to infer specific routing logic from even small data snippets, while smaller models currently lack this capability. However, a model's performance on one path-related field was a strong predictor of its performance on similar fields (see XIV).

The parallel success across these logically linked fields provides strong evidence that top-performing models are inferring the router's internal connectivity rules, reflecting an internal model of the network's structure.

C. High-Level Inter-Net Topology

The generation of Source & Destination AS Number evaluates a model's ability to map high-level inter-net topology- also measured by the performance on the Next Hop BGP, which is intrinsically linked to the AS-level path.

The model is required to select and reproduce the correct AS Number and Next Hop BGP from a very small set based on the surrounding flow data, which realistically mirrors how a single network peers with a limited number of other networks.

There is a clear performance hierarchy: the more recent or code-specialised models like Gemini-2.0-Flash, LLaMa3.1, and DeepSeek-Base-Coder generally outperformed less recent or general-purpose models (see Tables XV and XIV). Many of the top-performing models achieved high accuracy and perfectly replicated the frequency distribution of the AS Number, demonstrating a sophisticated understanding of flow destinations and sources in the high-level inter-net topology.

D. Impact of Input Context Size on Inference Capabilities

We also investigated how performance varied when models were provided with increased input sizes (specifically 5, 10, 20, and 50 preceding records).

When input length increased from 20 rows to 50 performance often declined, suggesting that while LLMs can extract patterns from limited sequences, larger contexts may introduce noise or conflicting signals that hinder generalisation.

Models seemed able to leverage larger windows– to a point– when generating certain fields, particularly simpler ones, with model performance remaining high and stable as context increased from 5 to 10 to 20 rows. Fields dominated by a single value (e.g. source type of service) showed high stability. For fields requiring more nuanced inference - such as source port or Output Interface - performance often peaked at 10 or 20 rows before declining, often dramatically, at 50 rows.

This behaviour could be caused by models struggling to weigh the relevance of information across larger contexts, or latching on to spurious correlations present in larger sequences.

Gemini-2.0-Flash generally demonstrated greater resilience or improvement when the context window increases, behaviour contrasted with the majority of open-weight models tested. For fields such as Forwarding Status, source type of service, or Destination Mask Gemini-2.0-flash maintained its high performance while most models performance declined with inputs over 20 rows. This suggests Gemini-2.0-flash is able to handle larger amounts of zero-shot information with good performance.

Neither general purpose nor code-specialised models were able to leverage increased amounts of context for improved performance; though some models had higher baseline accuracy they still followed the same trend of diminishing returns or degradation with more extensive context.

Models with larger context training experience considerably less performance decay as the context size increases, though all models- even those with large context training- exhibited a degradation in performance at 50 rows. One possibility is that this occurs as a result of the majority of the model training inputs being shorter than the (approximately 12K tokens) length of the 50-row-long inputs.

V. Conclusion

Our evaluation reveals significant zero-shot capability of LLMs to generate NetFlow data.

Refuting RQ1, top-performing models consistently correctly generated constrained categorical features , providing strong evidence that models can infer the underlying rules of computer networks in general, and some evidence of the ability to infer the specifics of the origin network's.

Our expectation for RQ2 is largely supported: models trained on code-centric datasets demonstrated improved performance, particularly when generating features associated with logical and topological constraints. However training data mixture seemed to have a greater effect than dataset size.

Our expectation for RQ3 was strongly supported, with a critical performance gap between inferring static, rule-based configurations and dynamic traffic behaviour.

Contrary to our expectation for RQ4, models were usually able to generate semantically valid data across most fields.

The security implications of our findings are significant. The uncovered capabilities of LLMs constitute a tangible information leakage threat. An adversary could leverage these models to conduct detailed network reconnaissance with far greater efficiency than traditional methods.

Limitations include: i) the study was scoped to zero-shot models and thus did not evaluate the potential for fine-tuning models on NetFlow data; ii) the experiments were conducted on a dataset generated by a traffic simulator. While every effort was made to produce realistic traffic, the dataset may not capture the full complexity and noise of a production network; iii) minimal pre-processing of the input data was done before prompting the models, and thus the performance of the models on less noisy data is unknown.

Avenues for future research include: i) exploring the impact of domain-specific fine-tuning, and de-noised data to determine the upper bounds of model performance on this task; ii) exploring the hypothesis that code-specialised models can underperform due to over-fitting; iii) development of data anonymisation and obfuscation techniques resistant to LLM-based inference.

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2019.

[3] X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu, *et al.*, "Deepseek llm: Scaling open-source language models with longtermism," *arXiv preprint arXiv:2401.02954*, 2024.

[4] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, *et al.*, "Gemma: Open models based on gemini research and technology," *arXiv preprint arXiv:2403.08295*, 2024.

[5] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[6] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, and Others, "Llama: Open and efficient foundation language models," *ArXiv Preprint ArXiv:2302.13971*, 2023.

[7] J. Bai and Others, "Qwen technical report," *ArXiv Preprint ArXiv:2309.16609*, 2023.

[8] A. Yang and Others, "Qwen2.5 technical report," *ArXiv Preprint ArXiv:2412.15115*, 2024.

[9] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," 2024.

[10] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, *et al.*, "Starcoder 2 and the stack v2: The next generation," *arXiv preprint arXiv:2402.19173*, 2024.

[11] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li, *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[12] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, and Others, "Qwen2.5-coder technical report," *ArXiv Preprint ArXiv:2409.12186*, 2024.

[13] J. Petty, S. van Steenkiste, and T. Linzen, "How does code pretraining affect language model task performance?," 2025.

[14] M. Chudoba, T. Wiktorski, and S. Alyaev, "Revisiting structured representation use in llm," *Norsk IKT-konferanse for forskning og utdanning*, vol. 37, 11 2025.

[15] M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen, "Efficient training of language models to fill in the middle," 2022.

[16] P. Haag, "nfdump: Netflow processing tools," 2024. BSD 2-Clause License.

[17] J. A. Cordero and O. Bonaventure, "Understanding the topological properties of internet traffic: A view from the edge," in *2014 IFIP Networking Conference*, pp. 1–9, 2014.

[18] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *2012 IEEE Symposium On Security And Privacy*, pp. 332–346, 2012.

[19] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[20] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings Of The 54th Annual Meeting Of The Association For Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, aug 2016.

[21] T. Kudo, "Subword regularization: Improving neural network translation models with multiple subword candidates," 2018.

[22] N. Lambert, J. Morrison, V. Pyatkin, S. Huang, H. Ivison, F. Brahman, L. Miranda, A. Liu, N. Dziri, S. Lyu, Y. Gu, S. Malik, V. Graf, J. Hwang, J. Yang, R. Bras, O. Tafjord, C. Wilhelm, L. Soldaini, N. Smith, Y. Wang, P. Dasigi, and H. Hajishirzi, "Tulu 3: Pushing frontiers in open language model post-training," 2025.

[23] C. T. et al, "Codegemma: Open code models based on gemma," 2024.

[24] OLMo, T., P. Walsh, L. Soldaini, D. Groeneveld, K. Lo, S. Arora, A. Bhagia, Y. Gu, S. Huang, M. Jordan, N. Lambert, D. Schwenk, O. Tafjord, T. Anderson, D. Atkinson, F. Brahman, C. Clark, P. Dasigi, N. Dziri, M. Guerquin, H. Ivison, P. Koh, J. Liu, S. Malik, W. Merrill, L. Miranda, J. Morrison, T. Murray, C. Nam, V. Pyatkin, A. Rangapur, M. Schmitz, S. Skjonsberg, D. Wadden, C. Wilhelm, M. Wilson, L. Zettlemoyer, A. Farhadi, N. Smith, and H. Hajishirzi, "2 OLMo 2 Furious," 2025.

[25] Keysight, "Breaking Point Applications and Security Testing — keysight." https://www.keysight.com/gb/en/assets/3120-1270/data-sheets/BreakingPoint-Applications-and-Security-Testing.pdf, 10 2025.

[26] Y. Du and N. Li, "Systematic assessment of tabular data synthesis," 2025.

[27] M. C. Stoian, E. Giunchiglia, and T. Lukasiewicz, "A survey on tabular data generation: Utility, alignment, fidelity, privacy, and beyond," 2025.

[28] Internet Assigned Numbers Authority (IANA), "Internet Assigned Numbers Authority (IANA) Service Name and Transport Protocol Port Number Registry." https://www.iana.org/assignments/service-ports/service-ports.xhtml, 2025.

APPENDIX

Results

Significance is denoted by * for a p-value ≤ 0.05 and by ⁑ for a p-value ≤ 0.001. The ***significance threshold***—the minimum number of values matching the ground truth a model needed to generate (out of 100) to achieve a p-value ≤ 0.05—was calculated using the Poisson Binomial Distribution.
]

TABLE II
SOURCE ADDRESS (SA) GENERATION: EMP[1] VS. MPM[2]

Model	25 Values Removed		45 Values Removed	
	EMP[1]	MPM[2]	EMP[1]	MPM[2]
LLaMa2	0.030	0.150	0.030	0.228
CodeLLaMa	0.060	0.605	0.060	0.487
LLaMa3.1	0.080	0.480	0.060	0.400
Tulu3	0.100	0.65	0.060	0.492
Qwen2.5	0.000	0.245	0.030	0.355
Qwen2.5-Coder	0.070	0.552	0.040	0.302
Gemma	0.000	0.115	0.100	0.665
CodeGemma	0.030	0.302	0.100	0.647
DeepSeek-LLM	0.040	0.340	0.070	0.353
DeepSeek-Coder	0.010	0.260	0.030	0.065
DeepSeek-Base-Coder	0.070	0.332	0.040	0.245
StarCoder2	0.020	0.138	0.000	0.088
Olmo2	0.000	0.030	0.010	0.035
Gemini-2.0-Flash	*0.020*	*0.502*	*0.060*	*0.657*

Compares EMP[1] and MPM[2] scores across models to evaluate subnet placement in generated Source IP addresses.
[1] Exact Match Proportion
[2] Mean Partial Match

TABLE III
INPUT INTERFACE ('IN') RESULTS (25 VALUES REMOVED)

Model	Total Variation Complement	Correct Values[1]
LLaMa2	0.03	1
CodeLlama	0.43	22
LLaMa3.1	0.60	30
Tulu3	0.68	38
Qwen2.5	0.00	0
Qwen2.5-Coder	0.38	20
Gemma	0.18	7
CodeGemma	0.1	6
DeepSeek-LLM	0.13	8
DeepSeek-Coder	0.23	14
DeepSeek-Base-Coder	0.44	25
Gemini-2.0-Flash	*0.74*	*36*

Compares 'in' mapping scores across models to evaluate device level mapping.
[1] Significance threshold 56

TABLE IV
PATH CONSTRUCTION ('OUT' AND 'NH') WITH AND WITHOUT SOURCE AND DESTINATION ADDRESSES FOR 'GEMINI-2.0-FLASH'

	CV[1]('out')	CV ('nh')
With Source & Destination Address [1]	89	78
With Source & Destination Address [2]	89	81
Without Source & Destination Address [3]	42	43

Compares success of fields associated with path construction with & without endpoint addresses to evaluate understanding of network links.
Values removed
[1] Last 33 [2] Last 43 [3] All (last 48)

TABLE V
ROUTER ADDRESS RESULTS (ENTIRE RECORD REMOVED)

Model	Addresses Generated Correctly[1]
Qwen2.5	16
Qwen2.5-Coder	33
DeepSeek-LLM	0
DeepSeek-Coder	17

Compares code-trained & base models ability to reproduce constant values
[1] Significance threshold of 35

TABLE VI
OUTPUT INTERFACE ('OUT') RESULTS (LAST 33 VALUES REMOVED)

Model	TVC[2]	Correct Values[1]
LLaMa2	0.07	4
CodeLLaMa	0.23	13
LLaMa3.1	0.60	19
Tulu3	0.68	4
Qwen2.5	0.0	33
Qwen2.5-Coder	0.38	3
Gemma	0.18	38
CodeGemma	0.10	10
DeepSeek-LLM	0.094	3
DeepSeek-Coder	0.23	11
DeepSeek-Base-Coder	0.55	25

Compares 'out' mapping scores across models to evaluate device level mapping.

[1] Significance threshold 66 [2] Total Variation Complement

TABLE VII
COUNT OF PROTOCOL ('PR') VALUES GENERATED CORRECTLY)

Model	5[1]	15[1]	25[1]	All[2]
	Values Removed			
Lower Performers (w/ Recovery)				
CodeLLaMa	1	72	76	80
Qwen2.5-Coder	20	53	70	83
High Performers				
LLaMa3.1	76	69	65	82
Tulu3	70	15	81	79
Gemini-2.0-Flash	*86*	*83*	*65*	*84*

Compares 'high' and 'low' performing models scores at Protocol prediction to evaluate application & service level mapping
[1] Significance threshold of 87 [2] Significance threshold of 85

TABLE VIII
TCP FLAGS ('FLG') RESULTS: CORRECT FLAGS (CV) & CORRECT TYPE OF FLAG (CT)

Model	15		25		35	
	Values Removed					
	CV[1]	CT[2]	CV[1]	CT[2]	CV[1]	CT[2]
LLaMa2	0	0	0	1	0	1
CodeLLaMa	26	64	26	73	32	72
LLaMa3.1	21	48	21	51	22	56
Tulu3	9	21	24	54	13	33
Qwen2.5	4	16	0	0	0	0
Qwen2.5-Coder	24	52	23	60	23	64
Gemma	17	38	7	13	30	73
CodeGemma	1	7	2	5	1	2
DeepSeek-LLM	0	0	0	0	0	0
DeepSeek-Coder	1	2	1	1	1	1
DeepSeek-Base-Coder	8	12	8	12	8	11
StarCoder2	3	5	3	6	2	4
Olmo2	0	1	0	0	0	0
Gemini-2.0-Flash	*40*	*83*	*35*	*65*	*39*	*83*

Compares model's scores at TCP Flag prediction to evaluate application & service level mapping
[1] Significance threshold of 44 [2] Significance threshold of 87

TABLE IX
SOURCE (SP) & DESTINATION (DP) PORTS: CV (CORRECT PORTS) & CT (PORTS OF CORRECT CATEGORY) GENERATED (5 VALUES REMOVED)

Model	CV		CT	
	SP[1]	DP[2]	SP[3]	DP[4]
LLaMa2	1	2	1	11
CodeLLaMa	8	22*	26	40
LLaMa3.1	10	21*	46*	42
Tulu3	4	29*	39	46*
Qwen2.5	6	10	29	28
Qwen2.5-Coder	10	21*	33	47*
Gemma	7	30*	19	50*
CodeGemma	9	16	26	37
DeepSeek-LLM	2	2	13	22
DeepSeek-Coder	9	16	33	33
DeepSeek-Base-Coder	4	10	14	25
StarCoder2	1	2	19	29
Olmo2	0	0	18	25
Gemini-2.0-Flash	*11*	*24**	*51**	*49**

Compares source & destination port scores across models to demonstrate enhanced behaviour at predicting destination port
Significance thresholds
[1] 12 [2] 46 [3] 18 [4] 46

TABLE X
VALUES GENERATED REPEATED FROM INPUT ("REPEATS") & PERMISSIBLE VALUES GENERATED ("POSSIBLE")

Model	Field	Repeats			Possible		
		25	45	48	25	45	48
		Values Removed			Values Removed		
LLaMa2	ts[1]	23	37	6	28	59	40
	sp[2]	30	33	11	32	60	22
	nh[3]	0	1	1	0	1	1
	in[4]	6	4	6	86	96	91
CodeLLaMa	ts	96	100	63	99	100	100
	sp	88	95	70	89	100	100
	nh	28	10	11	28	10	11
	in	42	40	35	100	100	100
Qwen2.5-Coder	ts	85	78	84	100	100	100
	sp	73	48	95	75	70	100
	nh	47	31	33	48	32	34
	in	38	45	13	100	100	100
Gemini-2.0-Flash	ts	100	98	95	100	100	100
	sp	73	98	90	74	99	100
	nh	100	100	100	100	100	100
	in	75	97	99	100	100	100

Compares repeated and permissible values across representative selection of models & fields to assess if the degree of novelty aligns with expectations.
[1] Time Stamp Start: Very high degree of novelty expected
[2] Source Port: High degree of novelty expected
[3] Next Hop Address: Mid-Low degree of novelty expected
[4] Input Interface: Low degree of novelty expected

TABLE XI
DESTINATION ADDRESS (DA) GENERATION: EM[1] VS. MPM[2]

Model	5 Values Removed		25 Values Removed	
	EMP	MPM	EMP	MPM
LLaMa2	0.01	0.035	0.050	0.200
CodeLLaMa	0.050	0.335	0.100	0.622
LLaMa3.1	0.070	0.540	0.040	0.482
Tulu3	0.100	0.585	0.090	0.647
Qwen2.5	0.000	0.140	0.000	0.245
Qwen2.5-Coder	0.070	0.590	0.110	0.578
Gemma	0.010	0.177	0.010	0.120
CodeGemma	0.010	0.215	0.020	0.300
DeepSeek-LLM	0.030	0.169	0.030	0.353
DeepSeek-Coder	0.020	0.497	0.010	0.280
DeepSeek-Base-Coder	0.030	0.285	0.080	0.375
StarCoder2	0.000	0.088	0.010	0.135
Olmo2	0.000	0.035	0.000	0.045
Gemini-2.0-Flash	*0.040*	*0.625*	*0.070*	*0.485*

Compares EMP[1] and MPM[2] scores across models to evaluate subnet placement in generated Destination IP addresses.
[1] Exact Match Proportion [2] Mean Partial Match

TABLE XII
SOURCE ADDRESS (SA): ENTIRE ADDRESS GENERATED CORRECTLY ('ALL') & INDIVIDUAL OCTETS GENERATED (O1–O4) CORRECTLY

Model	25 Values Removed					45 Values Removed				
	O1[1]	O2[2]	O3[3]	O4[4]	All[5]	O1[6]	O2[7]	O3[8]	O4[9]	All[10]
LLaMa2	21	21	11	3	3	33	33	19	3	3
CodeLLaMa	86‡	86‡	49‡	6	6	72‡	72*	44‡	6	6
LLaMa3.1	67‡	67	36*	8	8*	60‡	60	29	6	6
Tulu3	97‡	97‡	48‡	10	10‡	74‡	74*	35	6	6
Qwen2.5	0	0	0	0	0	29	29	16	3	3
Qwen2.5-Coder	74‡	74*	40*	7	7*	39	39	17	4	4
Gemma	19	19	6	0	0	98‡	98‡	49‡	10	10‡
CodeGemma	13	13	8	3	3	95‡	95‡	48‡	10	10‡
DeepSeek-LLM	48*	48	25	4	4	51*	51	24	7	7*
DeepSeek-Coder	12	12	9	1	1	7	7	4	3	3
DeepSeek-BC[1]	47*	47	29	7	7*	33	33	21	4	4
StarCoder2	21	18	10	2	2	14	13	8	0	0
Olmo2	4	4	2	0	0	5	5	3	1	1
Gemini-2.0-Flash	71‡	71*	41‡	*2*	*2*	97‡	97‡	54‡	*6*	*6*

Compares the frequency of correctly generated octets and entire addresses across models to evaluate subnet placement.
Significance thresholds:
[1] 46 [2] 70 [3] 34 [4] 11 [5] 7 [6] 48 [7] 68 [8] 36 [9] 11 [10] 7

TABLE XIII
DESTINATION ADDRESS (DA): FULL ADDRESS (ALL) & INDIVIDUAL OCTETS GENERATED (O1–O4) CORRECTLY

Model	25 Values Removed					45 Values Removed				
	O1[1]	O2[2]	O3[3]	O4[4]	All[5]	O1[6]	O2[7]	O3[8]	O4[9]	All[10]
LLaMa2	29	29	15	5	5	44	44	25	5	5
CodeLLaMa	87‡	86‡	48‡	10	10*	98‡	97‡	55‡	11	11‡
LLaMa3.1	70‡	69	39	4	4	82‡	82‡	35	5	5
Tulu3	97‡	96‡	47*	9	9*	85‡	85‡	40	8	8*
Qwen2.5	0	0	0	0	0	32	32	16	3	3
Qwen2.5-Coder	74‡	74*	40	11	11‡	40	39	17	4	4
Gemma	19	19	6	1	1	98‡	97‡	48*	11	11‡
CodeGemma	13	13	8	2	2	96‡	96‡	49*	11	11‡
DeepSeek-LLM	51*	51	26	3	3	55‡	55	24	6	6
DeepSeek-Coder	16	16	10	1	1	18	18	8	3	3
DeepSeek-Base-Coder	54‡	53	31	8	8*	49*	49	29	6	6
StarCoder2	22	19	8	1	1	17	16	7	1	1
Olmo2	6	6	5	0	0	84‡	84‡	40	7	7
Gemini-2.0-Flash	66‡	*66*	*39*	*7*	*7*	98‡	97‡	54‡	*8*	8*

Compares the frequency of correctly generated octets and entire addresses across models to evaluate subnet placement.
Significance thresholds:
[1] 46 [2] 70 [3] 41 [4] 12 [5] 8 [6] 48 [7] 68 [8] 43 [9] 13 [10] 8

TABLE XIV
COUNT OF CORRECT RESPONSE ACROSS PATH CONSTRUCTION FIELDS

Model	in	out	nh	sas	das
First 25 Values Removed					
LLaMa2	1	3	0	1	2
CodeLLaMa	22	23	12	17	17
LLaMa3.1	30	33	35	27	30
Tulu3	38	38	37	36	37
Qwen2.5	0	0	2	0	0
Qwen2.5-Coder	20	21	21	16	17
Gemma	7	32	5	3	12
CodeGemma	6	6	7	6	6
DeepSeek-LLM	8	6	6	7	8
DeepSeek-Coder	14	14	13	16	17
DeepSeek-Base-Coder	25	25	38	26	26
StarCoder2	0	0	1	1	1
Olmo2	1	1	0	1	0
Gemini-2.0-Flash	*36*	*35*	*32*	*35*	*33*

Compares success of fields associated with path construction across models to establish relationship

TABLE XV
NEXT HOP BGP ADDRESS: ENTIRE ADDRESS GENERATED CORRECTLY ('ALL') & INDIVIDUAL OCTETS GENERATED (O1–O4) CORRECTLY

Model	O1[1]	O2[2]	O3[3]	O4[4]	ALL[5]
LLaMa2	0	0	0	0	0
CodeLLaMa	18	18	10	10	10
LLaMa3.1	77**	77**	37	36**	36**
Tulu3	77**	77**	40	38**	38**
Qwen2.5	0	0	0	0	0
Qwen2.5-Coder	26	26	13	13	13
Gemma	16	16	5	5	5
CodeGemma	14	14	7	7	7
DeepSeek-LLM	1	1	0	0	0
DeepSeek-Coder	6	6	3	3	3
DeepSeek-Base-Coder	62**	62**	39	39**	39**
StarCoder2	7	7	5	5	5
Olmo2	0	0	0	0	0
Gemini-2.0-Flash	68**	68**	*31*	31*	31*

Compares the frequency of correctly generated octets and entire addresses across models to evaluate subnet placement.
Significance thresholds:
[1] 47 [2] 47 [3] 41 [4] 28 [5] 28