CATO: Multi-Objective Optimization for Efficient ML Traffic Analysis Pipelines

Abstract

CATO leverages recent advances in multi-objective Bayesian optimization to efficiently identify Pareto-optimal configurations, and automatically compiles end-to-end optimized serving pipelines that can be deployed in real networks.

Motivation

- Traditional ML solutions prioritize predictive performance but often ignore system-level efficiency.
- In network environments, serving pipeline delays can lead to packet loss and degraded predictions.
- Balancing predictive performance and system cost is critical.

CATO: High-Level Workflow

- Formalizes ML model development as a multi-objective optimization problem.
- Uses Bayesian optimization with a realistic profiler.
- Jointly searches feature sets and capture depth.
- Measures real-world end-to-end system costs and model performance.

Why ML for Network Traffic?

- Increasing traffic volume.
- Rise in encryption limits rule-based methods.
- Rule-based heuristics fail in dynamic environments.

Current Limitations of ML-Based Traffic Analysis

- ML models are often evaluated offline, missing runtime complexities.
- Pipeline inefficiencies lead to high latency and packet loss.
- Real deployments require joint optimization of model and system performance.

Traffic Analysis Pipeline (1/2)

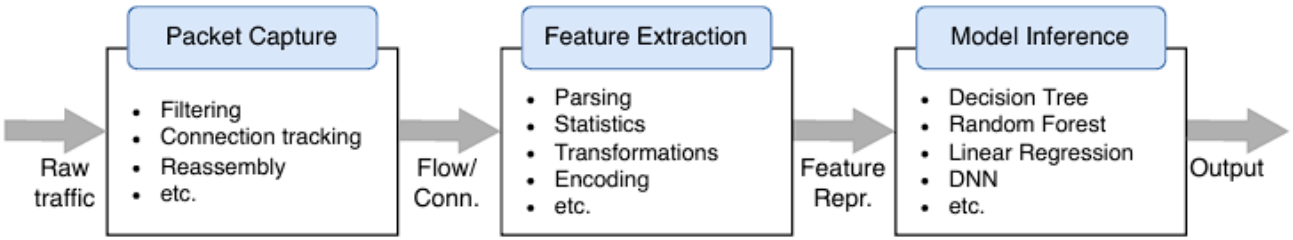


Figure 1: A typical serving pipeline for ML-based traffic analysis. Usability of a model hinges on both its predictive accuracy and the systems performance of the entire pipeline.

Traffic Analysis Pipeline (2/2)

Typical Stages:

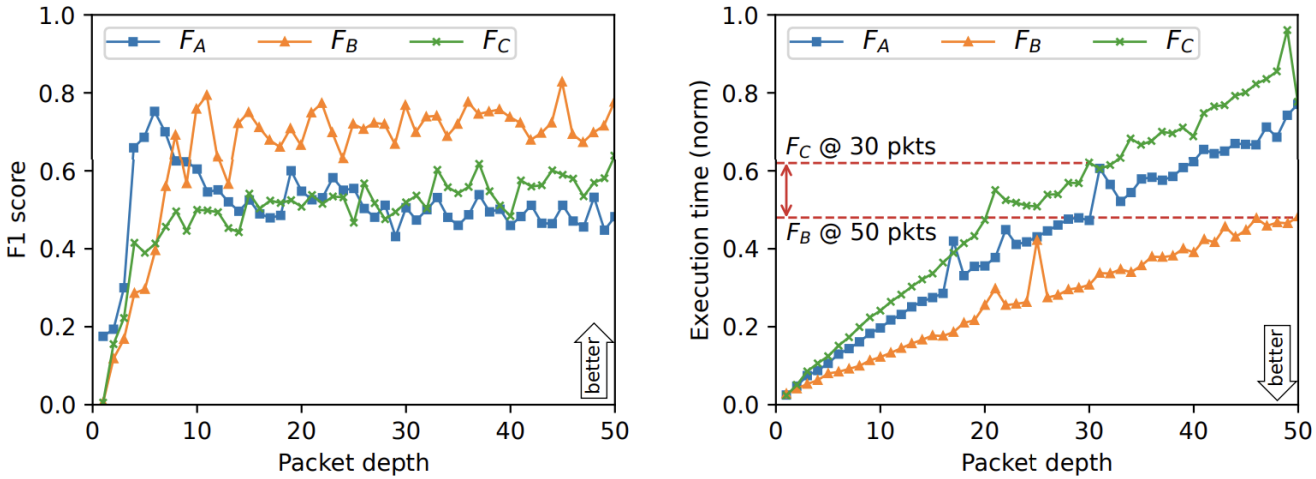
1. Packet Capture
2. Feature Extraction
3. Model Inference

Current Optimization Methods:

- Use of summary statistics and packet lengths.
- Lightweight models or early predictions (compromising accuracy).

Issue: No ideal capture depth or feature set without domain knowledge or trial-and-error.

Challenges in End-to-End Optimization (1/2)



(a) Packet Depth vs. F1 Score. The best feature sets differ at varying packet depths.

(b) Packet Depth vs. Exec. Time. It can be cheaper to extract low-cost features at greater depths.

Figure: Optimization Time and Search Space Complexity

Challenges in End-to-End Optimization (2/2)

- Goal is to create a pipeline that is pareto-optimal across both its execution and f1 score.
- Therefore we need a multidimensional and multi objective search space that extends beyond identifying features which results in accurate models
- Measuring all pipelines: 6 candidate features took 5 days; 25 would take 7000 years.

CATO Optimization

Goal: Construct pipelines that jointly minimize end-to-end system cost and maximize predictive performance.

CATO combines:

- Multi-objective Bayesian Optimization (BO)
- Pipeline generator
- Profiler for feature representation evaluation

Outcome: Servable pipelines suitable for deployment in real-world networks.

Problem Description

Symbol	Description
$\mathcal{F}$	Set of candidate network flow features
N	Maximum connection depth
$\mathcal{P}(\mathcal{F})$	Power set of $\mathcal{F}$
$\mathbb{X}$	Search space defined as $\mathbb{X} = \mathcal{P}(\mathcal{F}) \times N$
F	Set of features in a feature representation
n	Connection depth from which F is extracted
x	Feature representation $x = (F, n)$
$\texttt{cost}(x)$	Systems cost objective function
$\texttt{perf}(x)$	Predictive performance objective function
Γ	Set of Pareto-optimal solutions

Table 1: Summary of Variables

Problem Description

Input:

- Candidate network flow features
- Maximum connection depth (upper bound)

Formulation:

- Multi-objective optimization over search space $\mathcal{X}$
- Identify Pareto front Γ — optimal (F, n) combinations
- Γ: No further improvement in cost or performance is possible without compromising the other

Benefit: Flexibility — New tradeoffs can be selected without rerunning optimization.

CATO System Overview

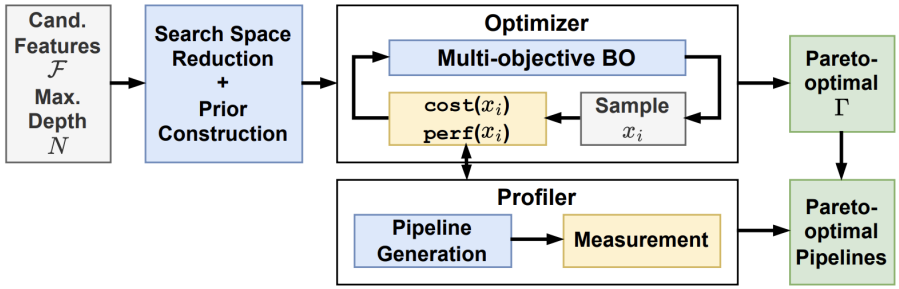


Figure 3: CATO combines a multi-objective BO-based Optimizer and a realistic pipeline Profiler to construct and validate efficient ML-based traffic analysis serving pipelines.

Optimizer:

- Performs BO-guided search over feature representations and depth
- Periodically queries Profiler to measure cost(x) and perf(x)

Profiler:

- Accepts sampled representations
- Compiles and runs end-to-end pipeline to return measurements

CATO Optimizer: Motivation and BO Fundamentals

Motivation: Measuring cost and performance for each configuration is expensive.

Bayesian Optimization (BO):

- Global optimization for black-box functions
- Effective for discontinuous, non-differentiable objectives
- Used in hyperparameter tuning, compiler optimization, robotics

BO Process:

- Build surrogate model (random forest)
- Predict objective values for candidate points
- Select next sample via Expected Improvement
- Update model and iterate

BO Formulation in CATO

Search space: $|F| + 1$ dimensions (features + connection depth)
Objectives:

- Minimize cost(F, n)
- Maximize perf(F, n)

Dimensionality Reduction:

- Discard features with mutual information $= 0$

Prior Injection: Encourages CATO to more frequently explore regions of the search space that include features with higher predictive power.

CATO Profiler: Purpose and Workflow

Role:

- Guides the Optimizer
- Validates system cost and model accuracy

Workflow:

- Compiles binaries for packet capture + feature extraction
- Trains model for sampled configuration
- Runs full pipeline to measure:
  - End-to-end system cost
  - Predictive performance

Profiler: Why Direct Measurement?

Why measure cost and performance directly?

- Heuristics can't capture complexity of real-world packet capture
- Feature interactions may change performance unexpectedly

Pipeline Generation:

- Uses conditional compilation instead of runtime branching
- Only includes operations up to connection depth n

Deployment:

- Supports simulated traffic or in-network execution

Implementation of CATO (1/3)

Bayesian Optimization Setup:

- Framework: **HyperMapper** — used for multi-objective, mixed-variable optimization.
- Extended with **πBO** — to inject domain-specific priors from CATO.
- Surrogate model: **Random Forest** — effective for non-linear, discontinuous objectives.
- Initialization: 3 random samples to explore design space.

Why Random Forest?

- Empirically outperforms Gaussian processes in high-dimensional, non-smooth spaces.

Implementation of CATO (2/3)

Priors Used in Optimization:

Feature Selection Prior:

- Encourages informative features while maintaining diversity.

$$P(f \in F \mid x \in \Gamma) = (1 - \delta)\frac{I(f)}{I_{\max}} + \frac{\delta}{2}, \quad \delta = 0.4$$

Packet Capture Depth Prior:

- Penalizes deeper packet captures to reduce latency and system overhead.

$$P(n) \sim \text{Beta}(\alpha = 1, \beta = 2)$$

Effect:

- Priors guide the search towards low-cost, high-performing configurations.

Implementation of CATO (3/3)

Pipeline Compilation and Execution:

- Built on modified Retina engine (Rust).
- Automatically compiles serving pipelines per configuration.
- Use-case agnostic features allow generalization across traffic types.

Evaluation:

- Each pipeline is tested for system cost and model performance.
- Real traffic input or simulation used to ensure deployment fidelity.

Model Training and Evaluation

Models Used:

- Decision Trees, Random Forests (scikit-learn)
- Deep Neural Networks
- Grid search with nested 5-fold cross-validation

Evaluation Metrics:

- System cost: Latency, throughput, execution time
- Model performance: F1 Score, RMSE

Datasets:

- Web App Classification (live traffic)
- IoT Recognition (Sivanathan et al.)
- Video Delay Inference (Bronzino et al.)

Conclusion

- CATO automates the end-to-end optimization of ML-based traffic analysis pipelines.
- Efficiently balances predictive performance with system cost using multi-objective Bayesian optimization.
- Profiler ensures real-world deployment feasibility by compiling and testing serving pipelines.
- Applicable to various domains including web traffic classification, IoT device detection, and video performance analysis.
- Future work: scaling to more complex models, extending to additional network tasks, and optimizing under evolving constraints.

Thank You!

Thank You!