Agentic AI CPE: Traffic Steering via Small Language Models and Zero-Shot Forecasting

Giacomo Sguotti, Sebastian Troia, and Guido Maier
Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milan, Italy
Email: {giacomo.sguotti, sebastian.troia, guido.maier} @ polimi.it

***Abstract*—Software-Defined Wide Area Networks (SD-WANs) have become a key enabler of multi-homed enterprise connectivity, allowing traffic to be dynamically steered across heterogeneous access networks. However, current traffic steering mechanisms remain largely reactive and policy-driven, limiting their ability to anticipate performance degradation and to reason effectively over complex, multi-dimensional Quality of Service (QoS) objectives. This paper introduces a novel agentic AI approach for SD-WAN Customer Premises Equipment (CPE) that elevates traffic steering from reactive control to proactive, context-aware decision-making. By combining foundation-model-based time-series prediction with lightweight semantic reasoning at the network edge, the proposed approach enables predictive traffic steering that operates out of the box in new environments and adapts to application-specific requirements. Unlike prior machine learning-based solutions, our design avoids site-specific training and remains compatible with the compute and latency constraints of commodity CPE hardware. Through an experimental evaluation using a controlled testbed and real-world network traces, we demonstrate the feasibility of deploying predictive and explainable AI-driven traffic steering directly on edge devices. Our results suggest that agentic AI architectures can play a key role in advancing SD-WANs toward truly self-driving network operation.**

***Index Terms*—SD-WAN (Software-Defined Wide Area Network); Agentic AI; Traffic Steering; Small Language Models (SLMs); Zero-Shot Forecasting.**

I. Introduction

Modern enterprise Wide Area Network (WAN) connectivity has shifted from single-homed access architectures toward multi-homed deployments built on diverse underlay links. This evolution has been largely driven by the adoption of Software-Defined WAN (SD-WAN) solutions, which decouple control and forwarding to enable centralized policy enforcement and dynamic traffic management across heterogeneous access technologies. In this context, today's Customer Premises Equipment (CPE) functions as an intelligent gateway, orchestrating multiple access technologies (i.e., the physical and logical connectivity services provided by Internet Service Providers) spanning from stable wired connections to more variable mobile and satellite networks [1].

While SD-WAN provides the mechanisms for abstraction and control, it also introduces a critical traffic-steering challenge: dynamically assigning application flows (e.g., VoIP, video, bulk data) to the optimal access networks by selecting among overlay tunnels built on top of available underlay links based on real-time Quality of Service (QoS) requirements.

However, the proliferation of latency-sensitive applications at the network edge, such as cloud gaming, AR/VR, and real-time VoIP, has placed unprecedented demands on this traffic steering logic [2]. Traditional network management relies heavily on static policies or simple threshold-based automation (e.g., "switch tunnel if latency exceeds 100 ms"). While robust and lightweight, such reactive approaches suffer from two fundamental limitations. First, they are prone to flapping, where frequent oscillations between tunnels are triggered by short-lived performance fluctuations or measurement noise, leading to control-plane instability and degraded application performance. While introducing hysteresis (e.g., using distinct thresholds for triggering and reverting a switch) can mitigate this instability, such workarounds rely on static, context-unaware parameters that cannot dynamically distinguish between harmless transient noise and meaningful service degradation. Second, they lack predictive capability, reacting only after network conditions have already deteriorated, and therefore failing to anticipate or mitigate congestion before it visibly impacts Quality of Experience (QoE).

Recent advancements in Artificial Intelligence (AI) have promised a shift toward Self-Driving Networks, a vision in which networks autonomously monitor, adapt, and optimize their behavior with minimal human intervention. Unlike traditional automation, which reacts to predefined events, self-driving networks leverage AI and machine learning (ML) to continuously interpret telemetry, predict performance issues, and make real-time decisions about configuration and traffic management—ultimately enabling self-configuration, self-optimization, and self-healing to meet service goals proactively [3].

Deep Reinforcement Learning (DRL) has been explored for traffic steering in this context [4], [5], yet such DRL solutions often remain tied to a specific access network and require extensive on-site training to converge on environment-specific distributions.

In parallel, a new paradigm has emerged in time-series analysis: Time Series Foundation Models (TSFMs). Inspired by Large Language Models (LLMs), architectures such as Amazon Chronos [6] and Google TimesFM [7] model time series as token sequences and are pre-trained on large, heterogeneous datasets. A key capability of TSFMs is zero-shot forecasting, i.e., the ability to generate accurate predictions on previously unseen time series without local training or fine-tuning. For networking applications, this enables the deploy-

ment of predictive agents that operate out of the box in new environments, eliminating the lengthy training phases required by learning-based traffic steering approaches.

However, prediction alone is insufficient for effective traffic steering. QoS objectives are inherently multi-dimensional, involving trade-offs among delay, jitter, packet loss, and bandwidth. Such trade-offs often lead to ambiguous operating points where deterministic policies fail to capture application-specific priorities.

To resolve these ambiguities, semantic reasoning capabilities are required. While LLMs offer strong reasoning capabilities, their computational footprint and inference latency make them unsuitable for real-time control on resource-constrained CPE devices. Recent advances in Small Language Models (SLMs) address this gap by delivering compact yet effective reasoning through techniques such as distillation and quantization. These models can operate within the compute and memory constraints of edge platforms [8]–[10], enabling low-latency, privacy-preserving decision-making directly on the CPE without reliance on external cloud services.

In this paper, we propose an "agentic AI architecture" for SD-WAN CPEs that enables proactive and context-aware traffic steering by combining zero-shot prediction with lightweight semantic reasoning. The proposed pipeline is composed of two complementary modules:

- **Zero-Shot Forecaster:** We utilize a pre-trained TSFM to perform Zero-Shot Forecasting of network latency. This allows the agent to anticipate congestion spikes proactively without requiring site-specific training data.
- **SLM Reasoner:** We deploy a quantized SLM to act as a semantic reasoner to handle non-trivial decisions. By abstracting raw numerical forecasts into semantic "tokens" according to the application SLA (e.g., *Current RTT: VIOLATED, RTT Prediction: POOR*), the SLM can make context-aware steering decisions based on network conditions and application name/category, while also providing an explanation of its choice.

We implement the complete pipeline on a standard Linux-based edge gateway. We evaluate each module independently and finally we perform a comparison with a traditional threshold-based method using a controlled Lab-in-a-Box testbed followed by a computational feasibility analysis. All experiments are conducted on commodity consumer-grade hardware without hardware acceleration (e.g., GPUs), demonstrating that the proposed approach is practical for real-world CPE deployments.

Our specific contributions are as follows:

1) **Innovative Pipeline Design:** We present a "Token-Based" Hybrid AI architecture that decouples numerical forecasting from semantic decision-making, improving the reasoning of lightweight models on the edge.
2) **Validation of Zero-Shot Forecasting:** We provide an empirical validation of Amazon Chronos Bolt for network latency prediction, quantifying error metrics against a self-collected dataset from a live LTE network characterized by high jitter and stochastic volatility.
3) **Validation of SLM decisions:** Through a custom-built dataset of steering scenarios, we evaluate the accuracy and consistency of the decisions produced by the SLM.
4) **Baseline Comparison:** We benchmark our agent against a traditional threshold-based policy.
5) **Feasibility Analysis:** We measure the computational cost (CPU, memory) and control loop latency of the full pipeline, proving its viability on commodity hardware.

The remainder of this paper is organized as follows. Section II reviews the state of the art in applying Large Language Models and Foundation Models to network management. Section III details the architecture of the proposed pipeline, defining the interaction between the forecasting module and the reasoning engine. Section IV-A outlines the hardware and software environments used for our evaluation. Section IV-B validates the zero-shot capabilities of Amazon Chronos Bolt against a real-world LTE latency dataset. Section IV-C measures semantic capabilities of the SLM in isolation. Section IV-D describes the "Lab-in-a-Box" experimental testbed. Section IV-E compares our approach with a threshold-based traffic steering method. Section IV-F presents a benchmark of the system's computational feasibility. Finally, Section V concludes the paper and outlines directions for future research.

II. Related Work

Traditional traffic steering techniques in SD-WAN rely on fixed QoS thresholds, such as end-to-end delay, packet loss, and jitter [1]. Despite their simplicity, these mechanisms are effective in enabling fast reactions to performance degradation by switching traffic among overlay tunnels, thereby ensuring compliance with Service Level Agreements (SLAs) [11].

Dynamic rerouting mechanisms further extend this approach. Established frameworks such as Fast ReRoute (FRR) enable rapid path switching during link failures, while Performance Routing (PfR) allows CPEs to actively redirect traffic to backup paths based on observed service-level parameters [12], [13]. This capability is particularly important in large-scale, geographically distributed SD-WAN deployments, where link reliability can vary significantly. By rerouting traffic away from degraded paths, SD-WANs can preserve service quality for latency-sensitive applications such as VoIP and video conferencing.

However, these techniques suffer from several well-known limitations, notably flapping and deterministic decision-making. In the former case, when QoS metrics (e.g., delay) fluctuate rapidly around a predefined threshold, frequent path switching may occur, leading to service instability [14]. While operators commonly deploy hysteresis mechanisms (e.g., distinct thresholds for triggering and reverting a switch) to dampen these oscillations, these static guardrails lack the contextual awareness to differentiate between stochastic noise and actual degradation. Moreover, the deterministic nature of threshold-based policies prevents the anticipation of imminent SLA violations, as decisions are triggered only after a

threshold is exceeded, potentially resulting in avoidable service degradation.

To overcome these limitations, ML-based techniques have been introduced to enhance SD-WAN traffic steering. By leveraging historical and real-time telemetry, ML models can infer complex traffic patterns and anticipate future network conditions, enabling more proactive and adaptive routing decisions [4], [15]–[17]. While promising, these approaches introduce non-trivial challenges. First, frequent retraining is required to cope with rapidly changing network dynamics, demanding data collection and periodic model updates. Second, the computational cost of training can be prohibitive: approaches based on Deep Reinforcement Learning or Deep Neural Networks require substantial processing power and memory, limiting their feasibility on edge CPE devices.

In this context, recent research has shifted toward models that can generalize across scenarios without per-deployment training, while maintaining low inference latency suitable for closed-loop control at the network edge.

Time Series Foundation Models (TSFMs) enable zero-shot forecasting across domains [6]. Liu et al. [18] validate this capability in networking through an extensive evaluation of TSFMs for ISP traffic-volume prediction, demonstrating strong generalization on real-world telemetry data. Furthermore, the AWS blog on Chronos Bolt [19] reports high forecasting accuracy with inference latencies below 100 ms, making TSFMs compatible with real-time edge control loops.

While TSFMs address the numerical prediction challenge, interpreting these predictions requires semantic reasoning. Large Language Models (LLMs) have been explored for network automation tasks, including configuration generation, troubleshooting (e.g., NetLLM [20]), and SDN flow-rule synthesis at the control plane (e.g., [21]). While effective for intent interpretation and policy reasoning, these approaches typically rely on cloud-scale inference and centralized deployment, rendering them unsuitable for latency-critical traffic steering directly on CPE devices.

To address edge constraints, recent work has focused on SLMs and model compression techniques. Lu et al. [22] benchmark over 60 SLMs and show that 4-bit quantized models can achieve sub-second inference latency on commodity hardware. Additional studies confirm that SLMs significantly reduce energy consumption and memory footprint while preserving reasoning capabilities [23].

Positioning of this paper: Motivated by the complementary strengths of TSFMs (zero-shot prediction) and SLMs (compact reasoning), we integrate both components into a composite CPE agent that operates entirely on edge dataplane hardware. Our approach moves beyond reactive threshold-based mechanisms and avoids the training overhead associated with Deep Reinforcement Learning [4] by exploiting the zero-shot generalization capabilities of foundation models.

III. System Architecture

We propose a modular AI-based architecture designed to operate autonomously on resource-constrained CPEs. The system, referred to as the *CPE Agent*, functions as a closed-loop controller that continuously feeds high-frequency network telemetry to context-aware traffic steering actions.

To reconcile the computational demands of Generative AI with the latency requirements of real-time networking, our design strictly separates numerical evaluation from semantic reasoning. As illustrated in Figure 1, the pipeline is composed of seven distinct modules: (A) Monitoring, (B) Forecaster, (C) Policy Enforcer, (D) Tokenizer, (E) Hierarchical Decision Gate, (F) SLM Reasoner, and (G) the Dataplane. With the exception of the Monitoring module, which runs continuously in the background to collect high-frequency telemetry, the remaining modules execute sequentially within each control-loop iteration (e.g., the Policy Enforcer starts after the Forecaster completes). This loop is repeated at a user-defined control period.

A. Monitoring Module

To ensure continuous, non-blocking state estimation, the Monitoring Module executes two concurrent threads, each pinned to a specific tunnel interface. These threads perform high-frequency active probing of the target cloud endpoint to capture real-time network dynamics.

At each time step t, the module collects a metric vector for each tunnel $x \in \{A, B\}$ comprising Round-Trip Time ($RTT_x(t)$), Jitter ($J_x(t)$), and Packet Loss Rate ($L_x(t)$). We utilize fixed-size **Circular Buffers** ($\mathcal{B}_x$) of length m to store these metrics:

$$\mathcal{B}_x(t) = \langle M_x(t-m+1), \ldots, M_x(t) \rangle \tag{1}$$

where $M_x(t) = \langle RTT_x(t), J_x(t), L_x(t) \rangle$. This sliding window design ensures that the Forecaster Module has immediate access to the most recent contiguous context without incurring memory reallocation overhead or garbage collection spikes. Specifically, N probes are dispatched in parallel (with $N = 5$ in our implementation). $RTT_x(t)$ is computed as the average of the valid samples, $J_x(t)$ as ($|RTT_x(t) - RTT_x(t-1)|$), and $L_x(t)$ as the ratio of lost to transmitted probes.

B. Forecaster Module

To enable proactive traffic steering, this module executes two parallel inference threads, isolating the prediction workload for each WAN interface. We utilize a pre-trained TSFM that treats network telemetry as a tokenized sequence, allowing for zero-shot generalization to unseen network conditions.

At each control interval t, the forecaster extracts the RTT time series from the historical context $\mathcal{B}_x(t)$ and generates a probabilistic forecast horizon of k steps. In this design, only the RTT samples in $\mathcal{B}_x(t)$ are used as input to the TSFM, while jitter is derived from the RTT sequence as the absolute difference between consecutive samples.

While not intended for extremely sensitive industrial applications, we consider the use of the **95th percentile (P95)** statistic to safeguard QoS for standard enterprise traffic (e.g.,

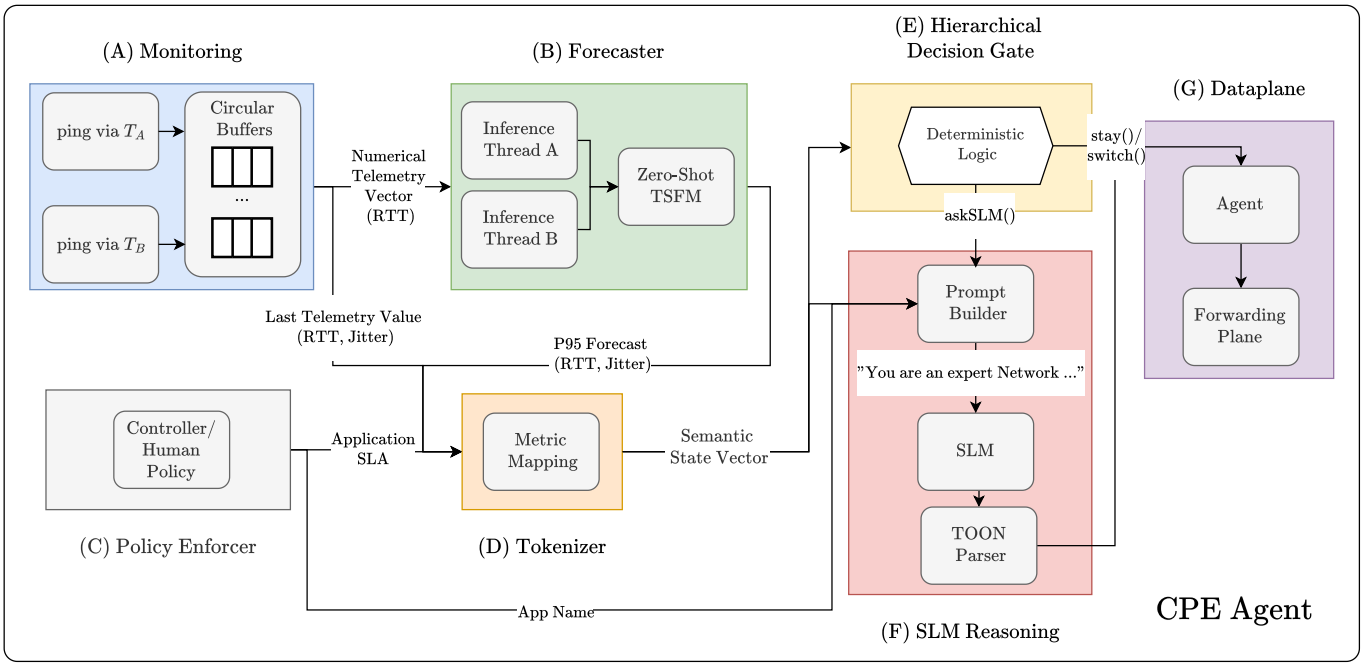

Fig. 1. **The proposed CPE Agent Architecture.** The pipeline functions as a closed-loop controller comprising seven distinct stages: (A) **Monitoring** via multi-threaded probes and circular buffers; (B) **Forecasting** of P95 trends using a Zero-Shot TSFM; (C) **Policy Enforcement** to inject application-specific SLAs; (D) **Tokenization** to bridge numerical metrics with semantic concepts; (E) **Gating** to filter stable states via deterministic logic; (F) **SLM Reasoning** for resolving complex arbitration scenarios; and (G) the **Dataplane** interface for executing traffic steering actions.

VoIP). The module outputs a prediction vector $\hat{V}_x(t)$ for each tunnel:

$$\hat{V}_x(t) = \langle \widehat{RTT}_x^{\text{P95}}, \hat{J}_x^{\text{P95}} \rangle \tag{2}$$

where $\widehat{RTT}_x^{\text{P95}}$ and $\hat{J}_x^{\text{P95}}$ denote the 95th percentile values extracted over the k-step forecast horizon. Note that future jitter is explicitly derived from the forecasted RTT sequence as the absolute difference between consecutive samples, i.e., $\hat{J}_x(t+i) = |\widehat{RTT}_x(t+i) - \widehat{RTT}_x(t+i-1)|$, before calculating its P95 statistic. By optimizing for the tail of the distribution (P95) rather than the average (P50), the agent becomes robust against stochastic volatility and transient micro-bursts, ignoring single isolated spikes.

C. Policy Enforcer Module

This module functions as the interface between the central control plane and the local agent, injecting business intent into the decision loop. The control plane provisions a "Critical Flow Fingerprint" (typically a 5-tuple) associated with a specific application profile.

For the active critical flow f, the module enforces a strict requirement tuple R_f:

$$R_f = \langle \tau_{req}, J_{req}, \phi_{req}, \text{ID}_{app} \rangle \tag{3}$$

where τ_{req}, J_{req}, and ϕ_{req} represent the maximum tolerable RTT, jitter, and packet loss, respectively. Additionally, the semantic tag ID_{app} (e.g., "SSH Terminal (Category: Admin/CLI)") is preserved to provide linguistic context to the SLM in the reasoning stage.

D. Tokenizer Module

To bridge the representational gap between high-precision numerical telemetry and the semantic domain, this module converts continuous metrics into discrete linguistic concepts. This design choice is motivated by evidence that LLMs exhibit fundamental limitations in numerical reasoning and comparison tasks [24]. We define a mapping function $\mathcal{T}(v, v_{req})$ that accepts a measured or predicted value v and its corresponding requirement v_{req}.

The mapping is governed by three tunable coefficients α, β, and γ which define the sensitivity boundaries of the system. The output token is determined as follows:

$$\mathcal{T}(v, v_{req}) = \begin{cases} \text{"EXCELLENT"} & \text{if } v < \alpha \cdot v_{req} \\ \text{"OK"} & \text{if } \alpha \cdot v_{req} \leq v < \beta \cdot v_{req} \\ \text{"POOR"} & \text{if } \beta \cdot v_{req} \leq v \leq \gamma \cdot v_{req} \\ \text{"VIOLATED"} & \text{if } v > \gamma \cdot v_{req} \\ \text{"UNKNOWN"} & \text{if } v \text{ is not available} \end{cases} \tag{4}$$

This transformation is applied to all measured metrics (RTT_x, J_x, L_x) and predicted P95 trends $(\widehat{RTT}_x^{P95}, \hat{J}_x^{P95})$ for each tunnel x. The result is a concise "Semantic State Vector" that abstracts numerical noise while preserving the severity of network conditions relative to the application's specific needs. The `UNKNOWN` token is used when a metric cannot be retrieved (e.g., RTT is undefined in the presence of severe packet loss).

E. Hierarchical Decision Gate

To reconcile the trade-off between control loop latency and reasoning depth, we employ a tiered decision strategy. This module acts as a deterministic filter, resolving unambiguous network states via lightweight heuristics while reserving the computationally expensive SLM for complex arbitration.

The module outputs one of three directives based on the semantic state vectors of the active tunnel (T_{active}) and the backup tunnel (T_{backup}):

- `stay()`: Triggered if all semantic tokens for T_{active} (collected and predicted) are $\in$ {"EXCELLENT", "OK"}. The system maintains the current route to prevent unnecessary flapping.
- `switch()`: Triggered if *any* metric in T_{active} is "POOR" or worse **AND** all tokens in T_{backup} are "OK" or better. This represents the "Fast Path" failover for obvious outages.
- `ask_SLM()`: Triggered in all remaining scenarios where both tunnels exhibit "POOR" or "VIOLATED" semantic states. In these ambiguous conditions, the decision requires contextual arbitration to determine which performance metric should be prioritized for the active application.

This hybrid approach ensures that the CPE Agent operates with sub-second latency for most of the routine operations, invoking the SLM only when semantic nuance is required.

F. SLM Reasoning Module

The decision-making core is a quantized SLM responsible for resolving the ambiguous network states identified by the Hierarchical Decision Gate. This module is invoked exclusively when the gating signal is `ask_SLM()`, ensuring that expensive inference cycles are spent only on non-trivial routing dilemmas.

1) Prompt Construction and Strategy: At runtime, the module constructs a dynamic prompt context combining the ID_{app} (provided by the Policy Enforcer) and the semantic state vectors from the Tokenizer. To guide the correct reasoning of the SLM, we employ a bipartite prompt engineering strategy:

1) **Role Prompting [25]:** We define the system persona as a "Network Reliability Agent," constraining the model's latent space to the domain of traffic engineering and minimizing hallucination risks.
2) **In-Context Learning (Few-Shot) [26]:** Domain-specific decision logic is injected through a small set of curated examples. For example, the prompt illustrates that for latency-sensitive applications with high switching cost (e.g., "VoIP"), maintaining a "POOR" link may be preferable to triggering a tunnel switch that could disrupt the session, whereas for applications with low switching cost (e.g., "Streaming"), proactive switching is favored. This approach enables policy generalization without requiring model fine-tuning.

2) Token-Oriented Object Notation (TOON): To minimize inference latency on edge hardware, we forego verbose output formats like JSON in favor of a novel **Token-Oriented Object Notation (TOON)** [27]. The model is constrained to output a single line using a pipe-delimited syntax:

```
Action | Reason
```

This format significantly reduces the token count compared to structured JSON (eliminating braces, keys, and quotes), resulting in faster generation times and higher accuracies [28]. The output is grammar-constrained and parsed via a computationally inexpensive string split operation[1].

3) Decision Execution: The extracted action a_t is strictly mapped to the binary set $\{\texttt{STAY}, \texttt{SWITCH}\}$ and applied via Dataplane module function call.

G. Dataplane Module

The final stage of the pipeline is the execution layer, which translates high-level steering decisions into kernel-space routing rules. This module is designed to be stateless and lightweight (see Section IV-F), exposing a minimal API to the upper layers of the control loop:

- `get_active_tunnel()`: Queries the underlying OS to identify the interface currently carrying the target application flow.
- `switch_tunnel(flow_id, tunnel_id)`: Atomically updates the routing table to steer the specific 5-tuple flow to the designated WAN interface.

IV. Experiments

This section first details the experimental setup, followed by an isolated validation of the Forecaster and SLM components. Subsequently, we compare our solution against a traditional threshold-based algorithm using a real-world dataset modified to simulate challenging network conditions. Finally, we assess the computational feasibility of the proposed architecture on Commercial Off-The-Shelf (COTS) hardware without AI acceleration, representative of typical CPE deployments.

A. Experimental Setup

Hardware. All experiments were executed on a single host equipped with an Intel Core i7-9700 CPU and 16 GB of RAM, running Ubuntu 22.04 LTS. No dedicated NPU or GPU acceleration was utilized. To emulate edge constraints, the CPE resources were restricted via Linux `cgroups`.

Software and Models. The forecasting module utilizes the Amazon Chronos Bolt TSFM (Tiny variant, 9M parameters). The reasoning module employs Qwen3-0.6B [29] (Unsloth GGUF Q4_K_M version[2]), quantized to 4-bit precision to minimize memory footprint and inference latency. The pipeline is implemented in Python 3.10, leveraging `llama.cpp` for SLM inference and the `chronos-forecasting` library.

[1]The complete prompt is publicly available at https://github.com/giacomosguotti/agentic-ai-cpe

[2]https://huggingface.co/unsloth/Qwen3-0.6B-GGUF

Inference and Control Loop. The SLM is configured with a temperature of 0.1 and a maximum generation limit of 25 tokens. To minimize latency, model reasoning features were disabled. The control loop executes at a 5-second interval. To prevent failures from malformed outputs, we apply constrained decoding as described in [30].

Tokenizer parameters. For Eq. 4, we set $\alpha = 0.7$, $\beta = 0.9$, and $\gamma = 1.1$. These empirical values are densely quantized around the SLA threshold to accurately capture boundary conditions; their fine-tuning remains future work.

B. Validation of Zero-Shot Forecasting

Before integrating the forecasting module into the real-time control loop, it is essential to validate the zero-shot performance of our chosen TSFM (Amazon Chronos Bolt) on real-world network data. Unlike synthetic datasets, real-world WAN traffic exhibits stochastic volatility, packet reordering, and sudden jitter spikes.

The objective of this section is twofold:

1) To assess the generalizability of the pre-trained Chronos Bolt model on unseen, untrained network data.
2) To perform a sensitivity analysis on the Context Window (W_c) and Prediction Horizon (W_p) to identify the optimal trade-off between forecasting accuracy and inference latency.

This evaluation was performed using a real-world dataset comprising RTT measurements collected via a Linux-based CPE deployed in Milan, Italy, connected to a VPS hosted in Helsinki, Finland through a WireGuard VPN tunnel built on top of a consumer 4G/LTE access network. The collection spanned a duration of approximately 11.5 hours with a nominal probing frequency of $f_s = 5\,\text{Hz}$ (200 ms interval).

The evaluation methodology employs a non-overlapping sliding window strategy. Let t denote the discrete time step index. Because the sampling frequency is 5 Hz, the physical time windows W_c (**Context Window**) and W_p (**Prediction Horizon**) translate to discrete sample counts $n_c = W_c \times f_s$ and $n_p = W_p \times f_s$, respectively.

At each step t, the model ingests a context vector of exactly n_c samples, denoted as $X(t - n_c + 1 : t)$, and predicts the subsequent n_p samples, denoted as $\hat{Y}(t+1 : t+n_p)$. Inference is performed strictly in zero-shot mode using the pre-trained model. For the next iteration, the window slides forward by n_p steps. The model then predicts $\hat{Y}(t + n_p + 1 : t + 2n_p)$, taking as input the updated context $X(t+n_p-n_c+1 : t+n_p)$. We conducted a grid search over varying lengths of physical time $W_c \in \{5, 10, 30, 60, 120\}$ s and $W_p \in \{2, 5, 10, 30\}$ s. Performance is quantified using three accuracy metrics (MAE, MAPE, and RMSE) as well as inference time. Figure 2 illustrates the performance landscape across the hyperparameter grid.

First, regarding the Context Window (W_c): increasing the historical context initially yields significant accuracy gains. As shown in the MAE heatmap, extending W_c from 5 s to 30 s reduces error by approximately 8% (from 2.09 ms to 1.93 ms). However, diminishing returns are observed beyond

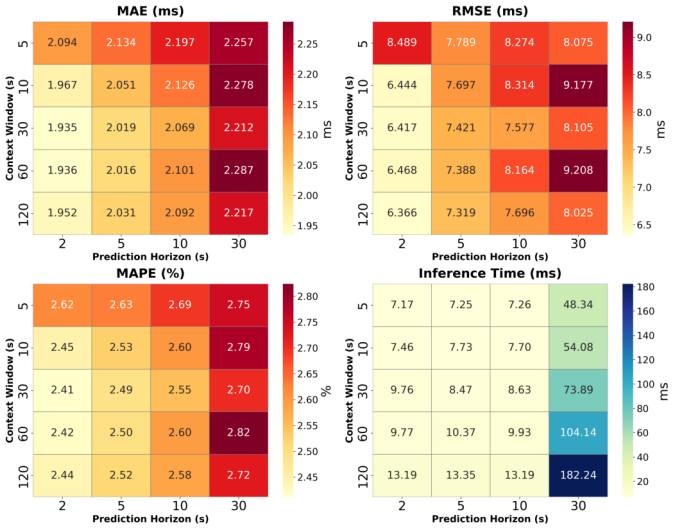

Fig. 2. **Sensitivity Analysis of Zero-Shot Forecasting.** Heatmaps display the impact of Context Window (W_c) and Prediction Horizon (W_p) on error metrics and inference time. While accuracy stabilizes at $W_c \geq 30\,s$, inference latency spikes disproportionately at $W_p = 30\,s$.

$W_c = 60\,\text{s}$, suggesting that immediate jitter dynamics are sufficiently captured within a one-minute window.

Second, regarding the Prediction Horizon (W_p): error rates naturally increase with the horizon length. More critically, the Inference Time heatmap reveals a non-linear "latency cliff." Predicting short horizons (2–10 s) requires only 7–13 ms per inference. However, extending W_p to 30 s causes a $14 - 26\times$ spike in computation time (up to $\approx 182\,\text{ms}$).

C. Validation of SLM Performance

This subsection evaluates the decision-making capabilities of the quantized SLM in resolving ambiguous traffic steering scenarios. To this end, we constructed a custom validation dataset comprising 10 distinct routing dilemmas. Each scenario presents a conflict between competing QoS objectives (e.g., low latency vs. packet integrity) for diverse application profiles, ranging from real-time VoIP to bulk data transfer. These scenarios were specifically designed to fall outside the deterministic logic of the *Hierarchical Decision Gate*, thereby triggering the `ask_SLM()` condition.

We treat the human-expert decision as the Ground Truth and compare it against the SLM's output. Figure 3 presents the resulting confusion matrix. The model achieves an overall accuracy of 70% and a precision of 100% for the `SWITCH` class (no False Positives). This indicates that the agent is highly conservative: it never triggers a tunnel switch unless it is strictly necessary, thereby eliminating the risk of unnecessary flapping.

However, the recall of 40% reveals a limitation: the model tends to default to `STAY` in highly ambiguous edge cases (False Negatives). Despite this, the F1-score of 0.57 confirms that even a highly compressed 0.6B model can effectively reason about network semantics, providing a viable safety net for edge decision-making.

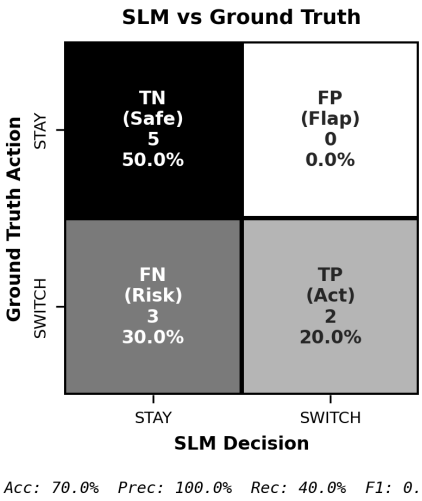

Fig. 3. **SLM Decision Accuracy.** Confusion matrix evaluating the SLM against 10 expert-labeled routing dilemmas. The model exhibits conservative behavior (100% Precision), effectively preventing unnecessary flapping (0 False Positives), though at the cost of missed optimization opportunities (Low Recall) in highly ambiguous scenarios.

D. Lab-in-a-Box Testbed

The experiments presented in Sections IV-E and IV-F were conducted using the emulation environment illustrated in Figure 4. The testbed utilizes Linux Network Namespaces to create isolated virtual nodes that share a single kernel, effectively mimicking a multi-node Wide Area Network (WAN) deployment without the logistical overhead of physical hardware. The topology consists of two primary namespaces: the **CPE Agent** (left) and the **Server** (right).

To simulate a multi-homed connectivity scenario, the CPE is connected to the Server via two distinct virtual Ethernet pairs (depicted as 'veth A' and 'veth B' in the figure). These pairs represent two independent overlay tunnels, referred to as Tunnel A and Tunnel B. The Server namespace acts solely as the tunnel endpoint and a sink for ICMP Echo Requests generated by the CPE's monitoring module. While our current prototype relies on the `ping` tool for practical feasibility, future iterations could integrate industry-standardized protocols (e.g., One-Way Active Measurement Protocol (OWAMP) [31]).

Network impairments are controlled via the Linux Traffic Control (`tc`) subsystem using the Network Emulator (`tc-netem`) queuing discipline [32]. This toolset allows for the precise injection of synthetic latency distributions, jitter profiles, and stochastic packet loss directly onto the egress queues of the virtual interfaces, enabling reproducible evaluation of the steering logic under degraded conditions.

E. Comparison with Threshold-Based Method

To quantify the benefits of agentic steering, we compare our solution against a threshold-based baseline. We selected "Microsoft Teams" as the target application, given its high sensitivity to both latency and connection interruptions.

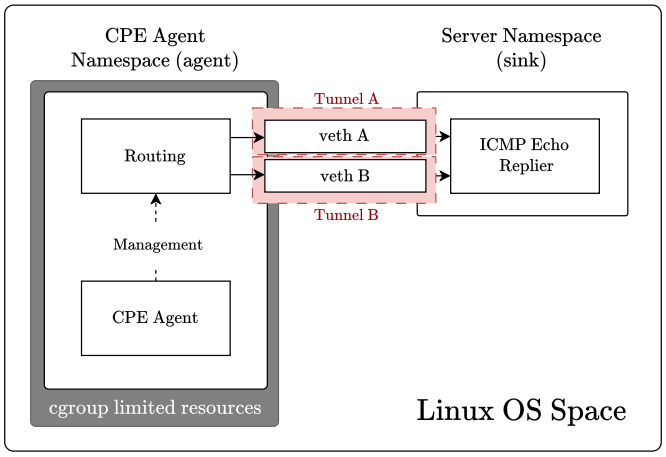

Fig. 4. **The "Lab-in-a-Box" Emulation Testbed.** The topology utilizes Linux Network Namespaces to isolate the CPE Agent (agent) and Server (sink). The CPE Agent manages two parallel WAN paths (Tunnel A and Tunnel B), where network impairments (latency, jitter, and packet loss) are injected dynamically via `tc-netem`.

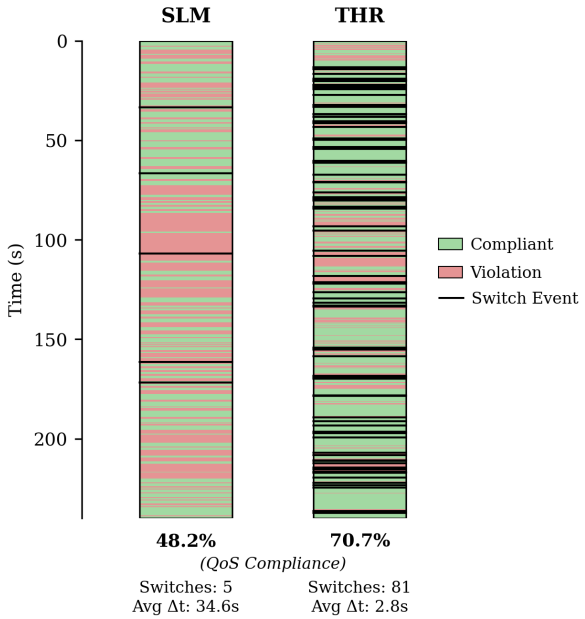

Fig. 5. **Stability vs. Compliance.** Comparison of traffic steering behavior over a 240-second window. The reactive Threshold (THR) approach maximizes theoretical compliance (70.7%) but causes severe flapping (81 switches). The Agentic SLM sacrifices marginal compliance (48.2%) to maintain session stability (5 switches), preventing a possible QoE degradation for the VoIP application.

We replayed a hybrid network trace consisting of a stable but latency-constrained fiber link (Tunnel A) and a highly variable 4G link (Tunnel B). To create a challenging decision boundary, the fiber RTT was offset to hover near the application's maximum tolerable latency, while the 4G trace introduced stochastic jitter. As a baseline, we implemented a reactive **Threshold (THR)** policy:

1) `STAY` if current RTT $< \tau_{req}$.

2) `SWITCH` if current RTT $> \tau_{req}$ & backup RTT $< \tau_{req}$.
3) `STAY` otherwise.

The results, visualized in Figure 5, reveal a fundamental trade-off between instantaneous QoS compliance and control stability. The Threshold baseline achieves a higher raw QoS compliance rate (70.7%) but incurs severe instability, triggering **81 tunnel switches** within a 240-second window (avg. $\Delta t = 2.8$ s). This frequency renders the VoIP session unusable due to constant re-handshakes and jitter buffer resets.

In contrast, our agentic SLM prioritizes stability. By predicting future trends and reasoning over the switching cost, it reduces the switching frequency by **16**× (only 5 switches, avg. $\Delta t = 34.6$ s). Although this results in a lower strict compliance score (48.2%), the agent successfully filters out transient violations. By minimizing the packet loss and jitter bursts associated with frequent handovers, this approach actively mitigates the primary factors known to degrade QoE in VoIP applications [33].

F. Computational Feasibility

To demonstrate that our agentic architecture is viable for deployment on edge devices, we conducted a rigorous resource profiling campaign. The CPE Agent process was strictly confined using Linux control groups (`cgroups`) to a resource budget of **4 vCPUs** and **8 GB of RAM**, emulating the specifications of a mid-range enterprise gateway.

During this evaluation, we employed a stress-test strategy where network metrics were randomly sampled from the four semantic quadrants (EXCELLENT to VIOLATED). This forced the system to explore the full state space and maximized the triggering frequency of the computationally expensive `ask_SLM()` condition.

Figure 6 illustrates the CPU consumption of the entire pipeline. The system operates efficiently within the enforced limits, exhibiting a mean load of **1.65 cores** and a 95th percentile peak of 3.40 cores during SLM inference bursts. This confirms that even without hardware acceleration, the pipeline leaves sufficient CPU headroom for packet forwarding and other control plane tasks. Memory usage remained highly stable throughout the experiment, with a mean footprint of **4.3 GB** ($\sigma = 27.35$ MB). Afterward, we decomposed the end-to-end latency of the control loop to identify bottlenecks. Figure 7 breaks down the contribution of each module.

Two distinct operational modes are observed:

- **Fast Path (Gated):** When the SLM is not invoked, the entire control loop completes in approximately **71 ms** (Average). This confirms that for the vast majority of routine checks, the agent adds negligible overhead.
- **Slow Path (Reasoning):** When the SLM is triggered, the mean execution time rises to **2.25 s**. Crucially, this is well within the predefined 5-second control interval.

As shown in Figure 7, the SLM inference is the dominant cost (≈ 2181 ms). This is consistent with the token generation rate which stabilized at approximately **12 tokens/s**. Given that our prompt strategy limits the output to a maximum of 25 tokens, the inference delay is bounded and predictable.

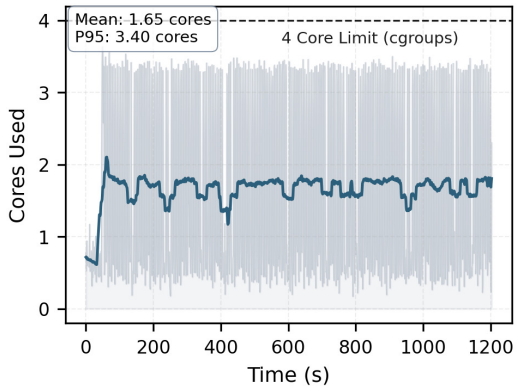

Fig. 6. **CPU Resource Utilization.** The solid teal line represents the rolling average, while the light grey background depicts raw sampling fluctuations. The system respects the 4-core limit even during intensive reasoning bursts.

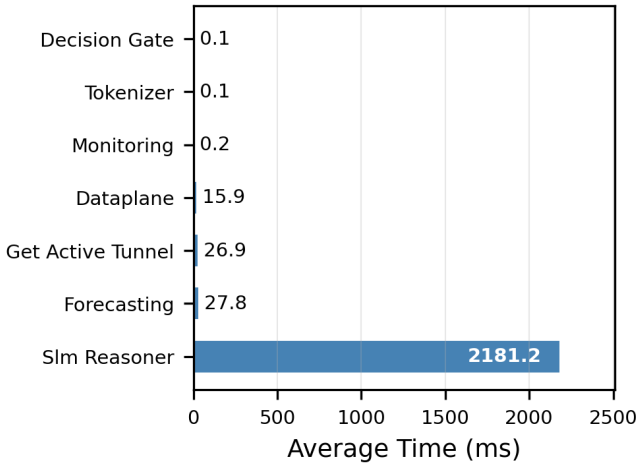

Fig. 7. **Latency Breakdown per Module.** SLM reasoning dominates the latency budget (≈ 2.2 s), while other components operate in the sub-100 ms range. The total time remains comfortably within the 5 s control loop period.

V. Conclusion

In this paper, we presented a novel Agentic AI architecture for SD-WAN traffic steering that shifts the control paradigm from reactive, threshold-based logic to proactive, context-aware reasoning. By decoupling numerical forecasting from semantic decision-making, our approach leverages the complementary strengths of Time Series Foundation Models (TSFMs) and Small Language Models (SLMs). This hybrid design enables the CPE to anticipate network degradation via zero-shot forecasting and resolve complex QoS trade-offs using natural language reasoning, all without requiring site-specific training.

Our experimental evaluation on a "Lab-in-a-Box" testbed demonstrated the practical feasibility of this approach on commodity hardware. We showed that a quantized SLM (0.6B parameters) can effectively reason about application-specific SLAs, reducing route flapping by 16× compared to traditional methods while operating within a strict resource budget of

4 vCPUs and 8 GB RAM. Although the reasoning module introduces a latency overhead (≈ 2.2 s), we verified that the system reliably maintains a 5-second control loop, making it viable for real-time orchestration at the network edge.

Future work will focus on three key directions. First, we aim to reduce inference latency by offloading the SLM workload to Neural Processing Units (NPUs), which are becoming standard in modern consumer CPUs. Second, while our zero-shot approach proved robust, we plan to investigate whether domain-specific fine-tuning of the SLM can improve the recall rate in highly ambiguous scenarios, provided that a sufficiently large labeled dataset of routing dilemmas can be curated. Finally, we intend to benchmark a wider array of emerging TSFMs and SLMs to continuously optimize the trade-off between reasoning depth and computational efficiency.

REFERENCES

[1] S. Troia, L. Borgianni, G. Sguotti, S. Giordano, and G. Maier, "A comprehensive survey on software-defined wide area network," *IEEE Communications Surveys & Tutorials*, 2025.

[2] P. Willars, E. Wittenmark, H. Ronkainen, C. Östberg, I. Johansson, J. Strand, P. Lédl, and D. Schnieders, "Enabling Time-Critical Applications over 5G with Rate Adaptation," Ericsson & Deutsche Telekom, White Paper, May 2021, accessed: 2026-01-26. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/white-papers/enabling-time-critical-applications-over-5g-with-rate-adaptation

[3] Hewlett Packard Enterprise, "What is a Self-Driving Network?" https://www.hpe.com/us/en/what-is/self-driving-network.html, 2025, accessed: 2026-01-26.

[4] S. Troia, F. Sapienza, L. Varé, and G. Maier, "On deep reinforcement learning for traffic engineering in SD-WAN," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2198–2212, 2020.

[5] L. Giacometti, I. Cerrato, H. Lu, F. Lucrezia, L. Oliva, T. Scordo, K. Selvamuthukumaran, G. Sguotti, S. Troia, and G. Verticale, "A MARL Approach to Employ Intelligent Traffic Steering in SD-WAN," in *2025 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2025, pp. 1–6.

[6] A. F. Ansari, L. Stella, C. Turkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S. S. Rangapuram, S. P. Arango, S. Kapoor *et al.*, "Chronos: Learning the language of time series," *arXiv preprint arXiv:2403.07815*, 2024.

[7] A. Das, W. Kong, R. Sen, and Y. Zhou, "A decoder-only foundation model for time-series forecasting," in *Forty-first International Conference on Machine Learning*, 2024.

[8] X. Li, Z. Lu, D. Cai, X. Ma, and M. Xu, "Large language models on mobile devices: Measurements, analysis, and insights," in *Proceedings of the Workshop on Edge and Mobile Foundation Models*, 2024, pp. 1–6.

[9] ByteShape Team, "A 30B Qwen Model Walks Into a Raspberry Pi… and Runs in Real Time," Jan. 2026, accessed: 2026-01-19. [Online]. Available: https://byteshape.com/blogs/Qwen3-30B-A3B-Instruct-2507/

[10] J. Geerling, "Raspberry Pi's new AI HAT adds 8GB of RAM for local LLMs," Jan. 2026, accessed: 2026-01-19. [Online]. Available: https://www.jeffgeerling.com/blog/2026/raspberry-pi-ai-hat-2/

[11] Z. Guo, C. Li, Y. Li, S. Dou, B. Zhang, and W. Wu, "Maintaining the Network Performance of Software-Defined WANs With Efficient Critical Routing," *IEEE Transactions on Network and Service Management*, 2023.

[12] O. Lemeshko, M. Yevdokymenko, O. Yeremenko, A. M. Hailan, P. Segeč, and J. Papán, "Design of the fast reroute QoS protection scheme for bandwidth and probability of packet loss in software-defined WAN," in *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*. IEEE, 2019, pp. 1–5.

[13] M. Ye, B. Han, X. Fang, X. Zou, X. Bao, X. Xie, S. Xiao, Y. Lin, and H. J. Chao, "Dynamic Path Switching for Traffic Engineering in SD-WAN with eBPF," in *2025 IEEE 26th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2025.

[14] RouteViews, Accessed: 2026-02-06. [Online]. Available: http://www.routeviews.org/

[15] Y. Zhang, H. Zhang, P. Cong, W. Wang, and K. Xu, "Grandet: Cost-aware Traffic Scheduling without Prior Knowledge in SD-WAN," in *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. IEEE, 2023, pp. 1–10.

[16] A. Botta, R. Canonico, A. Navarro, S. Ruggiero, and G. Ventre, "AI-enabled SD-WAN: the case of Reinforcement Learning," in *2022 IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE, 2022, pp. 1–6.

[17] L. Borgianni, S. Troia, D. Adami, G. Maier, and S. Giordano, "Assessing the Efficacy of Reinforcement Learning in Enhancing Quality of Service in SD-WANs," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 1765–1770.

[18] F. Liu, B. Farkiani, and P. Crowley, "Time-Series Foundation Models for ISP Traffic Forecasting," *arXiv preprint arXiv:2511.17529*, 2025.

[19] A. F. Ansari, C. Turkmen, O. Shchur, and L. Stella, "Fast and accurate zero-shot forecasting with Chronos-Bolt and AutoGluon," https://aws.amazon.com/blogs/machine-learning/fast-and-accurate-zero-shot-forecasting-with-chronos-bolt-and-autogluon/, dec 2024.

[20] D. Wu, X. Wang, Y. Qiao, Z. Wang, J. Jiang, S. Cui, and F. Wang, "NetLLM: Adapting Large Language Models for Networking," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 661–678.

[21] A. E. Hachimi, N. D. Cicco, M. Ibrahimi, F. Musumeci, and M. Tornatore, "Flow-Rule Generation for SDN Using LLMs with Retry-Based Deployment Validation," in *2025 21st International Conference on Network and Service Management (CNSM)*, 2025, pp. 1–5.

[22] Z. Lu, X. Li, D. Cai, R. Yi, F. Liu, W. Liu, J. Luan, X. Zhang, N. D. Lane, and M. Xu, "Demystifying small language models for edge deployment," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025, pp. 14 747–14 764.

[23] N. T. Pham, T. Kieu, D. M. Nguyen, S. H. Xuan, N. Duong-Trung, and D. Le-Phuoc, "SLM-Bench: A Comprehensive Benchmark of Small Language Models on Environmental Impacts," in *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025, pp. 21 369–21 392.

[24] H. Li, X. Chen, Z. Xu, D. Li, N. Hu, F. Teng, Y. Li, L. Qiu, C. J. Zhang, L. Qing *et al.*, "Exposing numeracy gaps: A benchmark to evaluate fundamental numerical abilities in large language models," in *Findings of the Association for Computational Linguistics: ACL 2025*, 2025, pp. 20 004–20 026.

[25] M. Shanahan, K. McDonell, and L. Reynolds, "Role play with large language models," *Nature*, vol. 623, no. 7987, pp. 493–498, 2023.

[26] T. Gao, A. Fisch, and D. Chen, "Making pre-trained language models better few-shot learners," in *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: long papers)*, 2021, pp. 3816–3830.

[27] TOON: Token-Oriented Object Notation, https://github.com/toon-format/toon, 2025, accessed: 2026-01-02.

[28] I. Matveev, "Token-Oriented Object Notation vs JSON: A Benchmark of Plain and Constrained Decoding Generation," *arXiv preprint arXiv:2603.03306*, 2026.

[29] Qwen Team, "Qwen3," April 2025. [Online]. Available: https://qwenlm.github.io/blog/qwen3/

[30] B. T. Willard and R. Louf, "Efficient guided generation for large language models," *arXiv preprint arXiv:2307.09702*, 2023.

[31] M. J. Zekauskas, A. Karp, S. Shalunov, J. W. Boote, and B. R. Teitelbaum, "A One-way Active Measurement Protocol (OWAMP)," RFC 4656, Sep. 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4656

[32] S. Hemminger *et al.*, "Network emulation with NetEm," in *Linux conf au*, vol. 5, 2005, p. 2005.

[33] ITU-T, "The E-model: a computational model for use in transmission planning," International Telecommunication Union, Recommendation G.107, June 2015. [Online]. Available: https://www.itu.int/rec/T-REC-G.107