

Toward a World Model for the Linux Kernel Network Stack

Luyao Bai

lbai27@uic.edu

University of Illinois Chicago
Chicago, United States

Xiaoguang Wang

xgwang9@uic.edu

University of Illinois Chicago
Chicago, United States

Abstract

Can operating systems learn a world model of their own behavior? Such a model would capture how system actions affect outcomes, allowing the OS to predict before acting rather than rely on online trial and error.

We present NETORACLE, an initial case of a “world model” for the Linux network stack. It learns from executions collected across diverse workloads, network conditions, and kernel configurations, and predicts the performance impact of configuration changes. At runtime, one live measurement anchors its predictions, while uncertain or incorrect predictions guide new measurements and subsequent fine-tuning. NETORACLE achieves 84% directional accuracy on unseen conditions and reaches an optimal or statistically equivalent configuration in 1–3 live measurements. Trained entirely in emulation, it transfers to a real wide-area path and reduces p99 latency under bufferbloat from 409 ms to 43 ms. These results show that execution feedback can provide the training signal for predictive, self-improving OS world models.

CCS Concepts

• **Networks** → **Network performance evaluation; Transport protocols**; • **Computing methodologies** → **Machine learning**; • **Software and its engineering** → **Operating systems**.

Keywords

Linux network stack, world model, configuration tuning, large language models

ACM Reference Format:

Luyao Bai and Xiaoguang Wang. 2026. Toward a World Model for the Linux Kernel Network Stack. In *The 25th ACM Workshop on Hot Topics in Networks (HotNets '26)*, November 16–17, 2026, Salt Lake



This work is licensed under a Creative Commons Attribution 4.0 International License.

HotNets '26, Salt Lake City, UT, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-3029-0/2026/11

<https://doi.org/10.1145/3847290.3859973>

City, UT, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3847290.3859973>

1 Introduction

The Linux network stack offers dozens of configuration knobs, including congestion control, queueing disciplines (qdiscs), and socket buffer limits. Yet no single configuration works best across all network conditions and workloads. Settings that work well on a short, loss-free path may give poor throughput on a lossy path. Large queues can also cause long delays under load, a problem known as bufferbloat. Linux provides mechanisms to address these problems, but choosing the right configuration for a given path and workload remains challenging.

In our testbed, adjusting a few kernel knobs improves goodput on lossy paths by 10–40× over the default and reduces 99th-percentile (p99) request latency under bufferbloat by 10×. But how should the system find the right configuration? Existing approaches search online, using contextual bandits and Bayesian optimization [1, 2, 20], or more recently LLM-guided tuning [4, 9, 17, 18]. These approaches must evaluate candidate configurations on live traffic, potentially degrading performance during exploration. Simply asking an LLM to predict the effect of a configuration change is not reliable either: prior studies show that LLMs can misunderstand knob semantics and misestimate performance effects [19, 26]. Could we replace online search with prediction? Given a measurement of the current configuration, can a model predict what would happen if one knob changed on the same path and workload?

Model-based reinforcement learning offers a useful perspective on this problem: a world model predicts how an environment will respond to an action without executing that action [6, 8, 14]. For the Linux network stack, the action is a *configuration change*, and the outcome is its impact on performance. Rather than reconstructing the underlying network, such a model could learn from the behavior observable at the host, including throughput, round-trip time, retransmissions, and local queue statistics. Given a measurement under the current configuration, it could predict what would happen if one kernel knob changed. This formulation is particularly attractive for deployment over the public

Internet, where network topology, competing traffic, and remote queue states are largely hidden. The challenge is whether limited observations can provide enough information to predict the effects of configuration changes across different network environments.

NETORACLE explores this question with a small language model trained on measured executions. An automated testbed runs workloads under systematically varied network conditions and kernel configurations. These measurements provide training examples for a 0.8B-parameter model that predicts how changing one knob affects performance. At run-time, one live measurement of the current configuration serves as an anchor: the model predicts performance changes relative to this observation. A gated policy applies a change only when enough sampled predictions agree that the gain exceeds a threshold. To improve the model, an offline loop selects configurations whose sampled predictions disagree, measures their outcomes, and retrains on the expanded dataset. Together, these components connect execution data, performance prediction, and configuration decisions.

We report two main findings. First, execution measurements enable a small model to make useful predictions. Fine-tuned on a few hundred testbed runs, a 0.8B model without retrieved context matches the directional accuracy of a 31B model given 14 retrieved measurements. Without fine-tuning on execution measurements, the same 0.8B model predicts the direction of performance changes no more accurately than random guessing. On network conditions outside the training grid but within the same emulator, it predicts whether performance increases, decreases, or stays unchanged with 84% accuracy, compared with 32% for nearest-neighbor lookup, 63% for gradient-boosted trees, and 47% for random forests trained on identical data.

Second, the predictions help choose network settings. The gated policy reaches the best measured setting or a statistical tie in every evaluated sweep. It needs 1–3 live measurements. On replays of the same sweeps, Bayesian optimization needs 4–6 measurements. A hill climb without the model needs 15–19. We also test the model on a live wide-area path after training it only in emulation. One gated change reduces p99 request latency under bufferbloat from 409 ms to 43 ms.

The model does not transfer equally well to all real paths. On paths from the Pantheon archive [31], bottleneck conditions are unknown to the model. When trained only in emulation, it barely beats a baseline that always predicts no change. Training on measurements from other real paths raises direction accuracy to 0.68. The policy then achieves 94% of the best performance among three congestion-control algorithms (§4). With only one anchor measurement, the model still cannot reliably predict tail metrics dominated by retransmission timeouts. We leave longer measurement windows for future work.

The source code, dataset, and benchmark are open source at <https://github.com/sysec-uic/NetOracle>.

2 Why the Kernel Network Stack Needs a World Model

An operator controls a configuration c spanning congestion control, queue disciplines, and buffer sizing, with a decision space of $5 \times 5 \times 2 = 50$ points, while facing an environment e characterized by workload, round trip time, loss rate, and bottleneck bandwidth that can be observed but not chosen. The objective is to solve $\arg \max_c U(c, e)$, selecting the configuration c that yields the best performance U for the environment e at hand. Every evaluation of U incurs a costly live measurement, and U exhibits sharp nonlinearities that punish naive interpolation: whether additional bandwidth improves performance depends on whether the flow is limited by loss, by buffers, or by bandwidth, and nothing on the surface of e reveals which regime holds. Nor can a fixed recommendation substitute: across the 46 corpus environments with at least five measured configurations, 23 distinct configurations are optimal in at least one scenario. In hindsight, the single best configuration achieves an average of only 77% of the per environment optimum and collapses to 5% in its worst environment, while the kernel default manages a mere 57% and 2%. Whatever makes the decision must explicitly condition on the environment.

The conventional approach is that when engineers observe high tail latency on a path operating a standard deep FIFO queue, they manually reconfigure it to active queue management, deploying queue control algorithms such as CoDel (Controlled Delay) or FQ-CoDel (Fair Queueing CoDel). Our central question is whether a small model can perform this configuration automatically and quantitatively.

Why put a language model at the core of that decision? The conventional alternatives are queueing analysis, packet-level simulation, and learned performance models over configuration features [15, 33, 34]; each excels where its abstraction holds, covering merely one slice of U , which is the product of dozens of subsystems co-evolving with every kernel release. A pre-trained model starts with three assets no bespoke model has: it has absorbed every subsystem’s documentation, commit history, and two decades of netdev debate; each new knob arrives with an accompanying textual description; and the patch stream changing the kernel simultaneously describes the change. The open question is whether these text-borne priors can be made quantitatively reliable. Our evaluation therefore keeps learned performance models as baselines with identical evidence.

Knowing how a network mechanism works is not enough to predict its effect on performance. Without measured examples, the best open-weight model predicts the direction

of change correctly in 90% of RTT tests. Its accuracy falls to 33%, no better than chance for a three-way choice (increase, decrease, or no change), when bottleneck bandwidth changes. General knowledge does not tell the model what limits performance on a given path. Its estimates of the size of the change are off by 2.4–4.9 $\times$ on average. For example, all tested models can explain how `fq_codel` reduces queueing delay [22]. Yet the best zero-shot model predicts only a 2–3 $\times$ reduction. In our measurements, CUBIC [7] with a 1000-packet FIFO has a p99 latency of 505 ms under load. With `fq_codel`, it is about 20 ms, a 25 $\times$ reduction. The models know the mechanism. They need measurements to learn how much it changes performance.

2.1 Two Corpora and a Data Engine

We first construct the *patch corpus* to provide models with domain knowledge regarding stack semantics and evolution. By mining the netdev mailing list archive, the kernel net/git history from 2019 to 2026, and official kernel documentation, we assemble a corpus of 41,758 textual retrieval units. These units provide operational context covering three areas: what each knob does, the rationale behind specific kernel heuristics, and past regressions detailing what broke during prior modifications.

Because static documentation cannot capture dynamic runtime interactions, our data engine automates the collection of the *execution corpus* across systematically varied operating environments. The engine deploys a four-namespace dumbbell topology with offloads disabled, following the composable isolation model of Mahimahi [21]. The testbed independently controls bottleneck bandwidth, queuing disciplines, round trip propagation delays, and packet loss rates via netem [10]. Across four representative workloads covering bulk transfer, request-response, mixed traffic, and short flows, we explore a combinatorial grid spanning 6 congestion control algorithms from CUBIC to BBR [3], 5 qdiscs from FIFO variants to CAKE [11], RTT $\in [10, 160]$ ms, packet loss $\in [0, 2]\%$, bandwidth $\in [8, 400]$ Mbit/s, and 2 buffer presets. Each trial logs end-to-end performance alongside fine-grained kernel observables, including TCP statistics, qdisc drops, and buffer occupancy. An individual run completes in approximately 25 seconds of automated execution, yielding a frozen evaluation matrix of 456 representative configurations. Repeatability benchmarks confirm a stable noise floor, with goodput variation between 0.4% and 12% and median latency variation under 9%.

From these two corpora, we introduce STACKBENCH, a benchmark of 278 prediction tasks across ranking, counterfactual estimation, and quantitative regression under three

evaluation regimes: in-distribution interpolation, unseen congestion control (held-out TCP Westwood), and extreme out-of-distribution extrapolation (our off-grid split: 160 ms RTT and 400 Mbit/s bandwidth). Training data and retrieval indices are strictly partitioned to ensure models never observe measurements or ground truths from the target task.

2.2 The Value of In-Context Measurements

We evaluate five open-weight models (ranging from 8B to 35B parameters) on our STACKBENCH benchmark across four progressive evidence tiers: L0 provides only the target environment description, L1 adds four text passages retrieved from the patch corpus, L2 supplies the 14 nearest real-world execution measurements, and L3 combines both text and measurements. For comparison, two non-LLM baselines receive the exact same L2 empirical evidence: a distance-weighted nearest-neighbor predictor with $k=5$ (k NN) and a gradient-boosted tree (GBT) fitted per task on the context pool. Four central findings (F1–F4) shape the system design in §3 (Figure 1).

F1: measured runs raise accuracy and retrieved text adds nothing. Supplying 14 real-world measurements (moving from L0 to L2) consistently improves directional accuracy across all five models by an average of +0.13. For the best model, directional accuracy increases from 0.69 to 0.87, while its mean relative prediction error drops sharply from 2.44 to 0.28. In contrast, adding retrieved documentation (L1) alters accuracy by only -0.01 on average, and combining text with measurements (L3) degrades performance across all models. Pre-trained weights already contain the necessary domain knowledge, so adding extra text simply crowds out numerical measurements in the prompt.

F2: the LLM extracts more from the same data than tabular baselines. Given identical empirical measurements, the LLM attains 0.87 in directional accuracy, surpassing both k NN (0.70) and GBT (0.78). This advantage is most pronounced in out-of-distribution extrapolation, where models encounter network conditions outside the training grid. In these unseen regimes, k NN accuracy collapses to 0.32, and GBT is limited to 0.58 because decision trees cannot extrapolate beyond their observed feature boundaries. In contrast, the LLM maintains 0.74 directional accuracy, with 71% of its predictions falling within a 1.5 $\times$ relative error bound. Pre-trained weights guide the direction of extrapolation, while live measurements calibrate its scale.

F3: measurements matter more than model size. Adding measurement context helps far more than increasing model parameters. At the L2 evidence level, a compact 9.7B Qwen model outperforms the much larger 35B Qwen variant (0.74 vs. 0.67 in directional accuracy). Similarly, an 8B Gemma

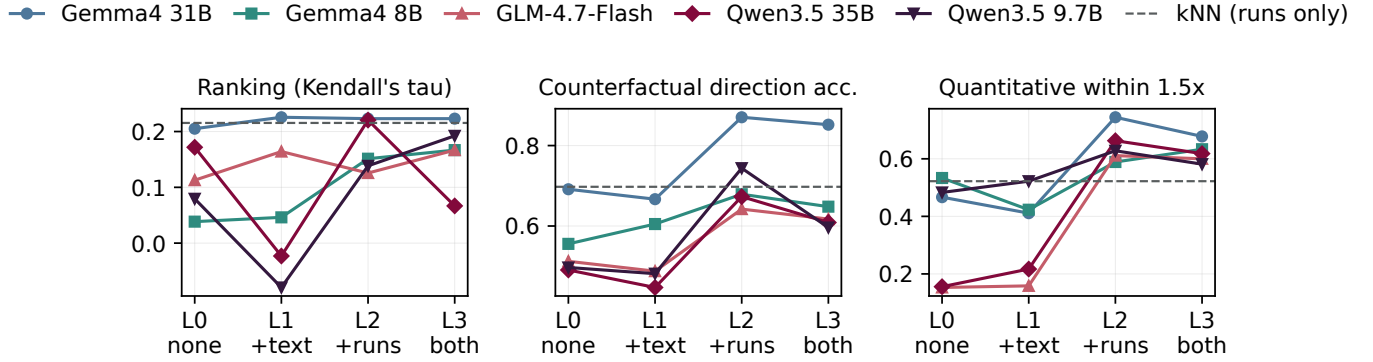


Figure 1: Measurements (L2) improve predictions; retrieved text (L1) adds little; ranking stays poor. Dashed: k NN.

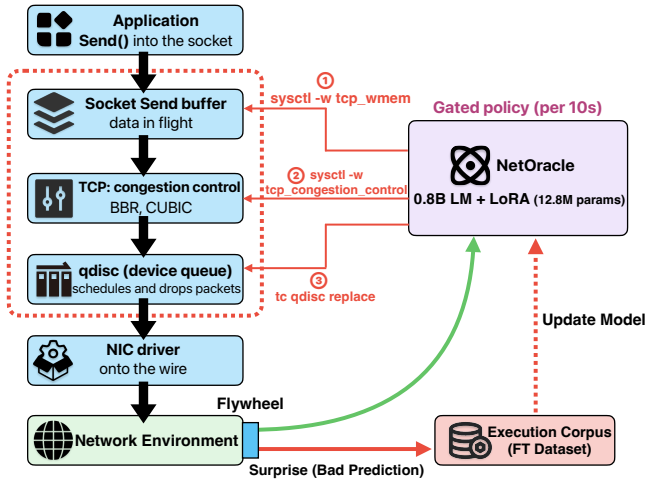


Figure 2: NETORACLE: an offline flywheel trains the world model from testbed runs; an online gate applies one knob to the live host or holds.

model supplied with measurements matches the accuracy of a 31B Gemma model operating without them.

F4: models compare pairs accurately and rank sets poorly. Prompting models to directly rank a set of four to six candidate configurations from best to worst produces poor results across all models and evidence levels, with Kendall's τ never exceeding 0.28, which corresponds to near-random ordering. Yet the same models achieve 87% accuracy when comparing pairs of configurations. Consequently, our system should not request end-to-end rankings in a single prompt; instead, global rankings should be synthesized algorithmically from individual pairwise comparisons.

3 NetOracle

§2.2 showed that empirical measurements are decisive and that a compact model can use them. Rather than passing measurements in the prompt at every query, NETORACLE

```
A measured run on this testbed:
- workload=bulk cc=cubic qdisc=pfifo_deep
  rtt=40ms loss=0.5% bw=50Mbit buf=ample
-> goodput_mbps=4.803 retransmits=21
   cwnd_mid=18 srtt_mid_ms=40.259
```

Now exactly one knob changes:
`cc` goes from cubic to bbr.
Predict the effect on goodput_mbps ...

```
model, 5 samples at temperature 0.8:
{"direction": "increase", "ratio": 3.41}
{"direction": "increase", "ratio": 4.78}
{"direction": "increase", "ratio": 5.85}
{"direction": "increase", "ratio": 7.16}
{"direction": "increase", "ratio": 4.57}
```

```
gate: 5/5 votes, median 4.78x > 1.15
-> move to bbr (measured: 9.7x)
```

Figure 3: One live decision: measured anchor, one-knob question, five sampled answers, gate verdict.

directly internalizes them by fine-tuning the model weights. As illustrated in Figure 2, the architecture decouples into two loops: an offline data flywheel that iteratively trains a compact world model of the network stack, and an online gated policy that uses this model to safely navigate configuration transitions. The online policy evaluates one-knob changes against live telemetry, and a safety gate admits a change only under strong model consensus (Figure 3).

At the core of NETORACLE lies a compact world model specialized for Linux network stack dynamics. We apply LoRA fine-tuning to Qwen3.5 0.8B, training it on two task structures: quantitative regression tasks predicting specific performance metrics, and pairwise counterfactual tasks that predict how performance shifts when adjusting exactly one knob relative to a measured baseline. Target outputs are enforced as strict JSON schemas. Training on 726 distinct operating points (approximately 3,800 examples) takes roughly

20 minutes on a single consumer GPU, making it computationally practical to retrain the world model from scratch during each iteration of the data flywheel and across successive kernel releases.

To expand model coverage efficiently without measuring every combination, NETORACLE uses an active learning data flywheel. In each round, the flywheel proposes candidate configurations, queries the model with $K=5$ samples per candidate, identifies the candidates exhibiting the highest prediction variance (spread), collects real-world measurements for them, and retrains the model. Compared to uniform random exploration, this uncertainty-guided acquisition [24] identifies genuine model surprises, defined as empirical measurements deviating by more than 20% from predictions, with 92% precision sustained across rounds, whereas random selection starts at 75% and decays to about 55%. While the sampling spread reliably ranks relative error magnitudes, its nominal 80% confidence interval covers the ground truth only 47% of the time; the online policy therefore treats the K samples as an ensemble vote rather than a formal posterior.

At runtime, a gated policy applies configuration adjustments safely. The controller begins at default kernel parameters, adopts current telemetry as a baseline anchor, and evaluates neighboring configurations that change exactly one knob. The safety gate permits a transition if and only if at least 4 out of 5 sampled counterfactual predictions agree on a performance gain of at least 15%, as shown in Figure 3. Consequently, policy execution never degrades performance below the baseline default, and every executed move is validated against the telemetry of the subsequent operational epoch. NETORACLE supports two practical deployment modes: on GPU-equipped hosts, a background daemon queries the world model directly, requiring approximately 8 seconds per inference decision, which comfortably overlaps with 10-second control epochs; on hosts without hardware acceleration, model predictions are pre-compiled offline into a compact hint table, an approach analogous to [30]. Requiring under a kilobyte per environment to map operating regimes directly to tuning decisions, this hint table allows NETORACLE to optimize production network stacks without hosting the model on the target machine.

4 Does It Work?

Our evaluation utilizes the 218 held-out tasks from our STACKBENCH benchmark whose target operating conditions were strictly withheld during fine-tuning, with every baseline evaluated on the exact same 218 tasks. Results for fine-tuned models are averaged across 3 random training seeds, and primary performance differences report paired-bootstrap 95% confidence intervals (CIs).

Model	Direction acc.		Quantitative	
	All	Off-grid	Med. APE	Within 2×
<i>k</i> -NN lookup	0.70	0.32	0.38	0.36
GBT (same data)	0.72	0.63	0.23	0.93
Random forest (same data)	0.78	0.47	0.14	0.86
Gemma4 31B + L2 context	0.84	0.74	0.02	0.93
Qwen3.5 0.8B base (zero-shot)	0.29	0.26	no parseable output	
NETORACLE 0.8B	0.78	0.84	0.10	0.88
NETORACLE 2B	0.81	0.79	0.03	0.91
NETORACLE 9B	0.82	0.68	0.03	0.86

Table 1: Prediction on 218 unseen STACKBENCH tasks. Off-grid: 160 ms RTT or 400 Mbit/s, outside training grid. Med. APE: median absolute percentage error. Within 2×: predictions within 2× of measurement. The base model produced no parseable JSON on the quantitative tasks.

Table 1 reports prediction accuracy on the 218 held-out tasks. With zero retrieved context at inference time, the fine-tuned 0.8B model matches the accuracy of the 31B model equipped with full L2 retrieval. On off-grid conditions, NETORACLE 0.8B outperforms every tabular baseline, achieving a +0.53 improvement over *k*NN (95% CI [+0.32, +0.74]) and +0.37 over the random forest (95% CI [+0.16, +0.58]). While scaling up base model capacity to 2B and 9B lowers the median relative prediction error (median APE) to 0.03, their off-grid directional accuracy degrades to 0.79 and 0.68, respectively. This confirms that pre-trained domain priors are the primary driver of out-of-distribution transfer, and the smallest model retains them best.

Why does increased model capacity impair off-grid generalization? With only a few thousand training pairs, fine-tuning larger base models (such as the 2B and 9B variants) fits the measured training points too tightly, memorizing discrete operating instances rather than learning the generalized boundary regimes that govern performance transitions. This tendency remains consistent across larger evaluation sweeps (yielding 0.67, 0.60, and 0.63 directional accuracy for 0.8B, 2B, and 9B models, respectively). Model capacity does not need to scale with network topology complexity: the model never ingests raw network topologies, but only observes the host’s end-to-end telemetry signature, which remains constant in dimensionality regardless of path length. What truly must scale with complexity is the corpus of empirical interventions, which is expanded by operating the automated testbed rather than adding model parameters.

Figure 4b explains why imperfect accuracy still leads to safe decisions: prediction errors concentrate in regimes where two configurations perform nearly identically. On large performance differences (greater than 2×), where choosing the

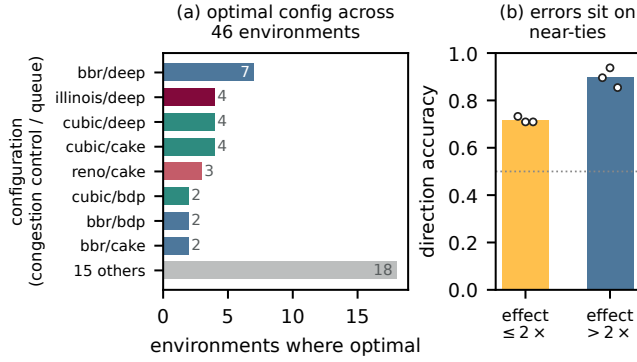


Figure 4: (a) 23 optimal configurations (congestion control / queue and buffer preset) across 46 environments. (b) NETORACLE 0.8B errors concentrate on near-ties; accuracy is near oracle on large effects.

Policy variant	Opt./tied	Worst outcome	Meas.
Kernel default	4/12	40× off optimum	0
Model argmax (unanchored)	8/12	7× worse than default	0
+ Anchored counterfactuals	12/12	Ties optimum	≤ 2.1
+ 15% Consensus gate (NETORACLE)	12/12	Ties optimum	1–3
+ Strict unanimity (5/5 votes)	8/12	Misses real wins	1–2

Table 2: Policy ladder replayed on 12 environments. Anchoring removes every regression; strict unanimity misses real wins.

wrong setting would genuinely degrade performance, directional accuracy reaches 85–94% across training seeds. In contrast, for mispredicted pairs, the true performance difference is typically small (only 1.1–1.4×), falling within our $\pm 10\%$ experimental noise margin. The model rarely errs on high-stakes decisions, so a 15% gate margin suffices.

Model predictions translate into safe operational decisions through our gated ladder, which Table 2 systematically ablates across 12 representative environments spanning 48 configuration trajectories. Without anchoring queries to live telemetry, greedily selecting the configuration with the highest predicted performance captures bulk-goodput gains but inadvertently triggers severe regressions, occasionally choosing a setting whose 99th-percentile latency is 7× worse than the default. In that environment, severe packet loss causes tail latency to fluctuate bimodally across runs, making unanchored point predictions unreliable. Anchoring model queries to current live telemetry completely eliminates regressions across the 48-setting replay. Furthermore, our consensus gate with a 15% margin restricts live exploration to just 1–3 measurements per path while matching the optimal configuration or achieving a statistical tie on every trial; in contrast, demanding unanimous agreement (5/5 samples) renders the policy overly timid. Replayed on the

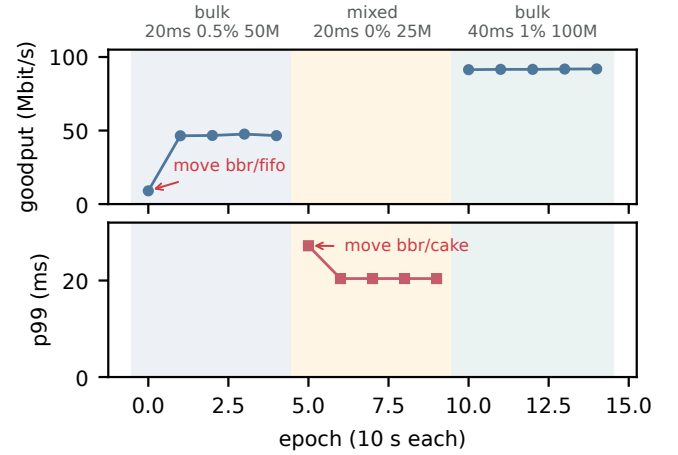


Figure 5: Three traffic phases (shaded), the gate re-run every 10 s: one move per change, no oscillation.

same sweeps, standard Bayesian optimization using Gaussian processes (the search algorithm adopted by classic tuners such as CherryPick [2] and recent LLM tuners [13]) requires a median of 4–6 live measurements to reach 99% of optimal utility (and 13–19 measurements in the worst 10% of cases), whereas climbing without a predictive model requires 15–19 measurements. This performance gap directly quantifies the operational value of pre-trained model weights.

Figure 5 demonstrates end-to-end online performance under dynamic traffic: the policy makes decisive transitions during workload phase changes without exhibiting control oscillations, despite re-evaluating the gate every 10-second epoch. The same control policy also transfers directly to real Internet paths. Deployed on a cloud host and routing traffic across a real Internet path to an academic campus, the policy reduces 99th-percentile latency from 409 ms to 43 ms in a single step. Median latency drops concurrently from 239 ms to 33 ms, confirming the elimination of persistent bufferbloat (standing queues) rather than transient tail luck. On paths already saturating their configured traffic shaper (bandwidth limit), the gate correctly halts exploration, refusing to perturb an optimal state.

The boundary of transfer out of emulation appears on measurements collected by others. First, addressing what telemetry a world model minimally needs, retrained on tasks stating only what an endpoint sees (goodput, smoothed RTT, retransmissions, queue drops and backlog from tcp_info and tc) and nothing about the path, the model maintains full accuracy on the testbed (0.74 vs. 0.73). While our earlier off-grid split fixed the emulator and varied only path parameters, transferring out of emulation presents a harder challenge. On 2,057 counterfactual transitions from 15 real paths in the public Pantheon archive [31], where internal

queue telemetry is missing, the emulation-trained model answers 0.37 directional accuracy (barely beating the 0.35 baseline of predicting no change). However, real paths can be learned by measuring them: training on measured outcomes from other real paths under six-fold cross-validation raises directional accuracy to 0.68. This enables the gated policy to select the best of CUBIC, BBR, and Vegas at 0.94 of optimal utility, against 0.76 for keeping default CUBIC. A random forest trained on the same replay reaches 0.70 direction accuracy. Deployed hosts generate such before-and-after pairs in routine operation (§5).

Finally, the computational economics of NETORACLE mirror software compilation rather than continuous machine learning training. Retraining the compact world model for a new kernel release incurs approximately the same compute cost as compiling the Linux kernel itself, while online inference requires either 8 overlapped seconds on a GPU or a sub-kilobyte table lookup without hardware acceleration. Online search instead pays 4 to 19 live measurements per path, on user traffic, every time.

5 Future Work

Our prototype evaluates a single kernel version in emulation, pointing to four next steps. First, multi-parameter tuning scales linearly because the policy searches one knob at a time in a greedy climb. Unseen knobs enter zero-shot via natural language descriptions; across sixteen measured pairs of two new queueing disciplines (SFQ and PIE), the model predicted every directional outcome without retraining. Second, dynamic traffic is tracked by re-anchoring each epoch (Figure 5), while unexpected online errors provide training data for continuous self-supervision, mirroring our Pantheon results (§4). Third, with one to three competing CUBIC flows, NETORACLE keeps neighbor goodput within -18% to $+5\%$ on seven of eight lossy paths while cutting competitor RTT by 60% to 88% under bufferbloat; adding an explicit neighbor harm check to the gate is the next step. Finally, tail metrics require pooling observations across epochs and calibrating uncertainty, as nominal 80% confidence intervals currently cover true outcomes only 47% of the time.

6 Related Work

Online configuration tuning searches live traffic using bandits or Bayesian optimization [1, 2, 20]; NETORACLE moves this search offline and embeds empirical insights directly into model weights. In databases, DB-BERT and GPTuner use manuals to guide tuning [13, 25], but still evaluate candidates on live systems. PF-LLM [30] is closest in spirit, fine-tuning an LM on simulator data for hardware prefetching; NETORACLE adds the active acquisition loop, runtime safety gate, and

emulation-to-real transfer. LLM-driven tuners and validators [4, 9, 16–19, 26, 28] run or document the trial-and-error pattern that NETORACLE replaces with offline prediction. Learned congestion control synthesizes new algorithms from scratch [5, 12, 27]; NETORACLE instead configures the proven mechanisms the kernel already ships. Learned performance models, from simulation surrogates to graph neural networks and tail estimators [15, 23, 33–35], predict from fixed numerical features rather than text. Pantheon and Puffer showed emulation-trained models diverge from real paths [31, 32]; our Pantheon replay quantifies this boundary and narrows it. Finally, we contribute a concrete kernel-layer recipe for network digital twins [29], grounded in model-based RL principles [6, 8, 14].

7 Conclusion

The knowledge needed to configure the Linux network stack fits within 0.8B parameters, trains from a few hundred testbed executions, and improves continuously by measuring configurations where predictions disagree. This yields tuning decisions that are near-optimal, measurement-frugal, and never worse than kernel defaults in any environment we evaluated. On paths with hidden conditions, training on real measured outcomes reliably adapts the model. While single-measurement tail latency prediction remains challenging, this limitation highlights the need for richer telemetry rather than a flaw in the approach. Linux documentation explains network mechanisms; NETORACLE provides a practical world model that predicts their quantitative impact.

Acknowledgments

We thank our shepherd, Prof. Hong Xu, and the anonymous reviewers for their comments and suggestions. This work is supported in part by US NSF grant CNS 2127491 and a Google Cloud gift award. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of these agencies.

References

- [1] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *ACM SIGMOD International Conference on Management of Data*.
- [2] Omid Alipourfard et al. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.
- [3] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* (2016).
- [4] Huilai Chen et al. 2024. AutoOS: Make Your OS More Powerful by Exploiting Large Language Models. In *International Conference on Machine Learning (ICML)*.

- [5] Mo Dong, Qingxi Li, Doron Zarchy, P. Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*.
- [6] David Ha and Jürgen Schmidhuber. 2018. World Models. *arXiv preprint arXiv:1803.10122* (2018).
- [7] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review* (2008).
- [8] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2020. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations (ICLR)*.
- [9] Zhiyuan He, Aashish Gottipati, Lili Qiu, Yuqing Yang, and Francis Y. Yan. 2025. Congestion Control System Optimization with Large Language Models. *arXiv preprint arXiv:2508.16074* (2025).
- [10] Stephen Hemminger. 2005. Network Emulation with NetEm. In *linux.conf.au*.
- [11] Toke Hoiland-Jørgensen, Dave Täht, and Jonathan Morton. 2018. Piece of CAKE: A Comprehensive Queue Management Solution for Home Gateways. In *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*.
- [12] Nathan Jay, Noga H. Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *International Conference on Machine Learning (ICML)*.
- [13] Jiale Lao et al. 2024. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *Proceedings of the VLDB Endowment* (2024).
- [14] Yann LeCun. 2022. A Path Towards Autonomous Machine Intelligence. *OpenReview preprint* (2022).
- [15] Chenning Li et al. 2024. m3: Accurate Flow-Level Performance Estimation using Machine Learning. In *ACM SIGCOMM Conference*.
- [16] Xinyu Lian et al. 2025. Large Language Models as Configuration Validators. In *IEEE/ACM 47th International Conference on Software Engineering (ICSE)*.
- [17] Georgios Liargkovas, Vahab Jabrayilov, Hubertus Franke, and Kostis Kaffes. 2025. An Expert in Residence: LLM Agents for Always-On Operating System Tuning. In *Machine Learning for Systems Workshop at NeurIPS*.
- [18] Hongyu Lin et al. 2025. BYOS: Knowledge-driven Large Language Models Bring Your Own Operating System More Excellent. *arXiv preprint arXiv:2503.09663* (2025).
- [19] Rajdeep Mondal, Alan Tang, Ryan Beckett, Todd Millstein, and George Varghese. 2023. What do LLMs need to Synthesize Correct Router Configurations?. In *ACM Workshop on Hot Topics in Networks (HotNets)*.
- [20] Usama Naseer and Theophilus A. Benson. 2022. Configanator: A Data-driven Approach to Improving CDN Performance. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*.
- [21] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX Annual Technical Conference (ATC)*.
- [22] Kathleen Nichols and Van Jacobson. 2012. Controlling Queue Delay. *ACM Queue* (2012).
- [23] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2020. RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN. *IEEE Journal on Selected Areas in Communications* (2020).
- [24] Burr Settles. 2009. *Active Learning Literature Survey*. Technical Report 1648. University of Wisconsin–Madison.
- [25] Immanuel Trummer. 2022. DB-BERT: A Database Tuning Tool that “Reads the Manual”. In *ACM SIGMOD International Conference on Management of Data*.
- [26] Changjie Wang et al. 2024. NetConfEval: Can LLMs Facilitate Network Configuration? *Proceedings of the ACM on Networking (CoNEXT)* (2024).
- [27] Keith Winstein and Hari Balakrishnan. 2013. TCP ex Machina: Computer-Generated Congestion Control. In *ACM SIGCOMM Conference*.
- [28] Duo Wu et al. 2024. NetLLM: Adapting Large Language Models for Networking. In *ACM SIGCOMM Conference*.
- [29] Yiwen Wu, Ke Zhang, and Yan Zhang. 2021. Digital Twin Networks: A Survey. *IEEE Internet of Things Journal* (2021).
- [30] Ceyu Xu et al. 2026. PF-LLM: Large Language Model Hinted Hardware Prefetching. In *31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 26)*.
- [31] Francis Y. Yan et al. 2018. Pantheon: The Training Ground for Internet Congestion-Control Research. In *USENIX Annual Technical Conference (ATC)*.
- [32] Francis Y. Yan et al. 2020. Learning in situ: A Randomized Experiment in Video Streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*.
- [33] Qingqing Yang et al. 2022. DeepQueueNet: Towards Scalable and Generalized Network Performance Estimation with Packet-level Visibility. In *ACM SIGCOMM Conference*.
- [34] Qizhen Zhang et al. 2021. MimicNet: Fast Performance Estimates for Data Center Networks with Machine Learning. In *ACM SIGCOMM Conference*.
- [35] Kevin Zhao, Prateesh Goyal, Mohammad Alizadeh, and Thomas E. Anderson. 2023. Scalable Tail Latency Estimation for Data Center Networks. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*.