Building Adaptive Networks with Machine Learning

Tushar Swamy

Stanford University
Networks need data-driven decisions: 

→ **Machine Learning**
Networks can benefit from machine learning

- Security
  - Anomaly Detection
  - Traffic Classification
- Control
  - Congestion Control
  - Active Queue Management
- Analytics
  - Queue Reconstruction
  - QoS Correlation
A Taurus network introduces ML for management

**Software Defined Network**

- **Control Plane**
  - Policy Creation (Flow Rules)

- **Data Plane**
  - Packet Forwarding (Match Action)

- **Packets In**
- **Packet Digest**
- **Flow Rule**
- **Packets Out**

**Software Defined Network with Taurus**

- **Control Plane**
  - Policy Creation (Flow Rules + ML Training)

- **Data Plane**
  - Packet Forwarding (Match Action) + Decision Making (ML Inference)

- **Packets In**
- **Packet Digest**
- **Flow Rule**
- **ML model weights**
- **Packets Out**
The Taurus switch pipeline

Packet Parsing  Preprocessing  ML Inference  Postprocessing  Scheduling

Packet Parser  Match-Action Tables  Map Reduce Unit  Match-Action Tables  Traffic Manager

Packet Parsing  Preprocessing  ML Inference  Postprocessing  Scheduling

Packet Parser  Match-Action Tables  Map Reduce Unit  Match-Action Tables  Traffic Manager
Taurus achieves line-rate ML inference

- Robustness and performance of the network is determined by quality and speed of reaction
- ML inference should happen per-packet in the dataplane
- Taurus enables line-rate, per-packet ML inference in the dataplane

How do we program Taurus?
Homunculus: a framework for dataplane model generation

• Provide **high-level directives** to express user intent

• Take **network and resource constraints** into account when building ML models

• **Generate binaries** for different dataplane devices with optimized ML models
Homunculus architecture

Input: Dataset, Constraints, Target

- HOMUNCULUS
  - ML Pipeline Generator
    - Alchemy Frontend
    - Optimization Core
    - Platform-Specific Backend

- Data Plane Binary

- Alchemy Frontend
- Topology Constraints
- Targets Library
- Model Schedules
- Candidate Models Selection
- Model Composition
- Design Space Creation
- Bayesian Optimization
- Model Training & Testing
- Backend Generation
- Optimization
- Feasibility Testing
- Final Model Selection & Code Generation
- Data Plane Binary
Homunculus overview

User specifies a dataset, environment configuration alongside a network DSL (P4)

ML models are generated through Bayesian optimization (HyperMapper) and ML frameworks (Keras) to meet optimization requirements

Platform-specific backends (Spatial, Tungsten) test resource usage and performance

P4 and Spatial are installed on a switch (Taurus)

Homunculus provides network operators a high-level compiler to program ML switches
Homunculus frontend: Alchemy

Input: Dataset, Constraints, Target

Homunculus
ML Pipeline Generator

Alchemy Frontend
Optimization Core
Platform-Specific Backend

Data Plane Binary
Homunculus frontend: Alchemy

Alchemy is a Python library

```python
import homunculus
from homunculus.alchemy import DataLoader, Model, Platforms
import ad_loader

@DataLoader  # training data loader definition
def wrapper_func():
    tnx, tny = ad_loader.load_from_file("train_ad.csv")
    tsx, tsy = ad_loader.load_from_file("test_ad.csv")
    return {
        "data": {"train": tnx, "test": tsx },
        "labels": {"train": tny, "test": tsy }
    }

# Specify the model of choice
model_spec = Model(
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func
)

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1, # GPkt/s
        "latency": 500,  # ns
        "resources": { "rows": 16, "cols": 16 }
    }
)

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)
```
# Load data

```python
import homunculus
from homunculus.alchemy import DataLoader, Model, Platforms
import ad_loader

@DataLoader  # training data loader definition
def wrapper_func():
    txn, tyn = ad_loader.load_from_file("train_ad.csv")
    txs, tss = ad_loader.load_from_file("test_ad.csv")
    return {
        "data": {"train": txn, "test": txs },
        "labels": {"train": tyn, "test": tss }
    }

# Specify the model of choice
model_spec = Model({
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func
})

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1,  # GPkt/s
        "latency": 500  # ns
    },
    "resources": {"rows": 16, "cols": 16 }
)

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)
```
import homunculus
from homunculus.alchemy import DataLoader, Model, Platforms
import ad_loader

@DataLoader # training data loader definition
def wrapper_func():
    tnx, tny = ad_loader.load_from_file("train_ad.csv")
    tsx, tsy = ad_loader.load_from_file("test_ad.csv")
    return {
        "data": {"train": tnx, "test": tsx },
        "labels": {"train": tnx, "test": tsy }
    }

# Specify the model of choice
model_spec = Model({
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func })

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1, # GPkt/s
        "latency": 500 }, # ns
        "resources": { "rows": 16, "cols": 16 })

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)
import homunculus
from homunculus.alchemy import DataLoader, Model, Platforms
import ad_loader

@DataLoader # training data loader definition
def wrapper_func():
    tnx, tny = ad_loader.load_from_file("train_ad.csv")
    tsx, tsy = ad_loader.load_from_file("test_ad.csv")
    return {
        "data": {"train": tnx, "test": tsx },
        "labels": {"train": tny, "test": tsy }}

# Specify the model of choice
model_spec = Model({
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func })

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1, # GPkt/s
        "latency": 500 }, # ns
        "resources": { "rows": 16, "cols": 16 })

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)

Constrain the platform
```python
import homunculus
from homunculus.alchemy import DataLoader, Model, Platforms

import ad_loader

@DataLoader  # training data loader definition
def wrapper_func():
    tnx, tny = ad_loader.load_from_file("train_ad.csv")
    tsx, tsy = ad_loader.load_from_file("test_ad.csv")
    return {
        "data": {"train": tnx, "test": tsx },
        "labels": {"train": tny, "test": tsy }
    }

# Specify the model of choice
model_spec = Model(
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func })

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1,  # GPkt/s
        "latency": 500 },  # ns
    "resources": { "rows": 16, "cols": 16 })

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)
```

Generate binary
Homunculus Optimization Core

Input: Dataset, Constraints, Target

HOMUNCULUS
ML Pipeline Generator

Alchemy Frontend
Optimization Core
Platform-Specific Backend

Data Plane Binary
Bayesian Optimization (HyperMapper)

Batch of hyperparameters

Homunculus

Train models

ML Model

Test models

Feasibility check

Optimization Objectives + Feasibility Constraints

Batched results
Homunculus Backend

Input: Dataset, Constraints, Target

Homunculus
ML Pipeline Generator

Alchemy Frontend

Optimization Core

Platform-Specific Backend

Data Plane Binary
Homunculus results

- F1 Score improvements in anomaly detection (AD), traffic classification (TC), botnet detection (BD)

<table>
<thead>
<tr>
<th>App</th>
<th>Features</th>
<th># NN Param</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base-AD</td>
<td>7</td>
<td>203</td>
<td>71.10</td>
</tr>
<tr>
<td>Hom-AD</td>
<td>7</td>
<td>254</td>
<td>83.10</td>
</tr>
<tr>
<td>Base-TC</td>
<td>7</td>
<td>275</td>
<td>61.04</td>
</tr>
<tr>
<td>Hom-TC</td>
<td>7</td>
<td>370</td>
<td>68.75</td>
</tr>
<tr>
<td>Base-BD</td>
<td>30</td>
<td>662</td>
<td>77.0</td>
</tr>
<tr>
<td>Hom-BD</td>
<td>30</td>
<td>501</td>
<td>79.8</td>
</tr>
</tbody>
</table>
Homunculus makes it easy to generate high-quality models

- Provides a **high-level interface** for users to express intent
- Uses **network and resource constraints** to traverse AutoML search space
- **Generate binaries** for different dataplane devices with optimized ML models

How do we supply data to Homunculus?
Data Pipeline supplies high quality data to Homunculus.
Questions?

Tushar Swamy
tswamy@stanford.edu

Taurus: https://dl.acm.org/doi/10.1145/3503222.3507726

Enable ML inference in the dataplane

Run line-rate, per-packet operation

Homunculus: https://arxiv.org/pdf/2206.05592

Allows automated construction of ML models

More accurate than hand-tuned models

Try it out! https://gitlab.com/dataplane-ai/taurus