TRACE: Traceroute-based Internet Route change Analysis with Ensemble Learning

Raul Suzuki Borges[1], **Rodrigo Moreira**[1],
Pedro Henrique A. Damaso de Melo[1], **Larissa F. Rodrigues Moreira**[1],
Flávio de Oliveira Silva[2]

[1]Institute of Exact and Technological Sciences – Federal University of Viçosa (UFV)
Rio Paranaíba – MG – Brazil

[2]Department of Informatics – School of Engineering
University of Minho (UMinho) – Braga – Portugal

{raul.borges, rodrigo, larissa.f.rodrigues}@ufv.br

pedro.henrique.melo@ufv.br, flavio@di.uminho.pt

Abstract. *Detecting Internet routing instability is a critical yet challenging task, particularly when relying solely on endpoint active measurements. This study introduces TRACE, a Machine Learning (ML) pipeline designed to identify route changes using only traceroute latency data, thereby ensuring independence from control plane information. We propose a robust feature engineering strategy that captures temporal dynamics using rolling statistics and aggregated context patterns. The architecture leverages a stacked ensemble of Gradient Boosted Decision Trees refined by a hyperparameter-optimized meta-learner. By strictly calibrating decision thresholds to address the inherent class imbalance of rare routing events, TRACE achieves a superior F1-score performance, significantly outperforming traditional baseline models and demonstrating strong effectiveness in detecting routing changes on the Internet.*

1. Introduction

Enhancing network connectivity involves more than merely increasing the network capacity and speed. Instead, optimization across various layers, from the application to the routing domain, is essential to ensure stability and ultimately provide an acceptable user experience for the application [Alberti et al. 2024, Katsaros et al. 2024]. Programmable and virtualized infrastructures, including network slicing [Moreira et al. 2021], enable finer traffic management that can expose or mitigate routing issues such as route changes, loops, outages, and even subtle attacks [Alaraj et al. 2023, Bhaskar and Pearce 2024].

Advancements in programmability and virtualization have simplified Network Slicing (NS) and, in doing so, transformed network management across many domains. The Internet is naturally collaborative and spread out, which poses challenges in sharing resources and managing them actively. Although there have been attempts to make routing programmable [Moreira et al. 2021], problems such as limited telemetry and measurement approximations still exist [Syamkumar et al. 2022]. While Artificial Intelligence (AI) techniques offer great benefits [Yang et al. 2022], measuring routing behavior remains difficult. Changes in routing can make applications unstable, leading users to

think that the quality and reliability are poor. This tension leads to a concrete problem: detecting Internet routing changes without imposing heavy control-plane monitoring.

Regarding Internet behavior, specifically route changes, we propose a Taceroute-based Internet Route change Analysiswith Ensemble Learning (TRACE) approach, which is a practical avenue for identifying these events, even when hidden by rare occurrences and stochastic scenarios. The main contributions of this paper are as follows: (i)a robust feature engineering strategy that captures statistical, temporal, and aggregate context patterns from raw latency measurements; (ii)a stacked ensemble architecture that integrates LightGBM, CatBoost, and XGBoost to effectively model nonlinear routing dynamics; and (iii)a decision threshold calibration mechanism designed to maximize performance on highly imbalanced data, achieving a higher F1-score and outperforming traditional heuristics.

The remainder of this paper is organized as follows: Section 2 reviews prior studies that closely align with our approach. In Section 3, we present our ensemble design for identifying the routing changes. Section 4 details our experiments, results, In Section 5, we offer conclusions and suggest directions for future research that can be explored.

2. Related Work

The behavior of the Internet, particularly its routing, is intriguing, and understanding it has been the subject of study for some time [Paxson 1996]. This section reviews prior work on Internet routing stability, traceroute-based measurement, and path-change tracking, with emphasis on how different approaches detect and react to route dynamics. We focus on studies that analyze end-to-end path persistence, multipath and load balancing effects, reverse-path inference, and control-plane–assisted monitoring, and we summarize their main methodological traits in Table 1 to highlight the gap addressed by TRACE. It is worth noting that operational Border Gateway Protocol (BGP) monitoring relies on protocols such as BGP Monitoring Protocol (RFC 7854) and tools like BGPalerter, which require direct access to control-plane feeds; TRACE, by contrast, operates entirely from data-plane measurements.

[Islam et al. 2024] addresses whether routers in the current Internet can provide consistent flow identification and path selection when endpoints rely on the Internet Protocol version six flow label together with source and destination addresses. It proposes a large scale active measurement study that extends Paris traceroute to control only network layer header fields, sending parallel traces with different flow label values from multiple worldwide vantage points to observe how load balancers actually use the Internet Protocol version six flow label in practice.

[Li et al. 2022] notes that Border Gateway Protocol Multipath routing is insufficiently understood, particularly regarding how often it is deployed in the Internet and how it actually balances traffic across parallel inter domain border links. The authors mine Looking Glass server data and perform traceroute measurements from the RIPE Atlas measurement platform to infer deployments of Border Gateway Protocol Multipath and to describe the resulting multipath routing behavior and load sharing schemes.

[Lin et al. 2025] examine the limited understanding of how major cloud providers configure distinct network service tiers that use either private wide area networks or the

public Internet and how these choices affect user perceived latency. They design a large scale measurement study that uses geographically distributed probes and cloud regions to reveal the routing behavior of each tier and to explore how a simple performance based routing strategy could further reduce access latency.

[Al-Qudah et al. 2020] revisits the stability and diversity of Internet routes under the widespread deployment of Multi Protocol Label Switching (MPLS), questioning whether earlier low rate measurement studies still capture current end to end path behavior. It analyzes two large traceroute data sets from PlanetLab and Ripe Atlas, using frequent probing and several path equality definitions to quantify route persistence, route prevalence and route diversity and to compare paths that traverse Multi Protocol Label Switching tunnels with paths that do not.

[Schmid et al. 2025] address the challenge of characterizing transient forwarding anomalies such as routing loops and black holes that arise during the convergence process of the Border Gateway Protocol. The authors propose an inference system named TRIX that reconstructs network wide traffic flow intervals from control plane logs by simulating the decision process and modeling the Forwarding Information Base update latency.

[Li et al. 2025] investigate whether reverse traceroute based on the Internet Protocol version four Record Route option can reliably reconstruct Internet paths, given that packets carrying options may traverse different router processing paths than ordinary traffic. They design a customized traceroute tool that injects various Internet Protocol options into Internet Control Message Protocol probes and run a large scale active measurement campaign from multiple worldwide vantage points to compare routing consistency between probes with options and probes without options.

[Shapira and Shavitt 2022] address the significant security challenge of detecting Border Gateway Protocol hijacking attacks that deflect traffic between endpoints and lead to man in the middle attacks. The authors introduce an unsupervised deep learning approach named AP2Vec that detects anomalies by embedding Autonomous System Numbers and Internet Protocol address prefixes into vector representations to identify functional role changes along routing paths.

[Sagatov et al. 2025] investigate how to detect anomalous network behavior and distributed denial of service (DDoS) attacks using Internet Protocol performance metrics, focusing on the limitations of existing one way delay measurement tools and timestamping procedures. They propose a global monitoring architecture based on NetTestBox devices and a new owping2 utility that uses precise kernel timestamps to measure one way delay and relate sudden delay shifts and route changes to the onset of attacks.

[Vermeulen et al. 2022] address the lack of visibility into reverse paths caused by routing asymmetry and the limitations of previous reverse traceroute tools regarding throughput, accuracy, and coverage. The authors present the second version of the Reverse Traceroute system which optimizes probe selection through ingress identification and traceroute atlas intersection while restricting symmetry assumptions to intradomain links to ensure high fidelity.

[Giotsas et al. 2020] examine how systems that depend on traceroutes can keep their view of Internet routing accurate when probing capacity is tightly limited, since simple periodic remeasurement leaves many paths stale and wastes probes on routes that

did not change. They introduce techniques that passively monitor Border Gateway Protocol updates and large public traceroute collections to infer which stored traceroutes have likely changed at the border router level, so that only those paths are selectively refreshed while the remaining traceroutes are safely reused.

[Fazzion et al. 2025] address the high probing overhead and inefficiency inherent in continuous topology mapping systems that perform a complete remapping of the network path whenever a routing change is detected. The authors propose a local remapping technique named RemapRoute that reduces measurement overhead by identifying the specific Local Change Zone using a binary search mechanism and restricting the Multipath Detection Algorithm to only the affected hops.

[Kirci et al. 2024] argue that the widely used Gao Rexford model of Internet routing is too coarse and cannot explain real path diversity because it represents each Autonomous System as a single node with restricted policy expressiveness. They propose a more granular routing model that incorporates internal router level topologies and richer intra domain and inter domain routing policies together with an efficient path finding algorithm that computes router level paths at Internet scale.

[Wassermann et al. 2017] studies the problem of predicting Internet path changes and latency variations from traceroute measurements using supervised machine learning, motivated by the link between routing dynamics, path inflation, and performance degradation. The authors introduce NETPerfTrace, an Internet path tracking system based on decision tree ensembles and empirical feature distributions that forecasts path lifetime, number of path changes, and round trip time while outperforming a previous prediction framework.

[Tian et al. 2019] addresses the challenge of accurately modeling and predicting future Border Gateway Protocol route decisions at the prefix level, since existing coarse and prior-knowledge-based models do not scale to the volume and dynamics of modern inter-domain routing data. The authors propose a data-driven deep learning framework that learns the Border Gateway Protocol decision process from large-scale control-plane traces to predict routing outcomes per prefix and assist in detecting routing dynamics and anomalies.

Table 1. Comparison with Related Work.

Approach	Route-change Detection	Stacking / Deep Learning	Temporal Feature Eng.	Imbalance Handling	Control-Plane Independent	Public Data Compatible
[Islam et al. 2024]	○	○	○	○	○	○
[Li et al. 2022]	○	○	○	○	○	○
[Lin et al. 2025]	○	○	○	○	●	●
[Al-Qudah et al. 2020]	●	○	○	○	●	●
[Schmid et al. 2025]	●	○	○	○	○	○
[Li et al. 2025]	○	○	○	○	●	○
[Sagatov et al. 2025]	●	○	●	○	●	○
[Vermeulen et al. 2022]	○	○	○	○	●	●
[Giotsas et al. 2020]	●	○	○	○	○	●
[Fazzion et al. 2025]	●	○	○	○	○	○
[Kirci et al. 2024]	○	○	○	○	○	○
[Shapira and Shavitt 2022]	●	○	○	○	○	●
[Wassermann et al. 2017]	●	○	●	○	●	●
[Tian et al. 2019]	○	●	○	○	○	●
TRACE (Ours)	●	●	●	●	●	●

Table 1 contrasts TRACE with prior art across six key methodological dimensions, where ● denotes the presence and ○ the absence of a feature. The "Route-change Detection" column identifies works that explicitly aim to detect or predict path changes over time. "Stacking / Deep Learning" distinguishes approaches that employ multi-level model composition (stacking) or deep neural architectures from those relying on single-stage classifiers or heuristics. "Temporal Feature Eng." marks methodologies that engineer features capturing time-series dynamics, such as rolling statistics or historical deltas, rather than treating measurements as isolated snapshots. "Imbalance Handling" highlights studies that explicitly address class imbalance through techniques like threshold calibration, a critical aspect for rare event detection. "Control-Plane Independent" indicates solutions that operate solely on data-plane measurements without requiring privileged access to BGP feeds or router configurations. Finally, "Public Data Compatible" denotes approaches capable of leveraging existing large-scale public datasets without necessitating bespoke active measurement infrastructure. As shown, TRACE uniquely combines these attributes to provide a robust, data-driven detection framework.

3. Proposed Approach

Figure 1 summarizes the TRACE pipeline. Starting from raw traceroute measurements, Phase 1 performs feature engineering and transforms each row into a compact vector that encodes statistical, temporal, and aggregate context. In Phase 2, these engineered features feed three gradient-boosted tree classifiers that act as base learners and produce out-of-fold (OOF) probability estimates. Phase 3 builds a meta-feature vector by combining these OOF predictions with selected original features and simple interaction terms, and trains a LightGBM meta-model on top of them. Finally, Phase 4 trains conventional baseline classifiers directly on the engineered features to isolate the benefit of stacking from the benefit of feature engineering alone.

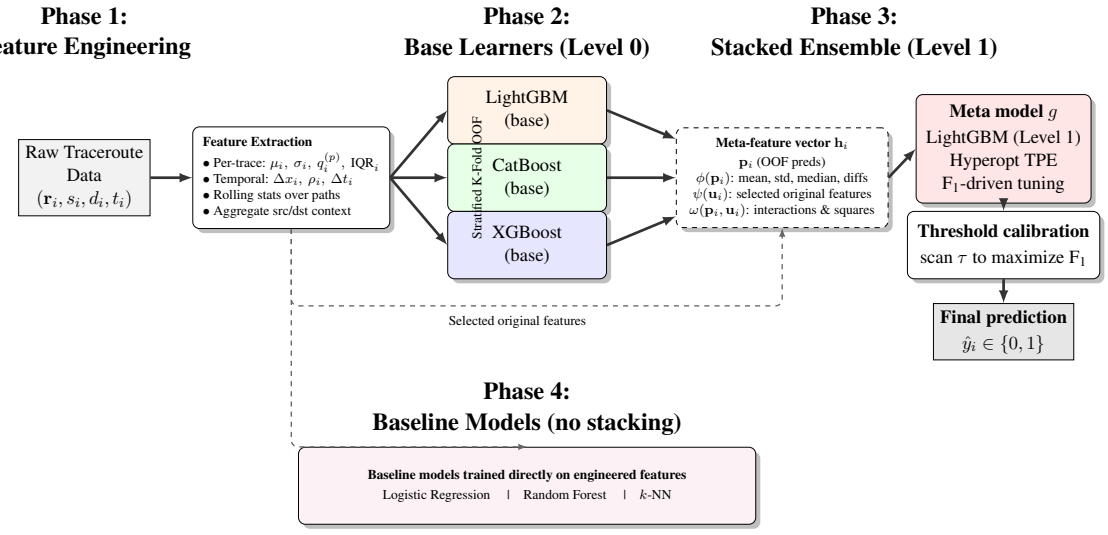


Figure 1. Stacked ensemble architecture (Phases 1–3) and additional baseline models (Phase 4) trained directly on the engineered features for comparison.

3.1. Data and Problem Formulation

The traceroute dataset used in this work is drawn from the Measurement Lab (M-Lab) open repository [Measurement Lab 2025]. For each traceroute, we have the source (`tr_src`), the destination (`tr_dst`), a list of round-trip times (`all_rtts`), probe outcome counters (`total_probes_sent` and `total_replies_last_hop`), and the binary label `route_changed` indicating whether a route change was detected.

After preprocessing, the training set contains N observations. Each observation is represented by an engineered feature vector $\mathbf{x}_i$ and a binary label $y_i \in \{0, 1\}$, where $y_i = 1$ denotes a route change. The goal is to learn a scoring function

$$f(\mathbf{x}_i) \approx \Pr(y_i = 1 \mid \mathbf{x}_i),$$

which we later convert into binary decisions through a calibrated threshold, as detailed in the following subsections.

To evaluate the proposed solution, we use a large-scale traceroute dataset comprising a total of 28,521,656 instances. We adopt a 70% / 30% split in terms of number of rows, with 19,965,159 samples for training and 8,556,497 samples for testing. Each data instance corresponds to a traceroute execution characterized by the source and destination identifiers, its round-trip time (RTT) vector, and the total number of probes sent, which averages 1.44 probes per traceroute.

Table 2. Dataset statistics and class distribution.

Category	Count	Percentage	Description
Total training	19,965,159	100.00%	Full training set
Class 0 (stable)	19,595,589	98.15%	Majority class
Class 1 (changed)	369,570	1.85%	Minority class (target)
Total testing	8,556,497	–	Unlabeled split (30%)
Imbalance ratio		**1 : 53.02**	

Class imbalance challenge. A critical characteristic of this dataset is its severe class imbalance. As summarized in Table 2, the target class (`route_changed` = 1) represents only 1.85% of the training data, whereas stable routes (`route_changed` = 0) account for 98.15%. This yields an imbalance ratio of approximately 1:53, which poses a significant challenge for standard classifiers that tend to bias predictions towards the majority class.

3.2. Feature engineering (Phase 1)

Phase 1 in Figure 1 converts raw traceroute rows into feature vectors that capture per-trace statistics, path-level temporal dynamics, and aggregate context for sources and destinations.

3.2.1. Per-trace statistics

Each traceroute row contains a list of round-trip time samples, denoted by $\mathbf{r}_i = (r_{i,1}, \ldots, r_{i,L_i})$. We first compress this list into stable summary statistics. The mean and

variance are defined as

$$\mu_i = \frac{1}{L_i} \sum_{\ell=1}^{L_i} r_{i,\ell}, \tag{1}$$

$$\sigma_i^2 = \frac{1}{L_i} \sum_{\ell=1}^{L_i} \big(r_{i,\ell} - \mu_i\big)^2, \tag{2}$$

so that μ_i reflects the typical latency of that traceroute and σ_i quantifies its variability. In addition, we compute empirical percentiles (e.g., 25th, 50th, 75th, and 90th), the minimum and maximum latency, the length L_i, and the interquartile range, which together characterize different regions of the latency distribution.

Reliability features are derived from the probe counters. From the number of probes sent and the number of successful replies at the last hop, we obtain the success rate and the corresponding loss rate. These variables summarize how often the traceroute reaches the destination and how frequently probes are dropped. All these statistics, together with the original scalar attributes, form the first block of the feature vector.

3.2.2. Path-level temporal context

Traceroute rows that share the same source and destination form a time-ordered sequence when sorted by their measurement time. For each path (s_i, d_i) we exploit this order to derive temporal features that describe how recent measurements differ from previous ones.

Consider a scalar feature x_i computed from a single row, such as the mean latency or the success rate. Within a fixed path, we sort rows by increasing time and, for each row i that is not the first, identify its immediate predecessor j with the same source and destination. We then define the absolute and relative change as

$$\Delta x_i = x_i - x_j, \tag{3}$$

$$\rho_i = \frac{x_i}{x_j + \varepsilon}, \tag{4}$$

where ε is a small constant to avoid division by zero. The difference Δx_i captures the direction and magnitude of the most recent change, while the ratio ρ_i measures this change relative to the previous level and is clipped to a fixed interval to limit the effect of outliers. The time interval between the two measurements is

$$\Delta t_i = t_i - t_j,$$

which indicates how long the path remained unobserved before the current traceroute. For the first observation of a path, these temporal features are set to zero.

3.2.3. Rolling and aggregate context features

In addition to local differences between consecutive rows, TRACE includes features that capture short-term trends and how typical each observation is for its source and destination.

For each path, we compute rolling means and standard deviations of selected features (such as mean latency and success rate) over windows of three and seven observations. These rolling statistics highlight whether latency or loss is gradually increasing, decreasing, or stable over recent traceroutes, without requiring the reader to follow explicit formulae.

At a coarser level, we aggregate information per source and per destination. For each source, we count how many traceroutes it originates and how many distinct destinations it reaches; the same is done per destination. We then compute empirical averages and standard deviations of key features for all rows that share a given source or destination and derive standardized deviations. For example, for a feature x_i and the mean and standard deviation associated with the source of row i, we define

$$z_i^{x,\mathrm{src}} = \frac{x_i - \mu_{s_i}^x}{\sigma_{s_i}^x + \varepsilon}.$$

Analogous quantities are computed with respect to destinations. These aggregate features indicate whether a particular measurement is unusually slow, variable, or unreliable for that source or destination, which can be a strong signal of route changes.

3.3. Base learners and baseline models (Phases 2 and 4)

After feature engineering, each row i is represented by a vector $\mathbf{u}_i \in \mathbb{R}^d$, obtained by concatenating per-trace statistics, temporal context, and aggregate features. Phase 2 in Figure 1 trains three gradient-boosted tree classifiers on this representation, implemented with LightGBM, CatBoost, and XGBoost. These models act as base learners and produce probability scores for route change.

To obtain reliable meta-features for stacking, we use stratified five-fold cross-validation. The indices $\{1, \ldots, N\}$ are partitioned into five folds; for each fold and each base learner, we train on four folds and compute predictions on the held-out fold. By concatenating the predictions across all folds, we obtain three vectors of out-of-fold probabilities $p_i^{(m)}$, one per base learner, which are used as inputs to the meta-model. For the test set, we train one model per fold and average their predictions.

Phase 4 in Figure 1 trains conventional baseline classifiers directly on the engineered feature vectors $\mathbf{u}_i$. Specifically, we consider logistic regression, Random Forest, and k-nearest neighbours, each tuned with a small grid over its main hyperparameters. These baselines share the same training–validation splits as the base learners and are later evaluated under the same F_1-oriented threshold calibration procedure as the stacked ensemble.

3.4. Stacked ensemble and meta-features (Phase 3)

Phase 3 combines the strengths of the base learners through stacking. For each training row i, we collect the out-of-fold predictions into the vector

$$\mathbf{p}_i = \left[p_i^{(1)}, p_i^{(2)}, p_i^{(3)}\right],$$

which summarizes how the three base models assess the probability of a route change. From this vector, we compute a small set of summary statistics, such as the mean, standard deviation, median, and pairwise differences. These statistics, denoted by $\phi(\mathbf{p}_i)$ in Figure 1, measure how much the base learners agree or disagree for each example.

In addition, we select a subset of the original features, denoted by $\psi(\mathbf{u}_i)$, that were empirically found to be particularly informative, including mean latency, variability, high percentiles, success rate, and a few temporal and rolling features. Finally, we construct simple interaction features, denoted by $\omega(\mathbf{p}_i, \mathbf{u}_i)$, that capture nonlinear relationships between predictions and original variables, for example products of base predictions with the success rate and squared prediction terms.

The complete meta-feature vector for row i is then

$$\mathbf{h}_i = \big[\, \mathbf{p}_i,\, \phi(\mathbf{p}_i),\, \psi(\mathbf{u}_i),\, \omega(\mathbf{p}_i, \mathbf{u}_i) \,\big],$$

as illustrated in the dashed block of Figure 1. The meta-model g is a LightGBM classifier trained on these vectors to produce final probability scores $\hat{p}_i = g(\mathbf{h}_i)$, which correspond to the rightmost block in Phase 3.

3.5. Training, hyperparameter optimization, and threshold calibration

The meta-model depends on a set of hyperparameters that control the number of boosting iterations, learning rate, tree depth, and regularization strength. We use stratified five-fold cross-validation on the meta-features and Hyperopt's Tree-structured Parzen Estimator to explore this space and select the configuration that maximizes the F_1 score. Once the best configuration is identified, a final copy of the meta-model is trained on all training rows and applied to the meta-features of the test set.

Because route-change events are rare, the default classification threshold of 0.5 is not appropriate. Instead, after training, we scan a fine grid of candidate thresholds on the training probabilities $\hat{p}_i$ and, for each value, compute precision, recall, and the resulting F_1 score. The threshold that maximizes F_1 is selected and then applied to the test probabilities for both the stacked ensemble and the baselines. This calibration step, explicitly represented at the bottom of Figure 1, ensures a balanced operating point between missed route changes and false alarms under severe class imbalance.

4. Results and Discussion

All experiments were executed on a headless Ubuntu 20.04.6 LTS virtual machine running Linux kernel 5.4, provisioned on an OpenStack/KVM cluster and equipped with 64 vCPUs from an AMD EPYC 7532 processor, 256 GiB of RAM, a 1 TB SSD-backed volume, and an NVIDIA Quadro RTX 6000 GPU.

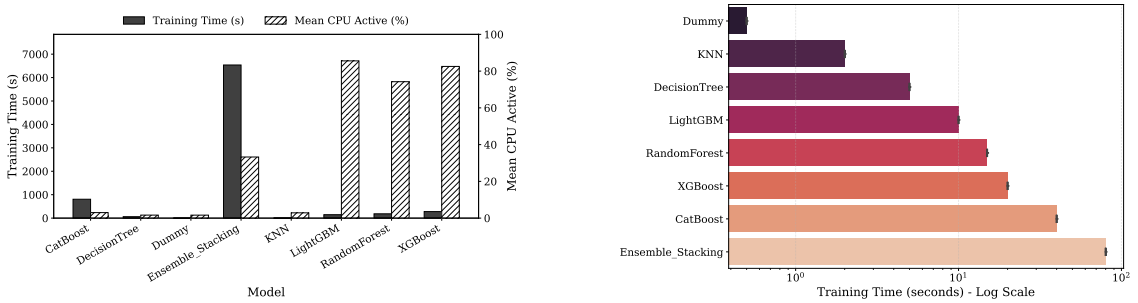
Hyperparameter search for the meta model. Initially, we conducted experiments to determine the best hyperparameters. Table 3 summarizes the search space used for the meta model and reports the configuration selected by the Tree-structured Parzen Estimator. The search favored shallow trees with a small number of leaves and a relatively high learning rate, which indicates that the meta-learner captures the residual structure left by the base classifiers without requiring deep tree ensembles or very aggressive regularization.

Computational footprint of individual models. Figure 2 summarizes the computational profile of all classifiers. The left panel contrasts training duration with mean CPU utilization, while the right panel reports the absolute training time on a logarithmic scale. As expected, the Stacking Ensemble exhibits the longest training duration, a direct

Table 3. Hyperparameter search space and TPE-selected optimal values for LightGBM.

Hyperparameter	Search Range	Step	Optimal Value
Number of Estimators	$[100, 5000]$	100	**200**
Learning Rate	$[0.001, 0.05]$	Cont.	**0.0409**
Maximum Tree Leaves	$[10, 80]$	2	**10**
Maximum Tree Depth	$[3, 10]$	1	**3**
Feature Fraction	$[0.6, 0.9]$	Cont.	**0.6687**
Bagging Fraction	$[0.6, 0.9]$	Cont.	**0.7547**
L1 Regularization	$[0.1, 10.0]$	Cont.	**0.5019**
L2 Regularization	$[0.1, 10.0]$	Cont.	**0.1471**

consequence of its sequential training pipeline, which necessitates training base learners prior to the meta-model. However, its mean CPU active percentage remains comparable to, or even lower than, standalone gradient boosting frameworks like XGBoost and LightGBM. This suggests that the increased duration is driven by the serial execution of tasks rather than a strictly higher instantaneous computational intensity. In absolute terms, the ensemble needs about ninety seconds to complete one training cycle, which is higher than for simpler models such as KNN and Decision Trees, yet still well within the daily or weekly window typically available for offline network management.



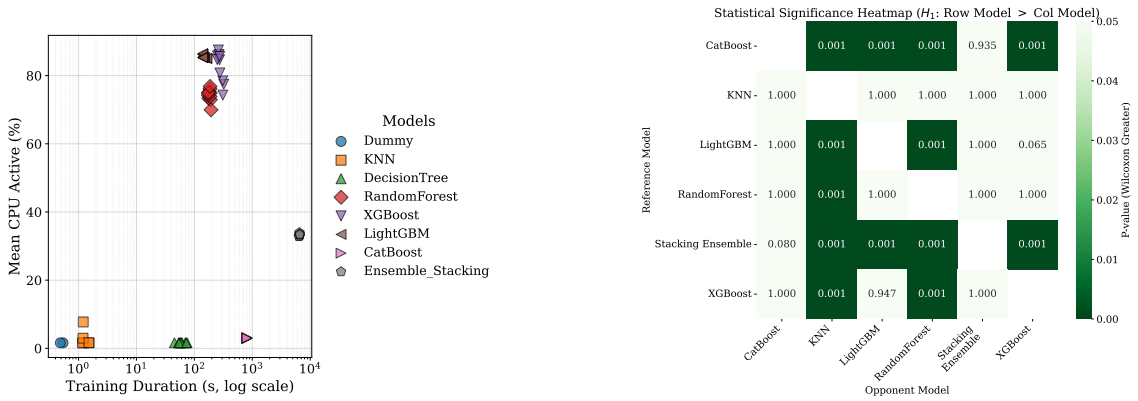
(a) CPU utilization vs. training duration.

(b) Training time comparison (log scale).

Figure 2. Computational footprint of all evaluated models.

Operational implications of training cost. From an operational perspective, this resource footprint is manageable within the proposed offline retraining schedule. While simple models like KNN and Decision Trees offer negligible training costs, they fail to achieve the statistical robustness required for this domain. The Stacking Ensemble represents a calculated trade-off: it demands a longer time window to complete, approximately one order of magnitude higher than single boosters, but does not impose a prohibitive load on the underlying hardware, allowing other monitoring processes to coexist on the same server without significant degradation.

Figure 3 relates computational cost to statistical benefit. Panel (a) further explores the relationship between training duration and processor saturation on a logarithmic time scale, revealing three operational clusters: lightweight baselines, high-performance boosters, and the Stacking Ensemble as the high-duration outlier that still fits the trend of increasing complexity required to capture non-linear route change patterns. Panel (b) complements this view by showing that, according to the Wilcoxon Signed Rank Test,



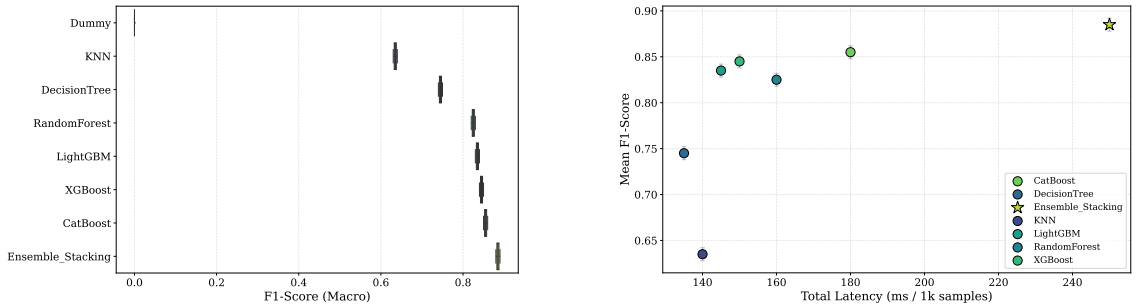
(a) Training duration versus CPU utilization.

(b) Wilcoxon pairwise significance heatmap.

Figure 3. Relationship between computational cost and statistical benefit across models.

the Stacking Ensemble statistically outperforms almost all baselines, with very small p values in most pairwise comparisons, and achieves statistical parity with CatBoost at a conservative significance level of $\alpha = 0.05$, thereby justifying its higher training time with consistent gains in predictive performance.

Time overhead versus scalability. The positioning of the Stacking Ensemble in the high-duration region confirms that the primary cost of this architecture is temporal latency rather than resource exhaustion. Since the application scenario prioritizes predictive stability and F1 score over instantaneous model updates, this delay is acceptable. The analysis demonstrates that the system scales predictably, and the overhead introduced by the meta-learning layer yields the necessary performance gains discussed in the previous section without pushing the system into an unstable computational regime.



(a) Distribution of F1 scores across ten experimental rounds.

(b) Trade-off between mean F1 and inference latency.

Figure 4. Predictive performance and latency trade-off across all models.

Figure 4 summarizes both predictive performance and latency considerations. Panel (a) shows that the Stacking Ensemble attains a median F1 score that is comparable to or higher than that of CatBoost while exhibiting a more compact interquartile range, indicating reduced predictive variance and more stable performance across runs.

Panel (b) depicts the trade-off between mean F1 and inference latency, where the ensemble lies close to the Pareto frontier by combining high predictive quality with inference times that remain suitable for near real time monitoring when operators process batches of one thousand samples.

Table 4. Overall performance comparison across all models.

Model	F1 Score	Accuracy
CatBoost	0.855	0.865
DecisionTree	0.745	0.755
Dummy	0	0.5
KNN	0.635	0.655
LightGBM	0.835	0.845
RandomForest	0.825	0.835
XGBoost	0.845	0.855
TRACE	**0.869**	**0.895**

Table 4 shows how well all classifiers did on the test set. Among the single models, gradient-boosted trees (LightGBM, XGBoost, and CatBoost) had the highest F1 scores. They did better than traditional models like random forest, decision tree, and KNN. TRACE, our ensemble model, improves these results by combining their predictions. It has the best F1 score and accuracy while keeping good precision and recall, even with uneven class distribution.

From a networking perspective, the high F1 score on this imbalanced dataset indicates that latency-derived features alone carry sufficient signal to flag events that would traditionally require BGP feed inspection, supporting the viability of lightweight, control-plane-independent monitoring.

5. Concluding Remarks

TRACE addresses the problem of detecting Internet route changes using only traceroute latency measurements without relying on control plane information. By combining temporal and aggregate feature engineering with a stacked ensemble and F1-driven threshold calibration, TRACE achieves a strong detection performance under a severe class imbalance while remaining deployable on large public datasets. This result supports practical data-plane-only monitoring workflows that can be retrained offline without privileged network access. The limitations include the reliance on labelled events, the sensitivity of the engineered features to measurement noise and dataset bias, and the current lack of explicit handling of load-balanced or multipath routes, which may appear as spurious route changes. Future work will explore representation learning with deep models and one-class or self-supervised approaches to reduce label dependence and improve generalization across heterogeneous paths and topologies.

Artifact Availability

In adherence to Open Science principles, the source code and scripts used in TRACE are publicly available at: `https://github.com/romoreira/RNP-DataChallenge-2025`. This repository includes our first-place entry for the CT-Mon Data Challenge 2025.

Acknowledgments

We acknowledge the financial support of the FAPESP MCTIC/CGI Research project 2018/23097-3 - SFI2 - Slicing Future Internet Infrastructures. We acknowledge the financial support of the FAPESP MCTIC/CGI Research project 2018/23097-3 and FAPEMIG (Grant APQ00923-24) and Centro ALGORITMI, funded by Fundação para a Ciência e Tecnologia (FCT) within the RD Units Project Scope 2020-2023 (UIDB/00319/2020) for partially support this work.

References

Al-Qudah, Z., Jomhawy, I., Alsarayreh, M., and Rabinovich, M. (2020). On the stability and diversity of Internet routes in the MPLS era. *Performance Evaluation*, 138:102084.

Alaraj, A., Bock, K., Levin, D., and Wustrow, E. (2023). A global measurement of routing loops on the internet. In Brunstrom, A., Flores, M., and Fiore, M., editors, *Passive and Active Measurement*, pages 373–399, Cham. Springer Nature Switzerland.

Alberti, A. M., Pivoto, D. G., Rezende, T. T., Leal, A. V., Both, C. B., Facina, M. S., Moreira, R., and de Oliveira Silva, F. (2024). Disruptive 6g architecture: Software-centric, ai-driven, and digital market-based mobile networks. *Computer Networks*, 252:110682.

Bhaskar, A. and Pearce, P. (2024). Understanding routing-induced censorship changes globally. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 437–451, New York, NY, USA. Association for Computing Machinery.

Fazzion, E., Teixeira, G., Veitch, D., Diot, C., Teixeira, R., and Cunha, I. (2025). RemapRoute: Local Remapping of Internet Path Changes. In *Proceedings of the 2025 ACM Internet Measurement Conference*, IMC '25, page 185–191, New York, NY, USA. Association for Computing Machinery.

Giotsas, V., Koch, T., Fazzion, E., Cunha, I., Calder, M., Madhyastha, H. V., and Katz-Bassett, E. (2020). Reduce, Reuse, Recycle: Repurposing Existing Measurements to Identify Stale Traceroutes. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 247–265, New York, NY, USA. Association for Computing Machinery.

Islam, S., Welzl, M., Hapnes, E., and Feng, B. (2024). Using the IPv6 Flow Label for Path Consistency: A Large-Scale Measurement Study. In *ICC 2024 - IEEE International Conference on Communications*, pages 3022–3027.

Katsaros, K., Mavromatis, I., Antonakoglou, K., Ghosh, S., Kaleshi, D., Mahmoodi, T., Asgari, H., Karousos, A., Tavakkolnia, I., Safi, H., Hass, H., Vrontos, C., Emami, A., Marcelo Parra-Ullauri, J., Moazzeni, S., and Simeonidou, D. (2024). Ai-native multi-access future networks—the reason architecture. *IEEE Access*, 12:178586–178622.

Kirci, E. C., Torsiello, V., and Vanbever, L. (2024). What is the next hop to more granular routing models? In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, page 343–351, New York, NY, USA. Association for Computing Machinery.

Li, J., Giotsas, V., Wang, Y., and Zhou, S. (2022). BGP-Multipath Routing in the Internet. *IEEE Transactions on Network and Service Management*, 19(3):2812–2826.

Li, Y., Huang, Y., Liu, R. D., and Sun, B. S. (2025). Is Reverse Traceroute Reliable? In *Proceedings of the 9th Asia-Pacific Workshop on Networking*, APNET '25, page 284–286, New York, NY, USA. Association for Computing Machinery.

Lin, S., Zhou, Y., Zhang, X., Arnold, T., Govindan, R., and Yang, X. (2025). Tiered Cloud Routing: Methodology, Latency, and Improvement. *Proc. ACM Meas. Anal. Comput. Syst.*, 9(1).

Measurement Lab (2025). M-lab open data. `https://www.measurementlab.net/data/`. Accessed: 2025-12-06.

Moreira, R., Rosa, P. F., Aguiar, R. L. A., and de Oliveira Silva, F. (2021). NASOR: A network slicing approach for multiple Autonomous Systems. *Computer Communications*, 179:131–144.

Paxson, V. (1996). End-to-end routing behavior in the Internet. *SIGCOMM Comput. Commun. Rev.*, 26(4):25–38.

Sagatov, E. S., Chernysh, D. P., Mayhoub, S., and Sukhov, A. M. (2025). Detection of anomalous network behavior based on one-way delay measurements. *Discover Internet of Things*, 5(1):129.

Schmid, R., Schneider, T., Fragkouli, G., and Vanbever, L. (2025). Transient Forwarding Anomalies and How to Find Them. *Proc. ACM Netw.*, 3(CoNEXT2).

Shapira, T. and Shavitt, Y. (2022). AP2Vec: An Unsupervised Approach for BGP Hijacking Detection. *IEEE Transactions on Network and Service Management*, 19(3):2255–2268.

Syamkumar, M., Gullapalli, Y., Tang, W., Barford, P., and Sommers, J. (2022). Bigben: Telemetry processing for internet-wide event monitoring. *IEEE Transactions on Network and Service Management*, 19(3):2625–2638.

Tian, Z., Su, S., Shi, W., Du, X., Guizani, M., and Yu, X. (2019). A data-driven method for future internet route decision modeling. *Future Generation Computer Systems*, 95:212–220.

Vermeulen, K., Gurmericliler, E., Cunha, I., Choffnes, D., and Katz-Bassett, E. (2022). Internet scale reverse traceroute. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 694–715, New York, NY, USA. Association for Computing Machinery.

Wassermann, S., Casas, P., Cuvelier, T., and Donnet, B. (2017). Netperftrace: Predicting internet path dynamics and performance with machine learning. In *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, Big-DAMA '17, page 31–36, New York, NY, USA. Association for Computing Machinery.

Yang, S., Tan, C., Madsen, D. Ø., Xiang, H., Li, Y., Khan, I., and Choi, B. J. (2022). Comparative analysis of routing schemes based on machine learning. *Mobile Information Systems*, 2022(1):4560072.