In-network inference with P4:

from stateless to hybrid approaches

P4 Developer Days

21 January 2026

Aristide Tanyi-Jong Akem

a.t-j.akem@soton.ac.uk

Lecturer, University of Southampton

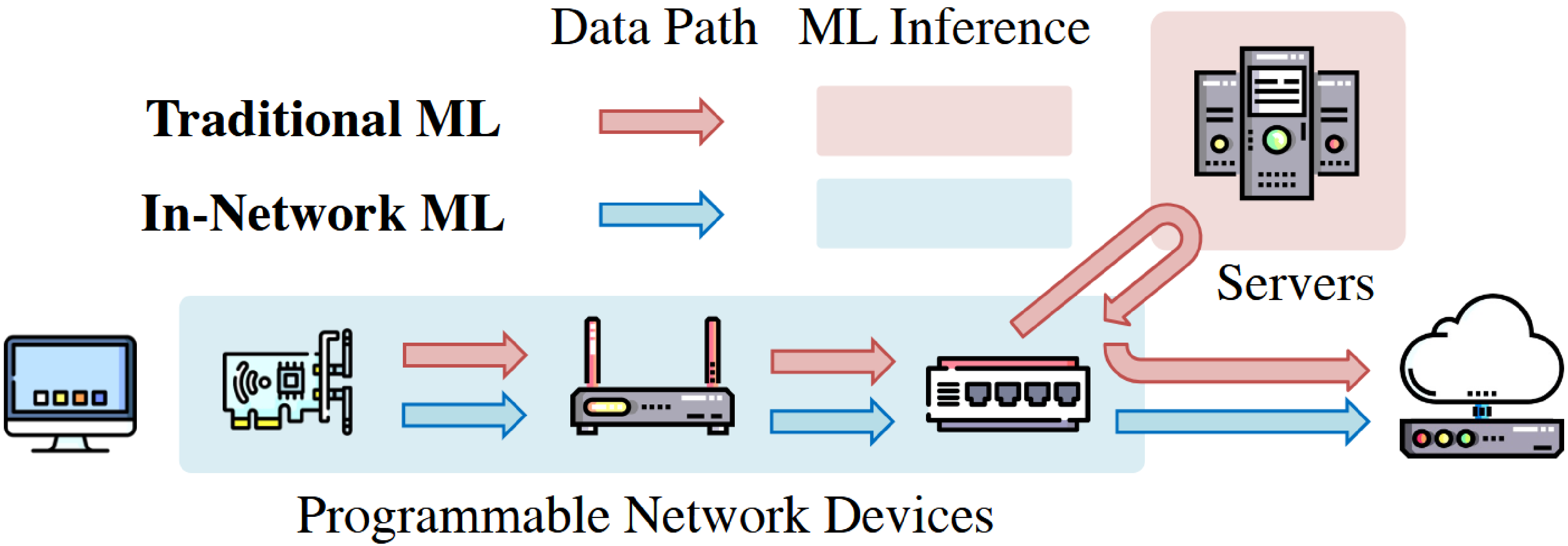
Outline

- Background on in-network ML inference
- In-network inference in switches
- From stateless to hybrid inference approaches
- Conclusion and next steps

"… there will be less of a need for people to make the network work on a day-to-day basis because it will be more automated but I think ***there will be far more things that we can do with the network***, so there will be a massive increase in people ***programming the network*** …"

– Nick McKeown, Stanford
Q&A ONS April '12

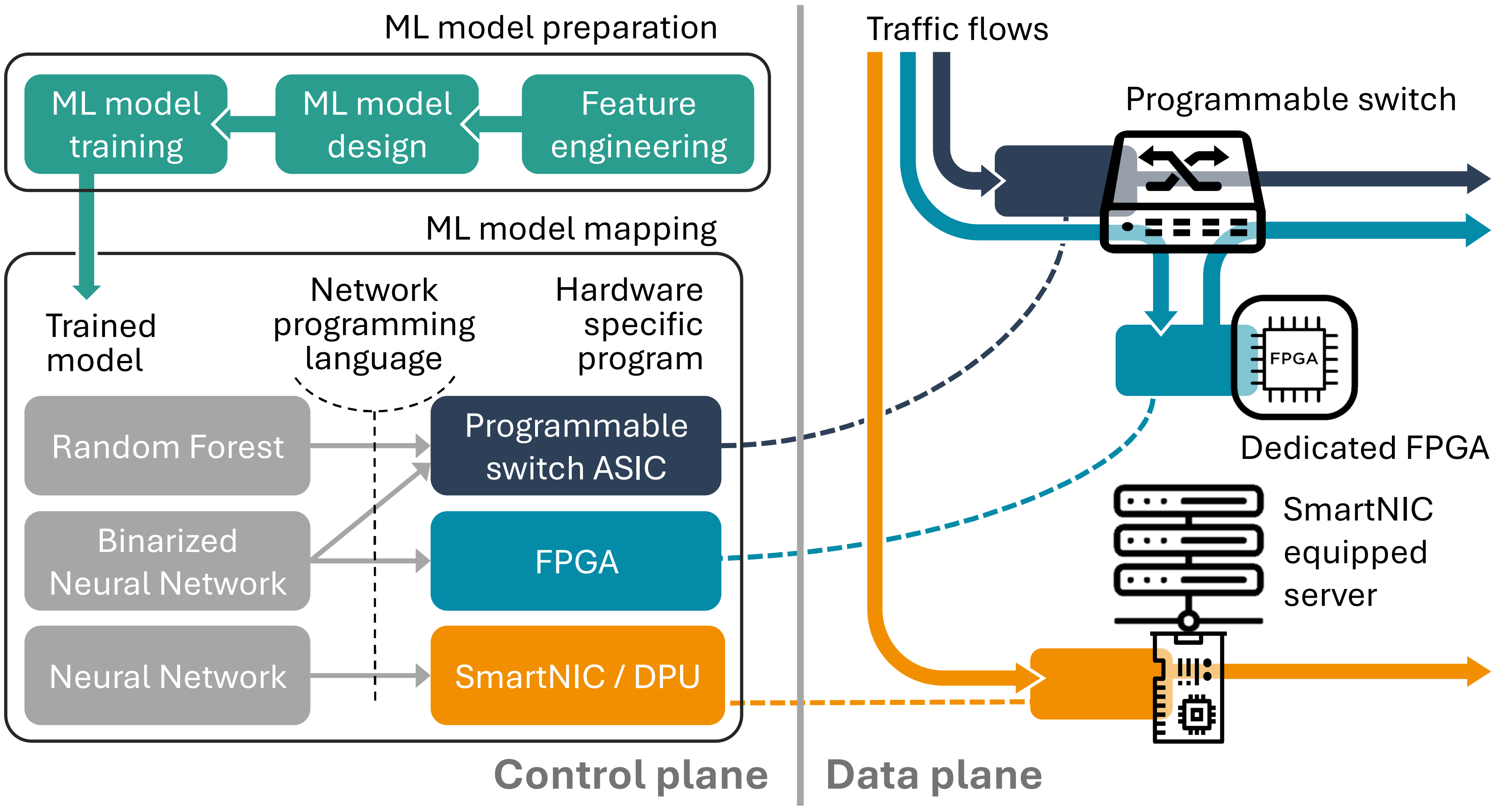
Background



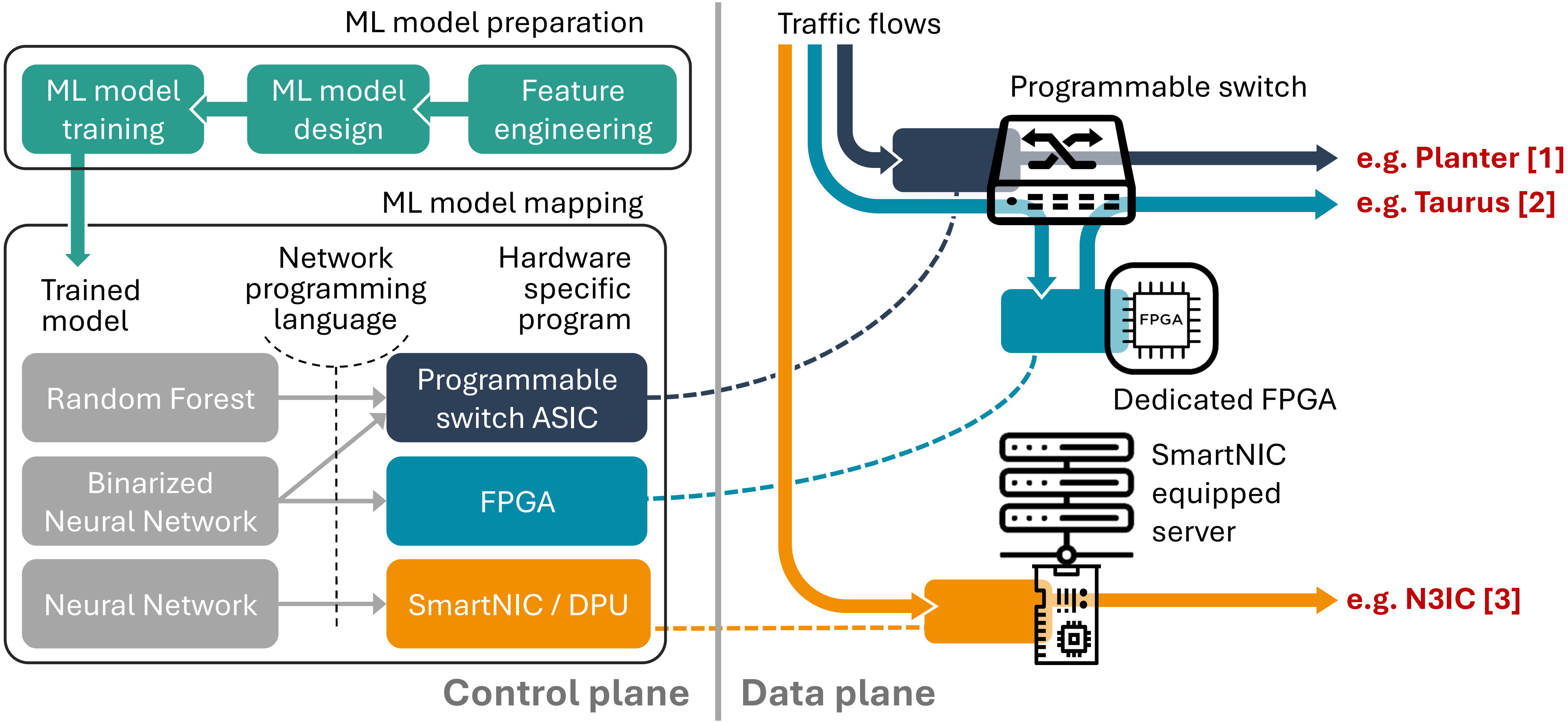
Use cases: cybersecurity, advanced routing, traffic engineering, etc.

Source: C. Zheng, M. Zang, X. Hong, L. Perreault, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Planter: Rapid prototyping of in-network machine learning inference," ACM SIGCOMM Communication Review, 2024.

In-network ML inference overview



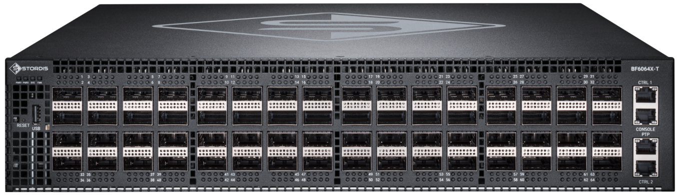
In-network ML inference overview



[1] C. Zheng and N. Zilberman. **Planter: Seeding trees within switches**. In SIGCOMM Poster Session. ACM, 2021.
[2] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun. **Taurus: A Data Plane Architecture for Per-Packet ML**. In ASPLOS. ACM, 2022.
[3] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, R. Bifulco. **Re-architecting traffic analysis with neural network interface cards**. In NSDI. Usenix, 2022.

In-network inference in switches

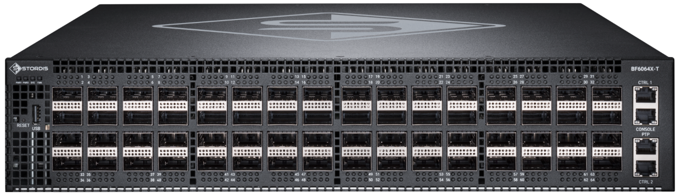
Why switches?



- Very high throughput
- Very low latency
- Many ports, e.g. 32x100 Gbps ports
- Ubiquitous presence in the network

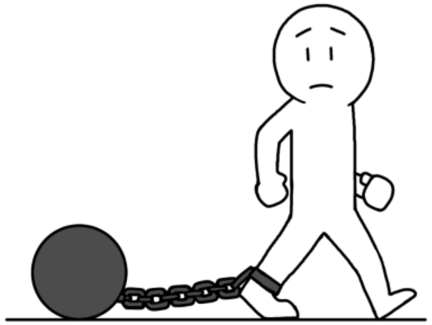
In-network inference in switches

Why switches?



- Very high throughput
- Very low latency
- Many ports, e.g. 32x100 Gbps ports
- Ubiquitous presence in the network

Yet, there are several constraints…

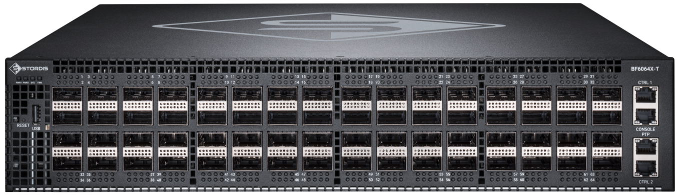


- Low available memory
- Limited support for mathematical operations
- Limited number of operations per packet

In-network inference in switches

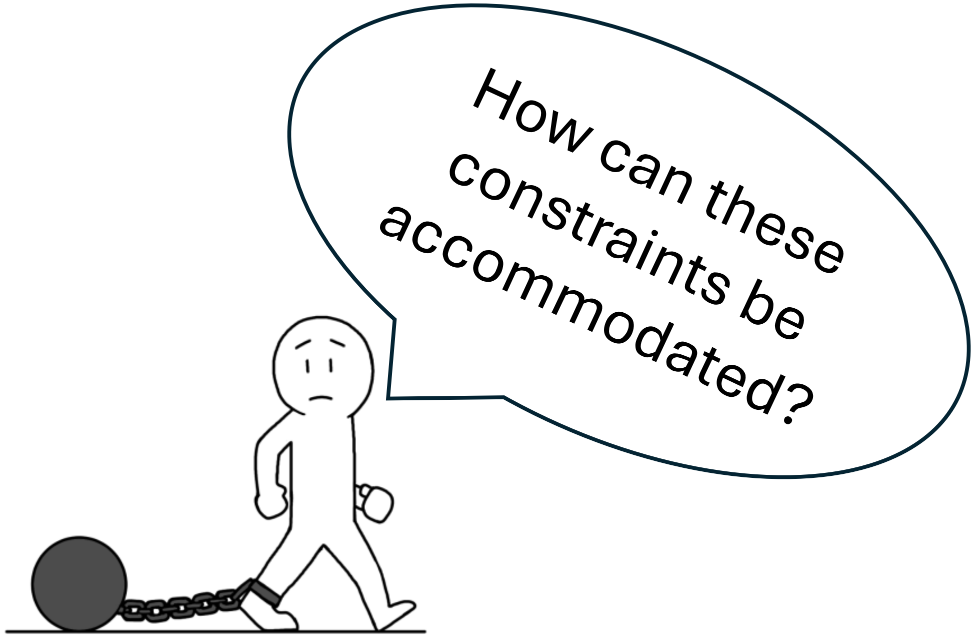
Why switches?

- Very high throughput
- Very low latency
- Many ports, e.g. 32x100 Gbps ports
- Ubiquitous presence in the network



Yet, there are several constraints…

- Low available memory
- Limited support for mathematical operations
- Limited number of operations per packet



Tree-based models for in-network inference

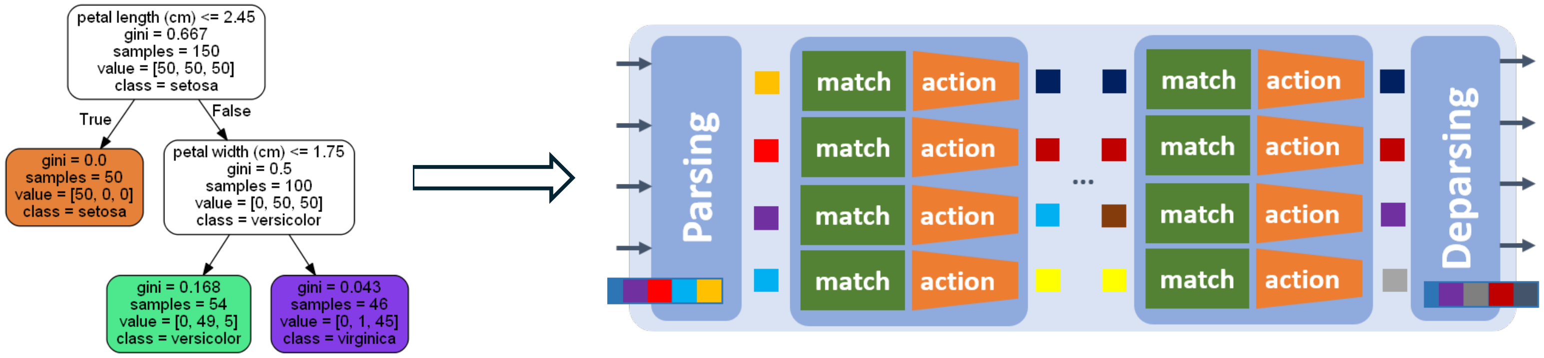
Tree-based models are most suitable for in-switch deployment

- Their simple logical structure makes them easy to map to the switch pipeline

  Zhaoqi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. In ACM HotNets. ACM, NY, USA, 25–33. https://doi.org/10.1145/3365609.3365864.

- They still outperform deep learning on tabular data

  Léo Grinsztajn, Edouard Oyallon, Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? NeurIPS 2022 Datasets and Benchmarks Track, Nov 2022, New Orleans, USA.

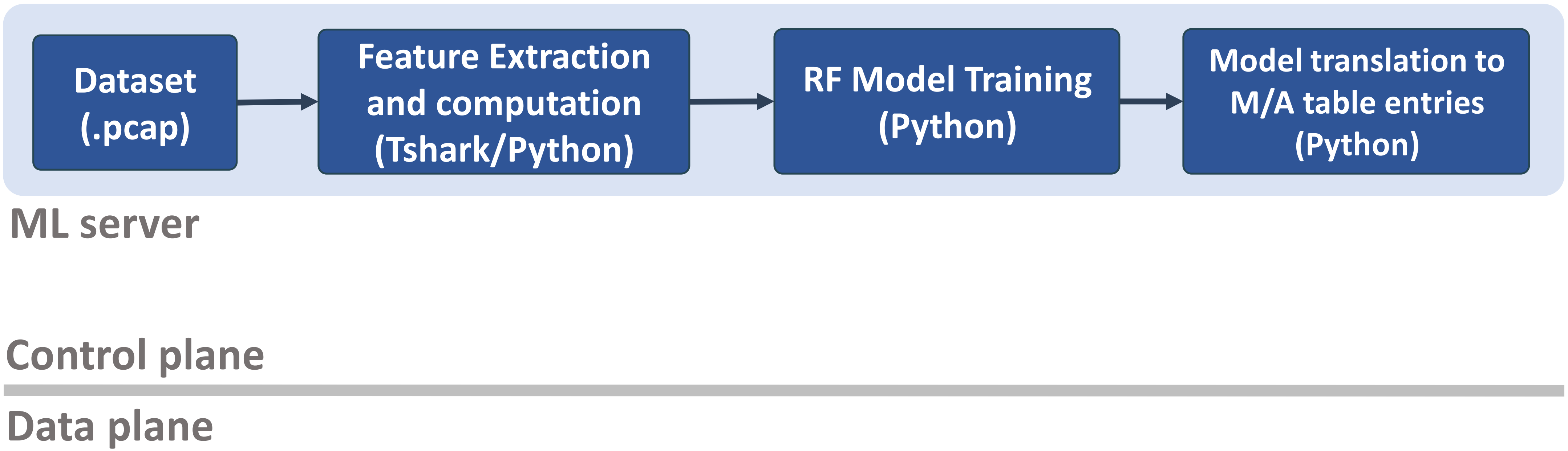


In-network ML inference workflow

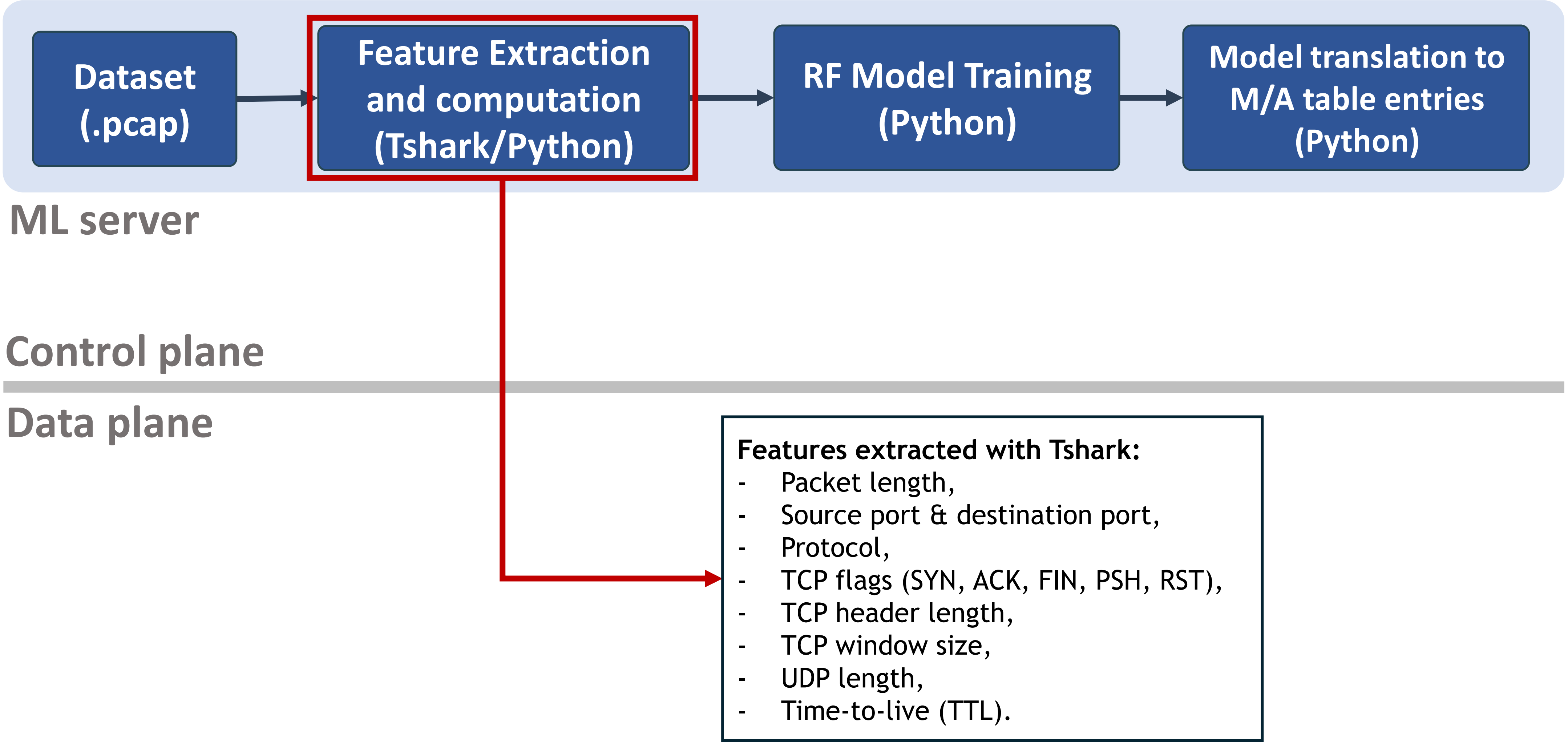
Control plane

Data plane

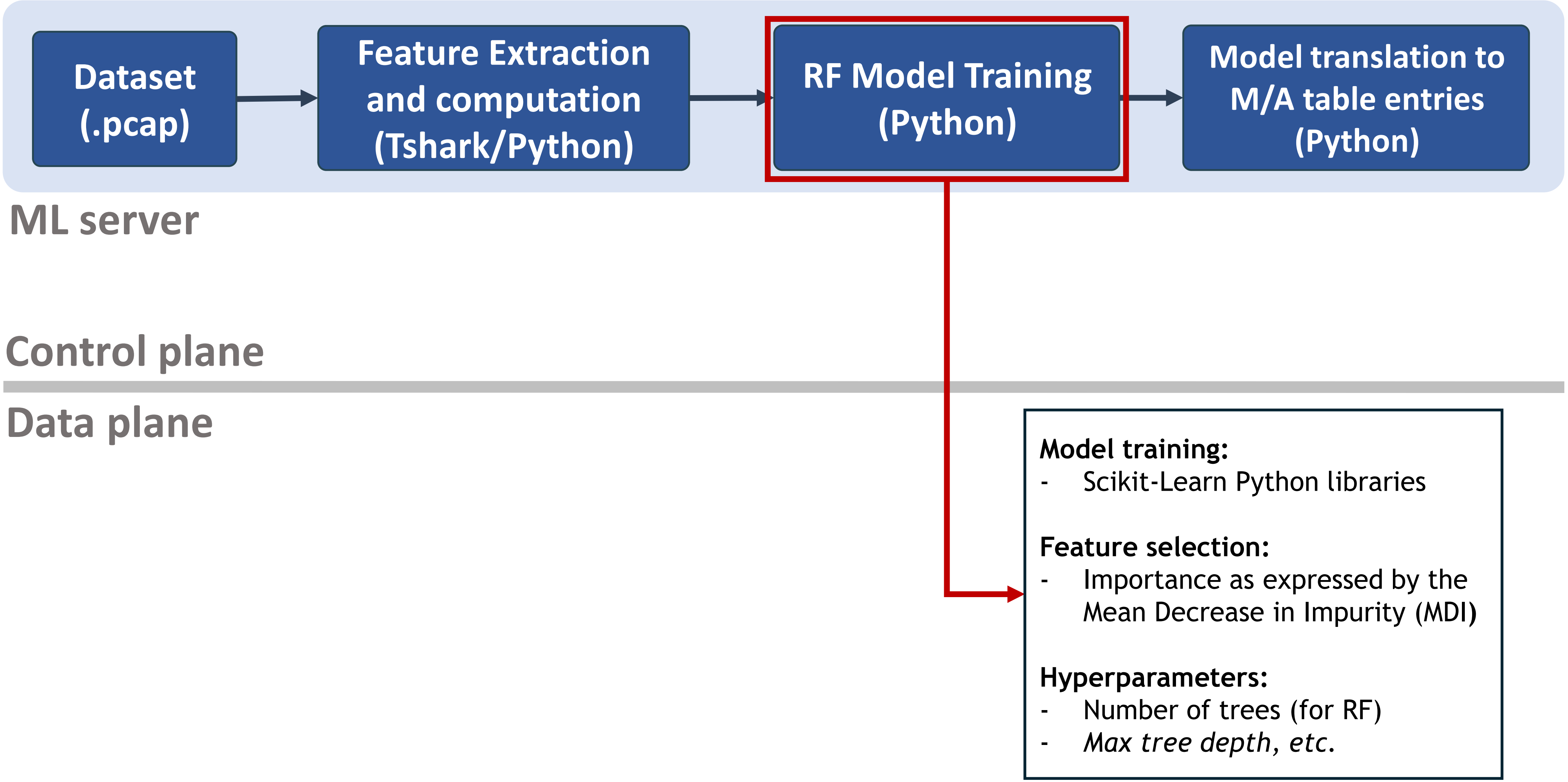
In-network ML inference workflow



In-network ML inference workflow



In-network ML inference workflow



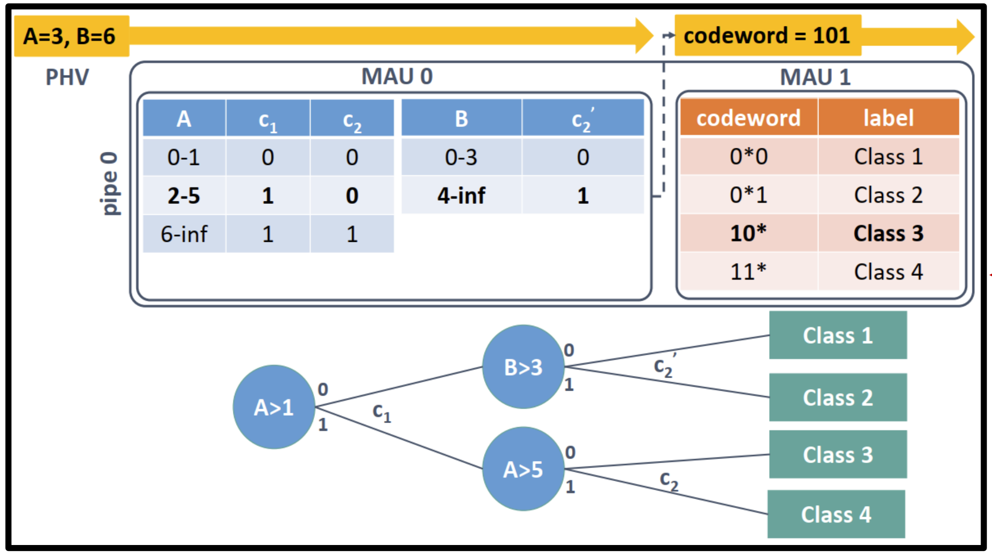
In-network ML inference workflow

Dataset (.pcap) → Feature Extraction and computation (Tshark/Python) → RF Model Training (Python) → Model translation to M/A table entries (Python)

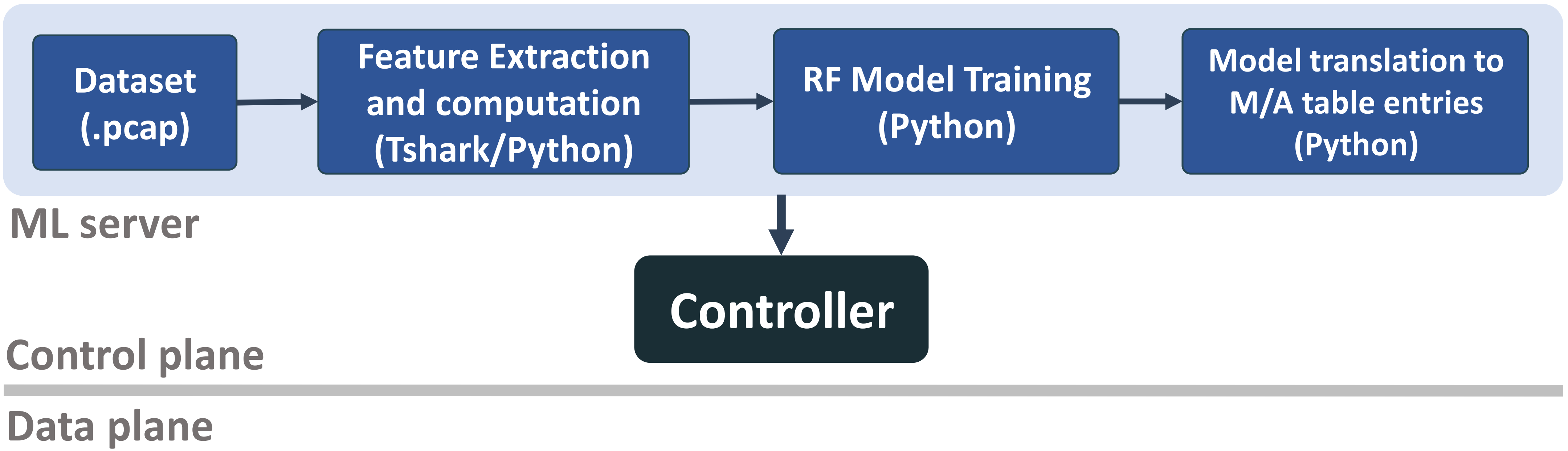
ML server

Control plane

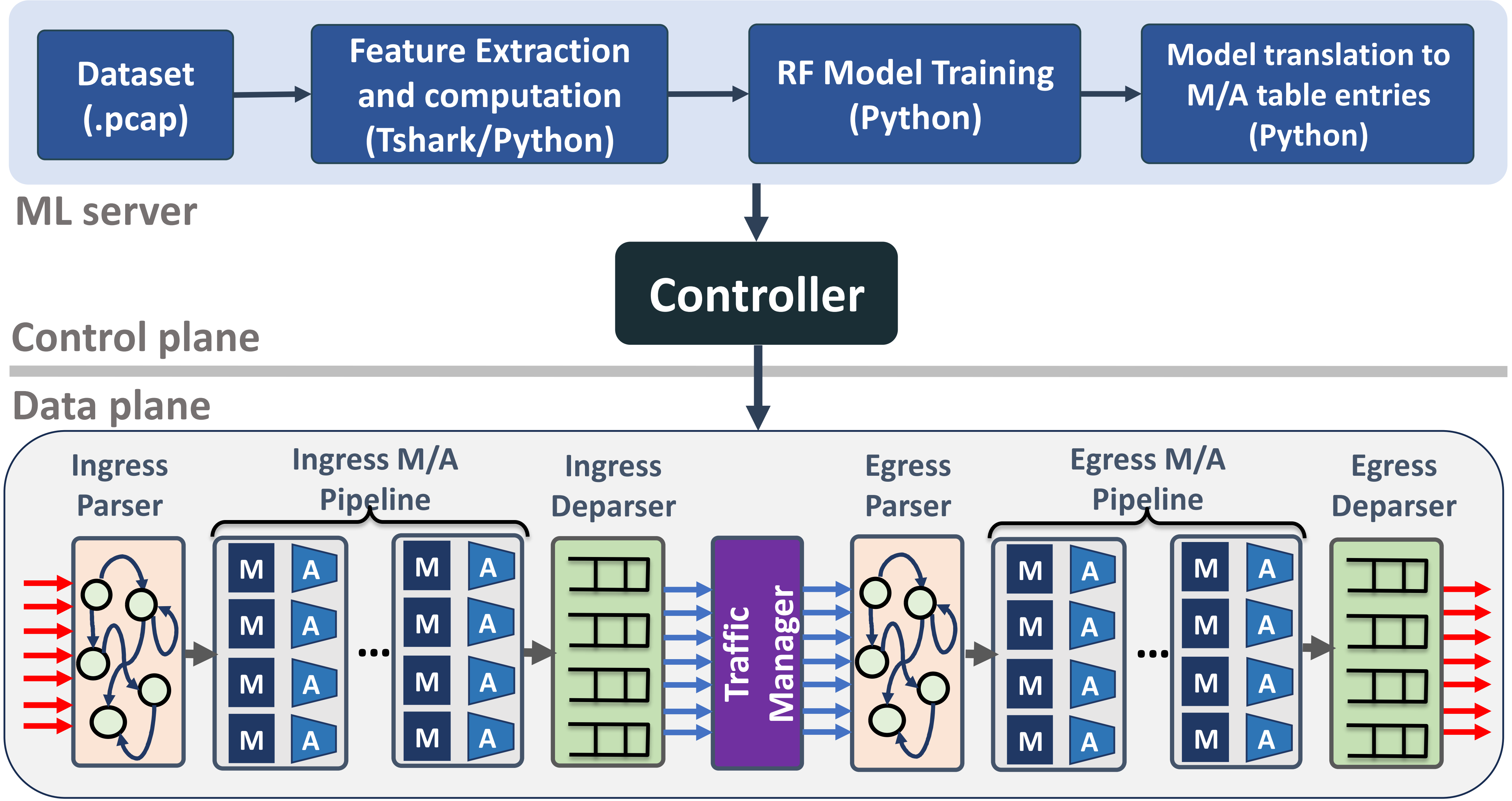
Data plane



In-network ML inference workflow



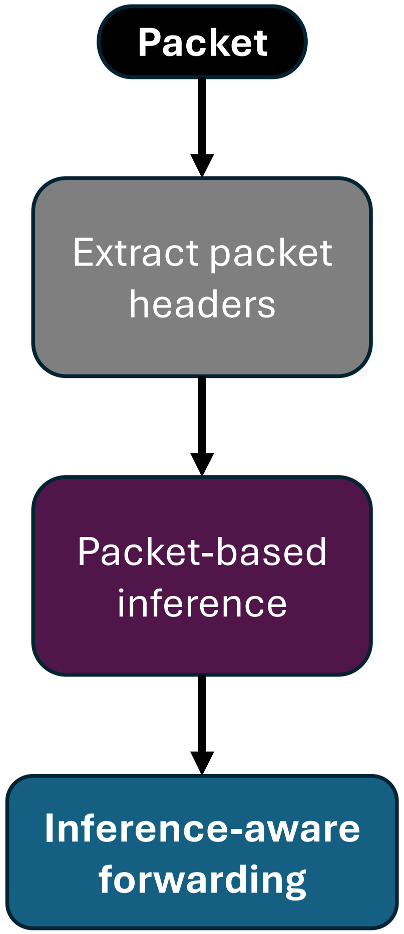
In-network ML inference workflow



In-network ML: from stateless to hybrid approaches

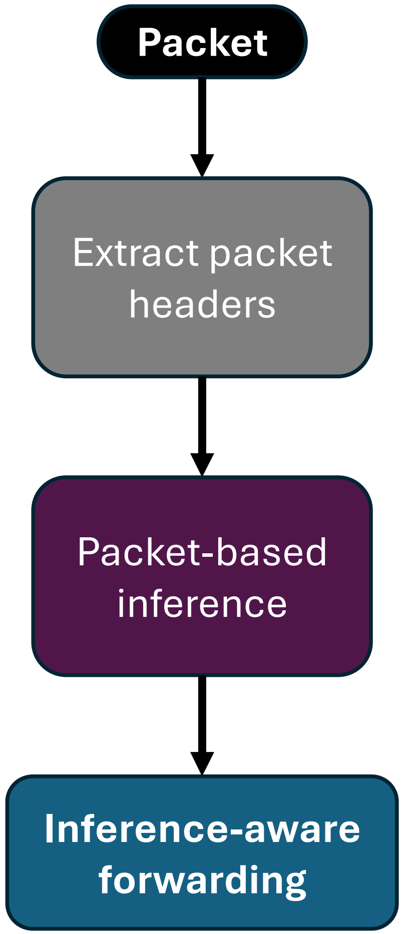
Per-packet (stateless) **Per-flow (stateful)** **Joint packet-flow (hybrid)**

In-network ML: from stateless to hybrid approaches



Per-packet (stateless) **Per-flow (stateful)** **Joint packet-flow (hybrid)**

In-network ML: from stateless to hybrid approaches



- Intuitive/natural
- All packets are classified
- No rich per-flow statistics
- Limited accuracy in complex tasks

Per-packet (stateless) **Per-flow (stateful)** **Joint packet-flow (hybrid)**

Henna: Stateless Hierarchical packet-level ML inference

Motivation

Inference task → Train a single model for the task and map it to the switch

Henna: Stateless Hierarchical packet-level ML inference

Motivation

Inference task → Train a single model for the task and map it to the switch

All prior works adopt this approach known as *flat classification*

Henna: Stateless Hierarchical packet-level ML inference

Motivation

Inference task → Train a single model for the task and map it to the switch

All prior works adopt this approach known as *flat classification*

Monolithic classifiers can be too complex for challenging tasks

Henna: Stateless Hierarchical packet-level ML inference

Motivation

Inference task → Train a single model for the task and map it to the switch

All prior works adopt this approach known as *flat classification*

Monolithic classifiers can be too complex for challenging tasks

Breaking down tasks hierarchically can simplify them

Henna: Stateless Hierarchical packet-level ML inference

Illustration

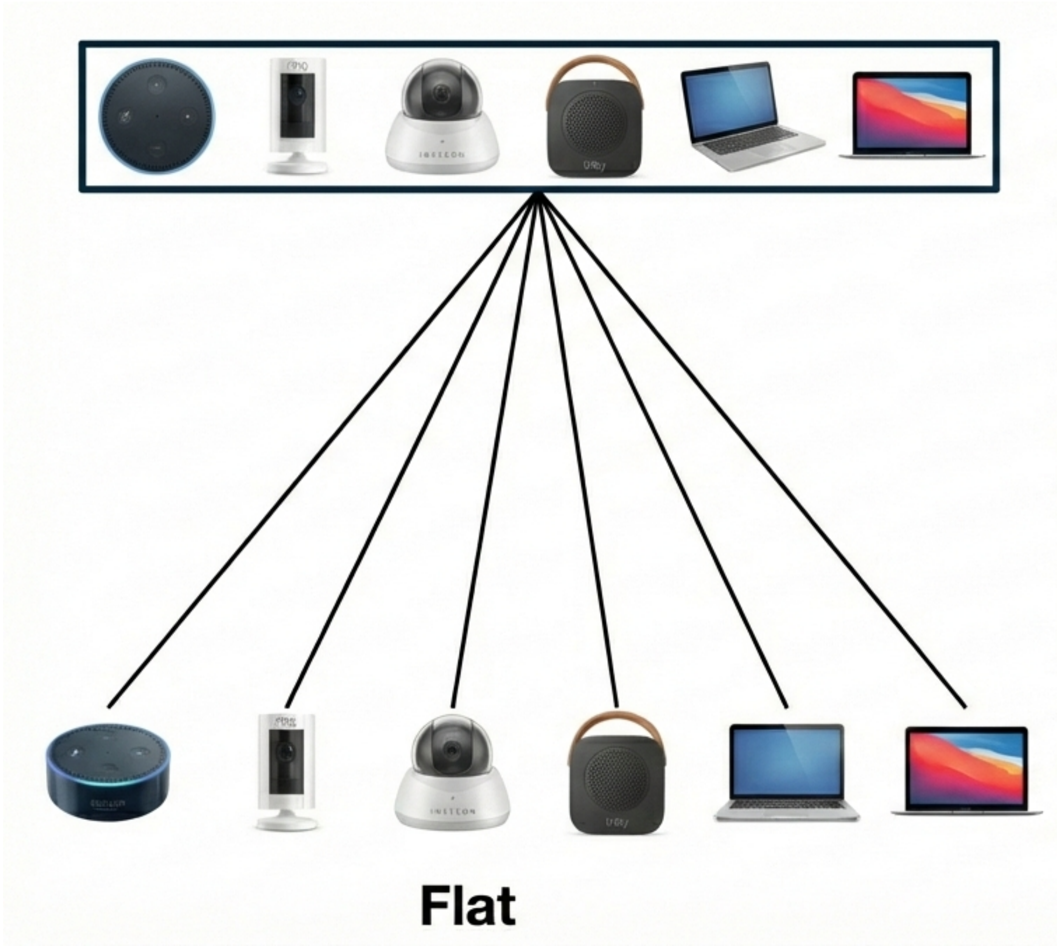


Image generated by Gemini

Henna: Stateless Hierarchical packet-level ML inference

Illustration

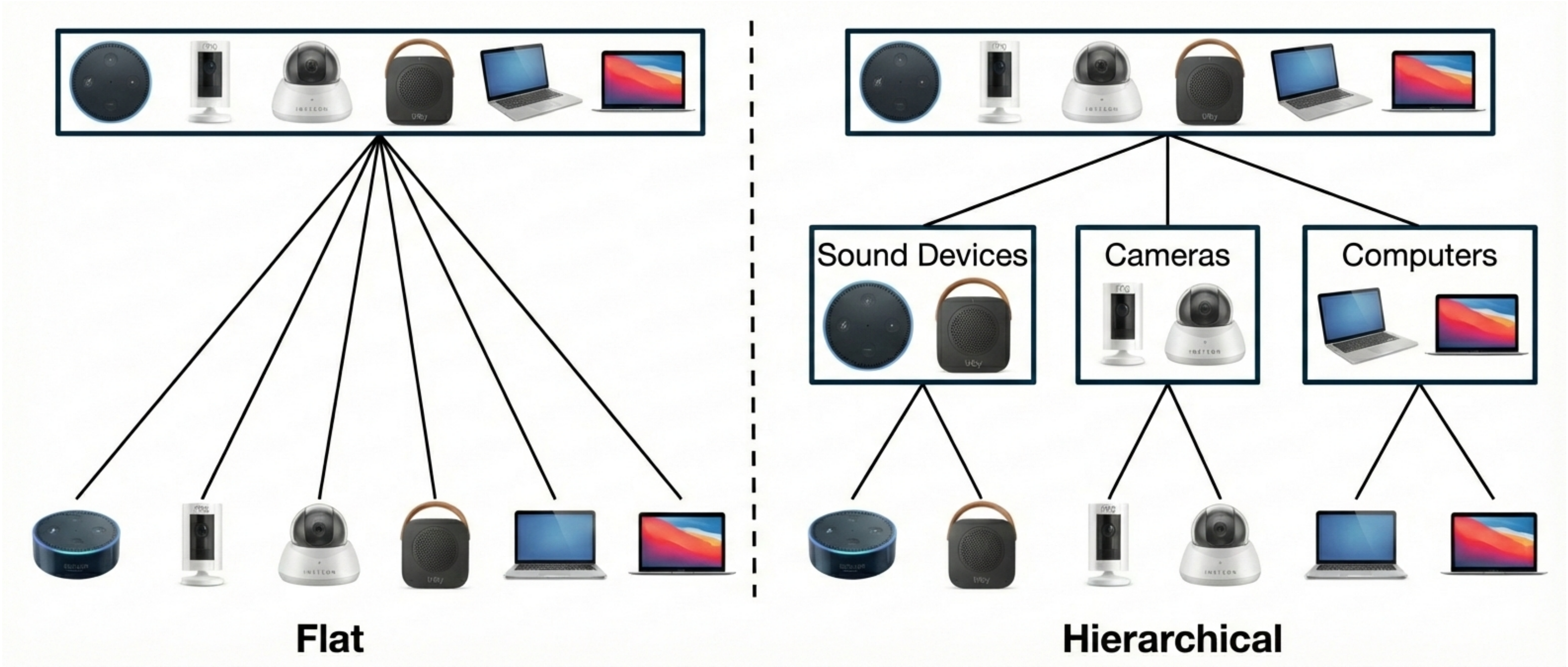
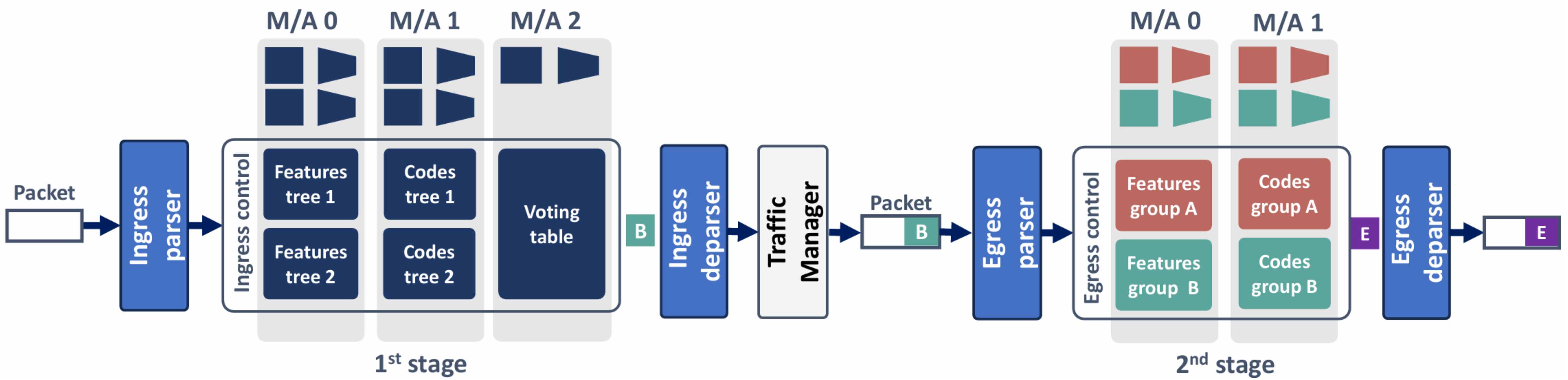


Image generated by Gemini

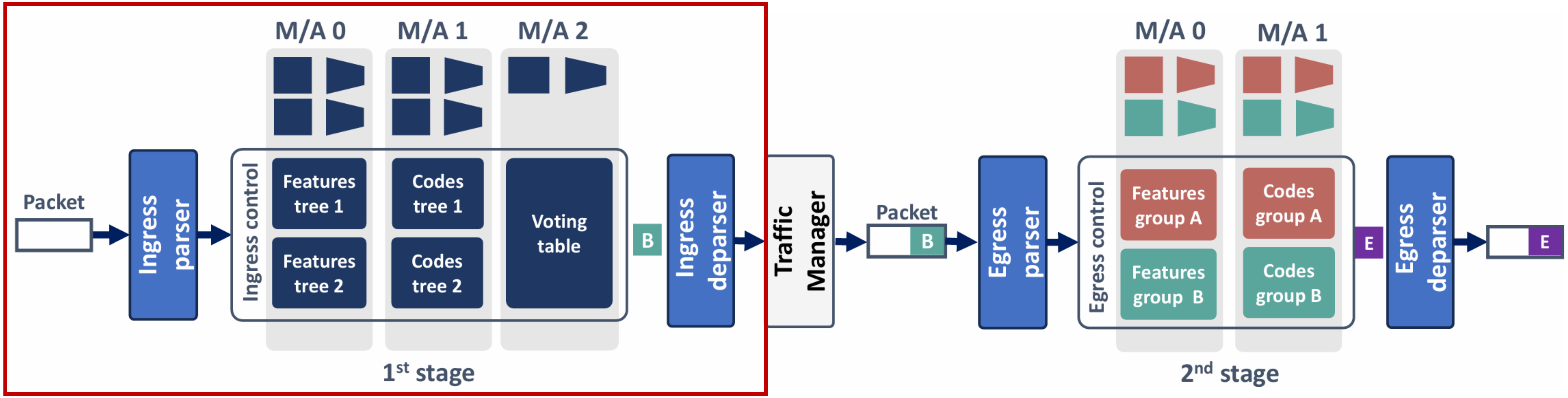
Henna: Stateless Hierarchical packet-level ML inference

Our proposal



Henna: Stateless Hierarchical packet-level ML inference

Our proposal




```
/* Feature tables for first stage RF*/
table tbl_s1_f0{
    key = {meta.hdr_srcport: range @name("s1_f0");}
    actions = {@defaultonly nop; SetCode_s1_f0;}
    size = 350;
    const default_action = nop();
}
```

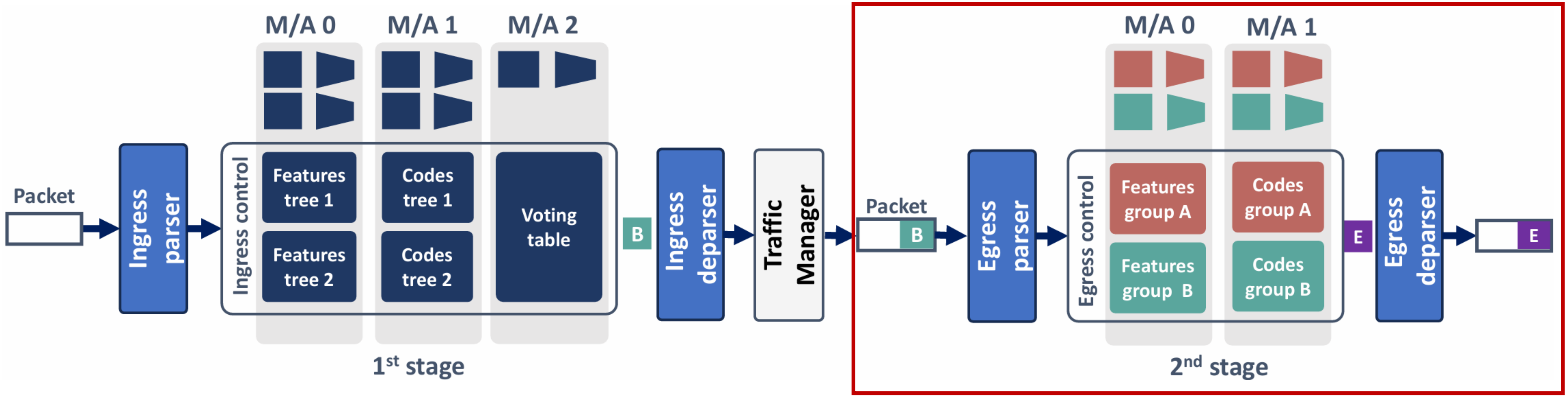
```
/* Code tables for first stage RF*/
table tbl_s1_cw0{
    key = {meta.cw_s1_t0: ternary;}
    actions = {@defaultonly nop; SetClass_s1_t0;}
    size = 490;
    const default_action = nop();
}
```

```
apply {
    // apply feature tables of 1st stage
    tbl_s1_f0.apply();
    tbl_s1_f1.apply();
    tbl_s1_f2.apply();
    tbl_s1_f3.apply();
    tbl_s1_f4.apply();
    tbl_s1_f5.apply();

    // apply code tables of 1st stage
    tbl_s1_cw0.apply();
    tbl_s1_cw1.apply();
    tbl_s1_cw2.apply();
```

Henna: Stateless Hierarchical packet-level ML inference

Our proposal




```
/* Feature tables for the second stage DT's */
    // computers - g4
    table tbl_s2_g4_f0{
        key = {meta.total_len: range @name("s2_g4_f0");}
        actions = {@defaultonly nop; SetCode_s2_g4_f0;}
        size = 60;
        const default_action = nop();
    }
```

```
/* Code tables for second stage DT's*/
// g4
table tbl_s2_g4{
    key = {meta.cw_s2_g4: ternary;}
    actions = {@defaultonly nop; SetClass_s2_g4;}
    size = 490;
    const default_action = nop();
}
```

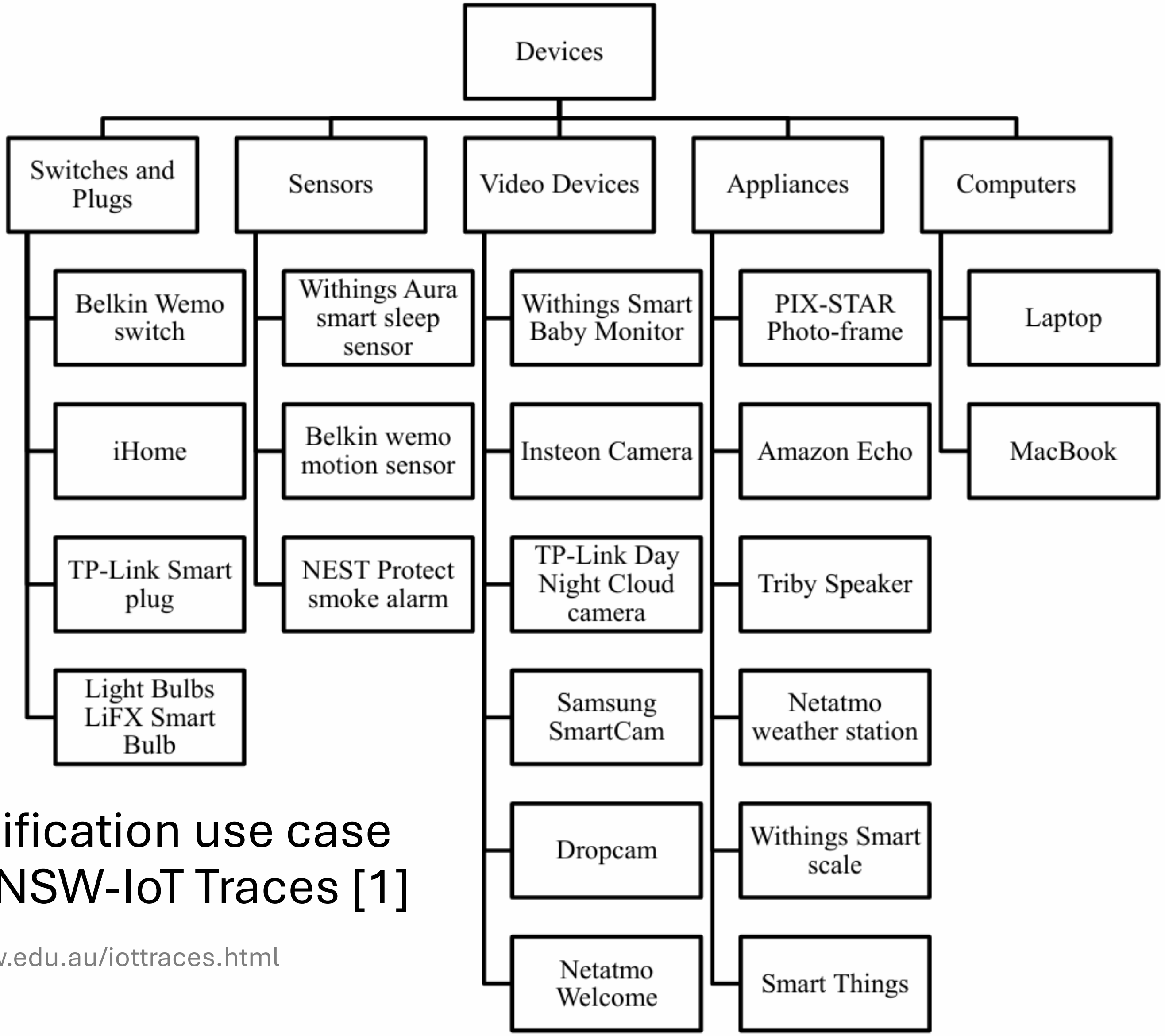
```
apply {

    // check result of first stage to determine which 2nd stage model to apply

    if (meta.group_class == 5){ //computers
        // apply the feature tables
        tbl_s2_g4_f0.apply();
        tbl_s2_g4_f1.apply();
        tbl_s2_g4_f2.apply();
        tbl_s2_g4_f3.apply();
        // apply the code tables
        tbl_s2_g4.apply();
    }
    else if(meta.group_class == 4){//appliances
```

Henna: Stateless Hierarchical packet-level ML inference

Evaluation

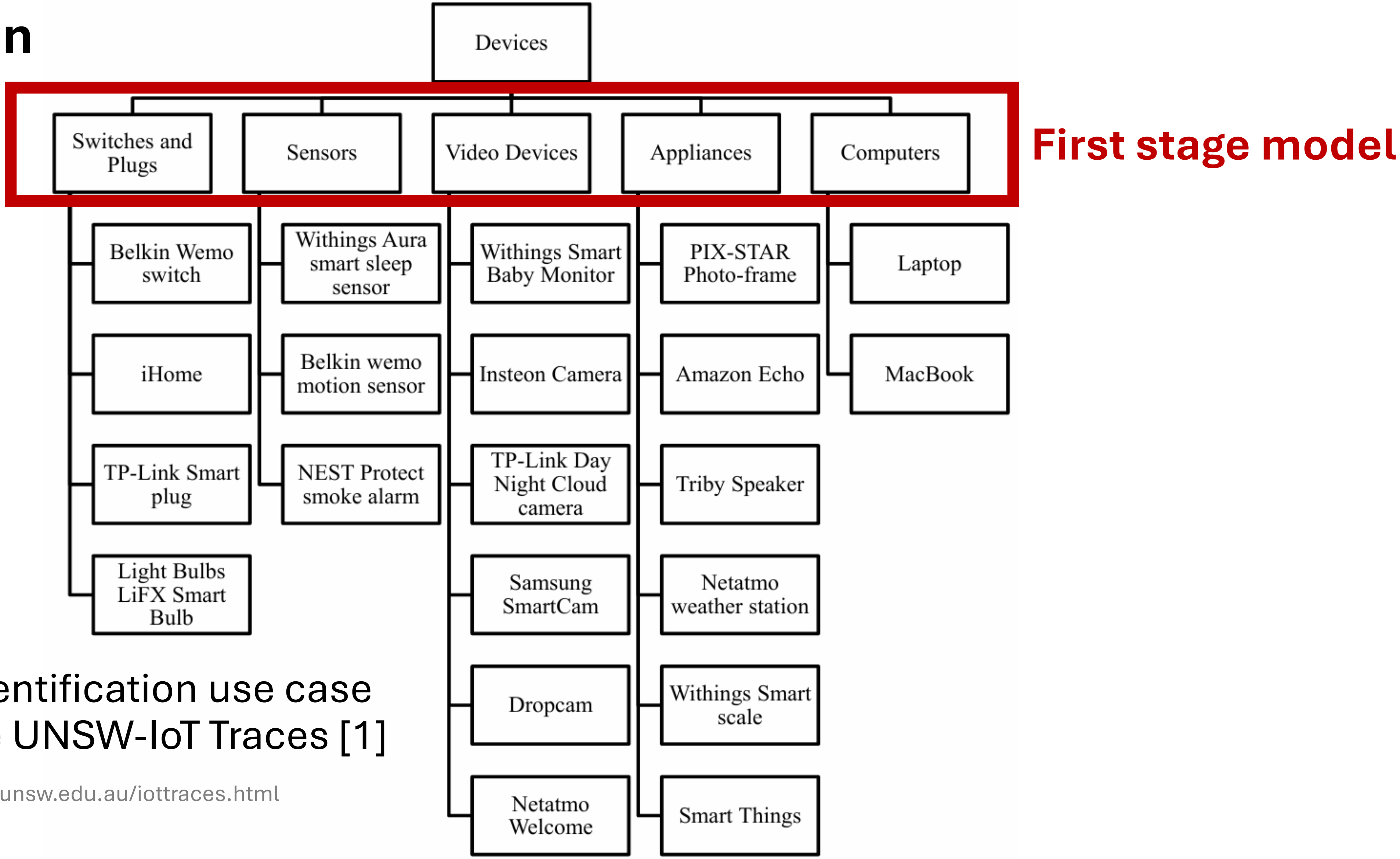


IoT device identification use case based on the UNSW-IoT Traces [1]

[1] https://iotanalytics.unsw.edu.au/iottraces.html

Henna: Stateless Hierarchical packet-level ML inference

Evaluation

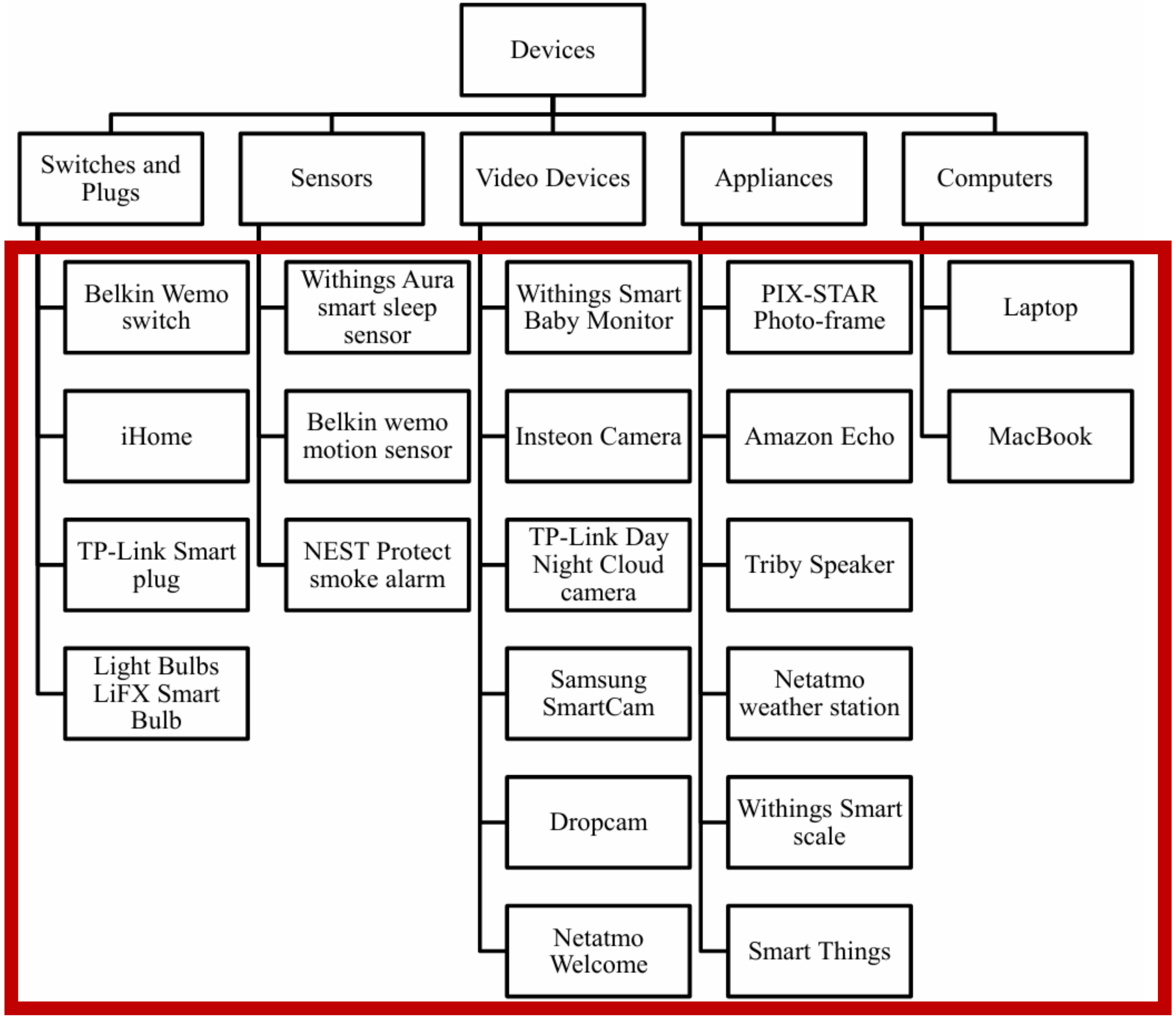


IoT device identification use case based on the UNSW-IoT Traces [1]

[1] https://iotanalytics.unsw.edu.au/iottraces.html

Henna: Stateless Hierarchical packet-level ML inference

Evaluation



Second-stage models

Henna: Stateless Hierarchical packet-level ML inference

Results

Metric	1-Stage	Henna		
		Value	Gain	
			Absolute	Relative
Precision	65.38%	70.50%	5.12%	7.83%
Recall	55.50%	70.95%	15.45%	27.84%
F1 score	55.54%	67.50%	11.95%	21.52%

Classification accuracy

Resource	1-Stage	Henna
Overall (w.r.t. total)	5.10%	8.50%
Overall (w.r.t. switch.p4)	13.42%	22.27%
Match-Action units	8	10
Latency at ingress	35.42%	43.40%
Latency at egress	59.15%	62.68%

Resource usage

Henna: Stateless Hierarchical packet-level ML inference

Results

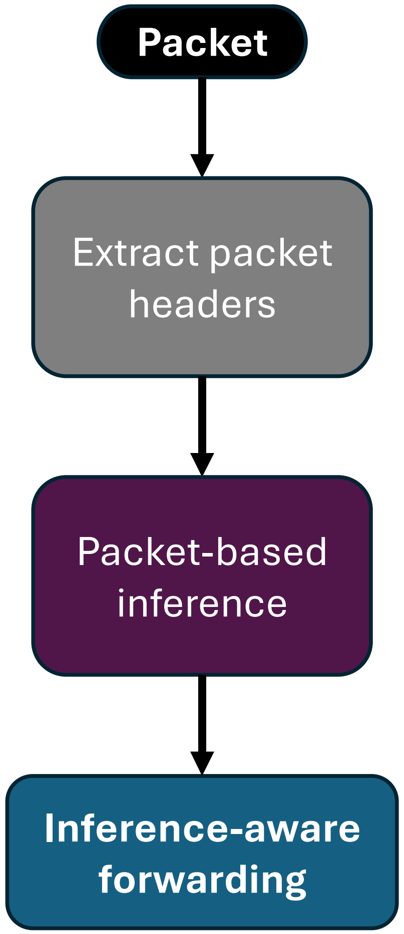
Metric	1-Stage	Henna		
		Value	Gain	
			Absolute	Relative
Precision	65.38%	70.50%	5.12%	7.83%
Recall	55.50%	70.95%	15.45%	27.84%
F1 score	55.54%	67.50%	11.95%	21.52%

Classification accuracy

Resource	1-Stage	Henna
Overall (w.r.t. total)	5.10%	8.50%
Overall (w.r.t. switch.p4)	13.42%	22.27%
Match-Action units	8	10
Latency at ingress	35.42%	43.40%
Latency at egress	59.15%	62.68%

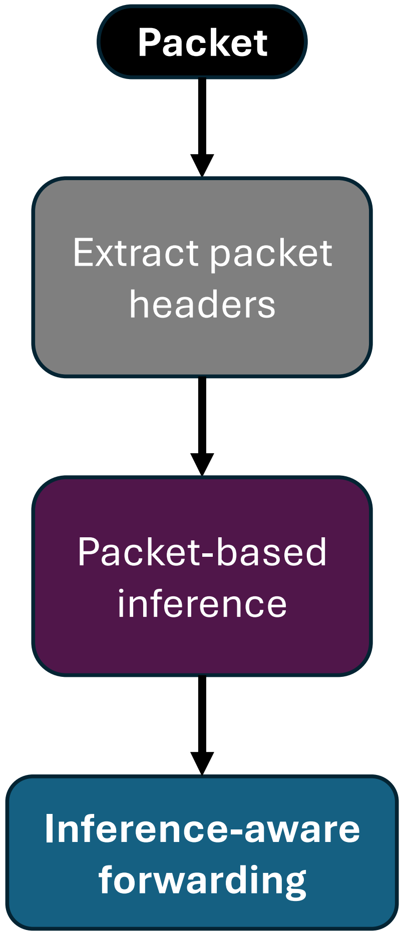
Resource usage

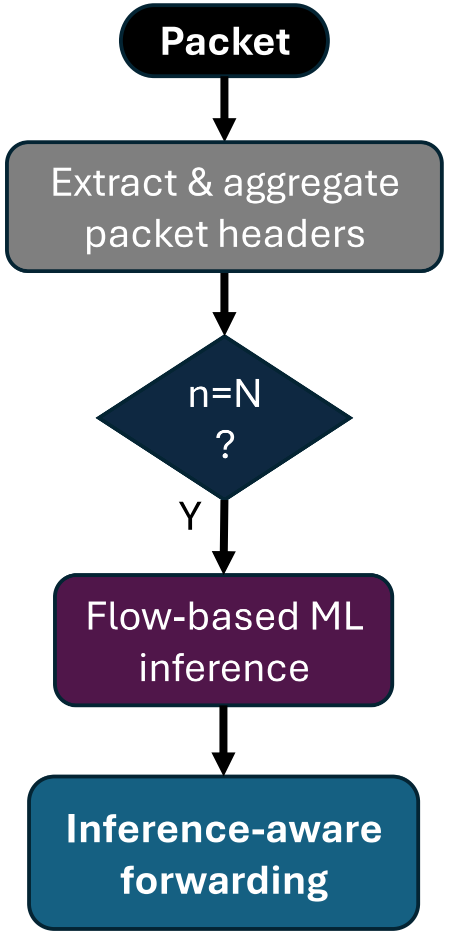
In-network ML: from stateless to hybrid approaches



Per-packet (stateless) **Per-flow (stateful)** **Joint packet-flow (hybrid)**

In-network ML: from stateless to hybrid approaches



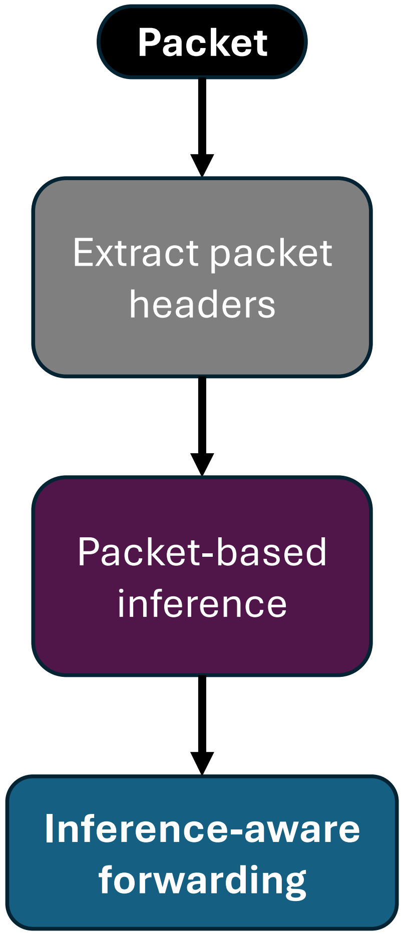


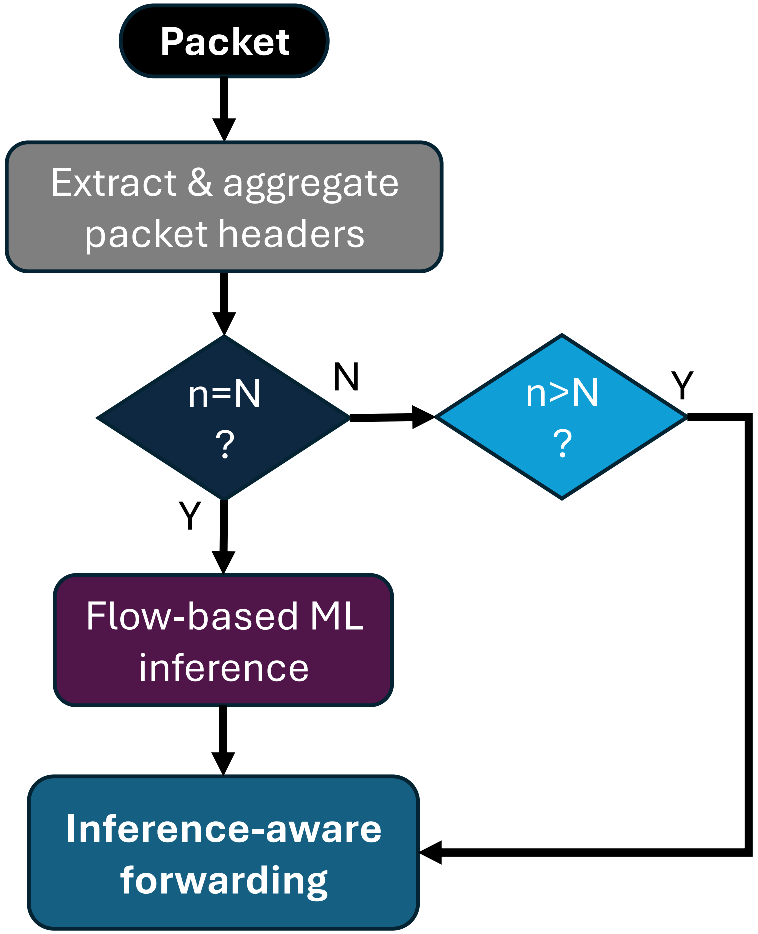
Per-packet (stateless)

Per-flow (stateful)

Joint packet-flow (hybrid)

In-network ML: from stateless to hybrid approaches



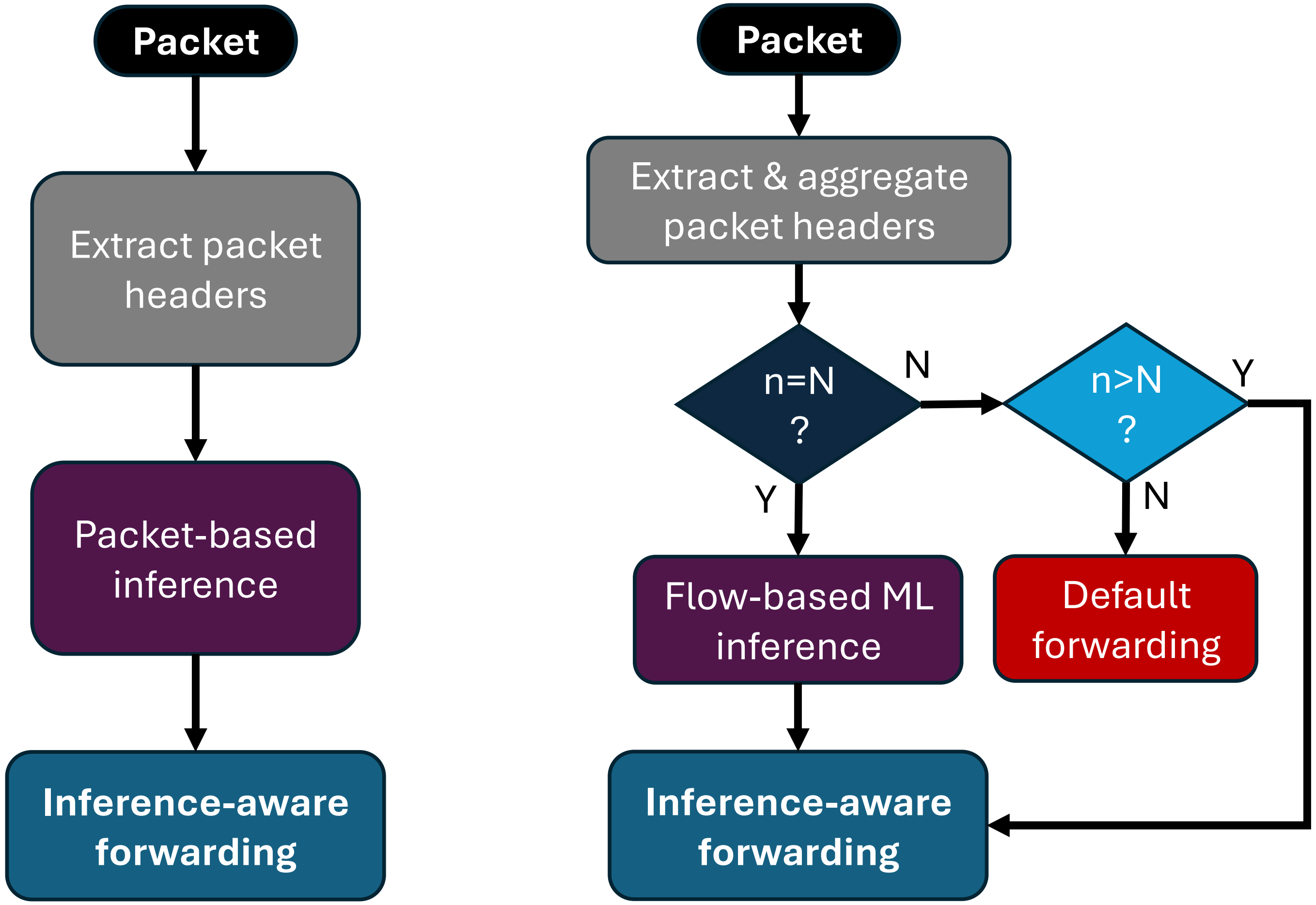


Per-packet (stateless)

Per-flow (stateful)

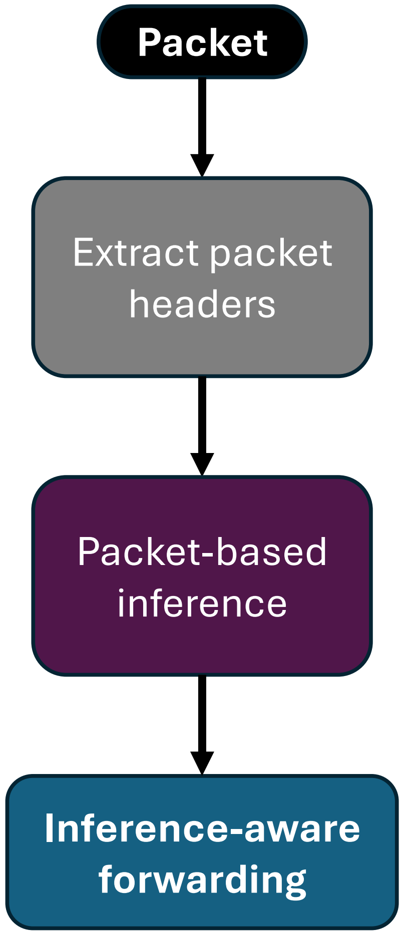
Joint packet-flow (hybrid)

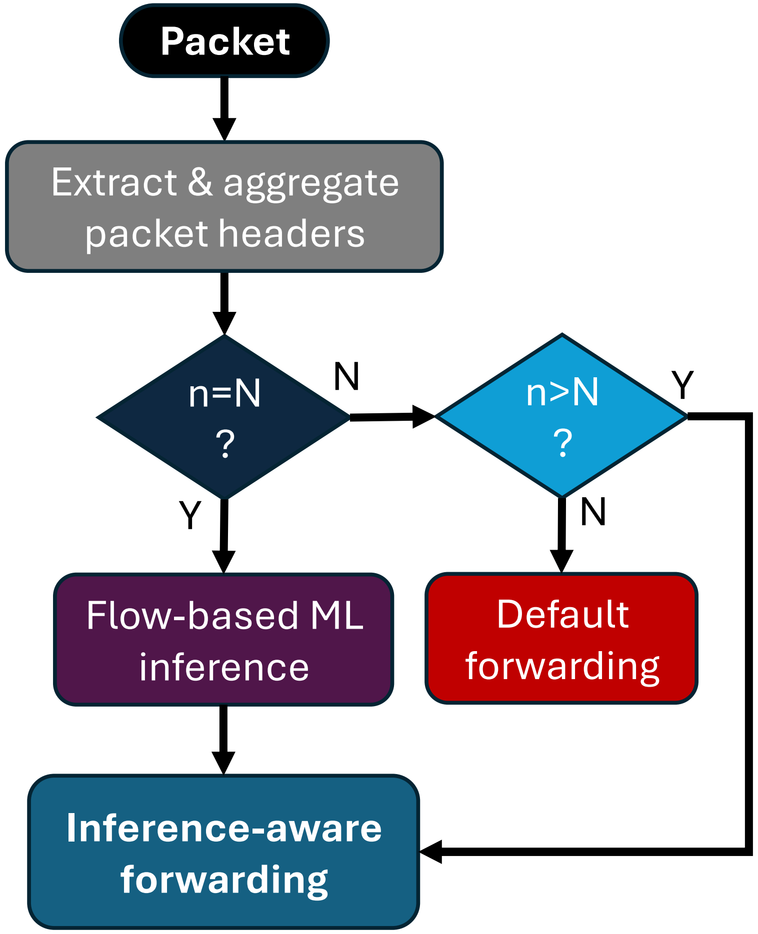
In-network ML: from stateless to hybrid approaches



Per-packet (stateless) **Per-flow (stateful)** **Joint packet-flow (hybrid)**

In-network ML: from stateless to hybrid approaches





- Richer flow-level features
- Policies implemented at flow-level
- Early packets go unclassified

Per-packet (stateless) **Per-flow (stateful)** **Joint packet-flow (hybrid)**

Flowrest: Practical stateful flow-level inference

Motivation

Flow = $\mathbb{P}$(src IP, dst IP, src port, dst port, protocol)

Packet-Level Approaches

- Relatively low accuracy in complex scenarios
- Do not use rich flow-level (FL) features

Flow-level classification provides more context

e.g., by leveraging relationships between flow packets

Most network-wide policies are implemented at flow-level

e.g., for QoS and QoE management

Flowrest: Practical stateful flow-level inference

Motivation

Flow = $\mathbb{P}$(src IP, dst IP, src port, dst port, protocol)

Packet-Level Approaches

- Relatively low accuracy in complex scenarios
- Do not use rich flow-level (FL) features

Flow-level classification provides more context

e.g., by leveraging relationships between flow packets

Most network-wide policies are implemented at flow-level

e.g., for QoS and QoE management

Deploying stateful FL models in switches involves maintaining state and computing FL features

Flowrest: Practical stateful flow-level inference

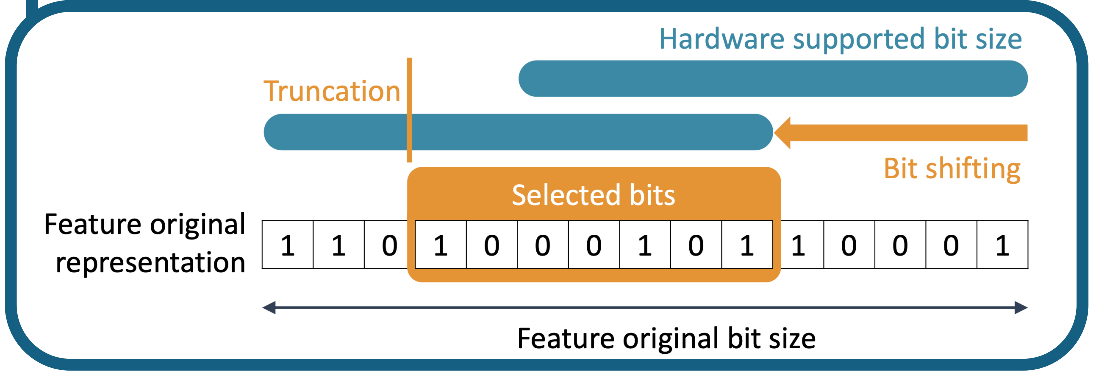
Proposed solution

Flow-level classification

- Features calculated over multiple packets in the same flow
- Features such as min, max, mean pkt sizes & IATs
- More effective for difficult inference tasks

Tailored Random Forest design

- Hardware-aware bit-level feature representation
- Hardware-constrained Random Forest hyper-parametrization

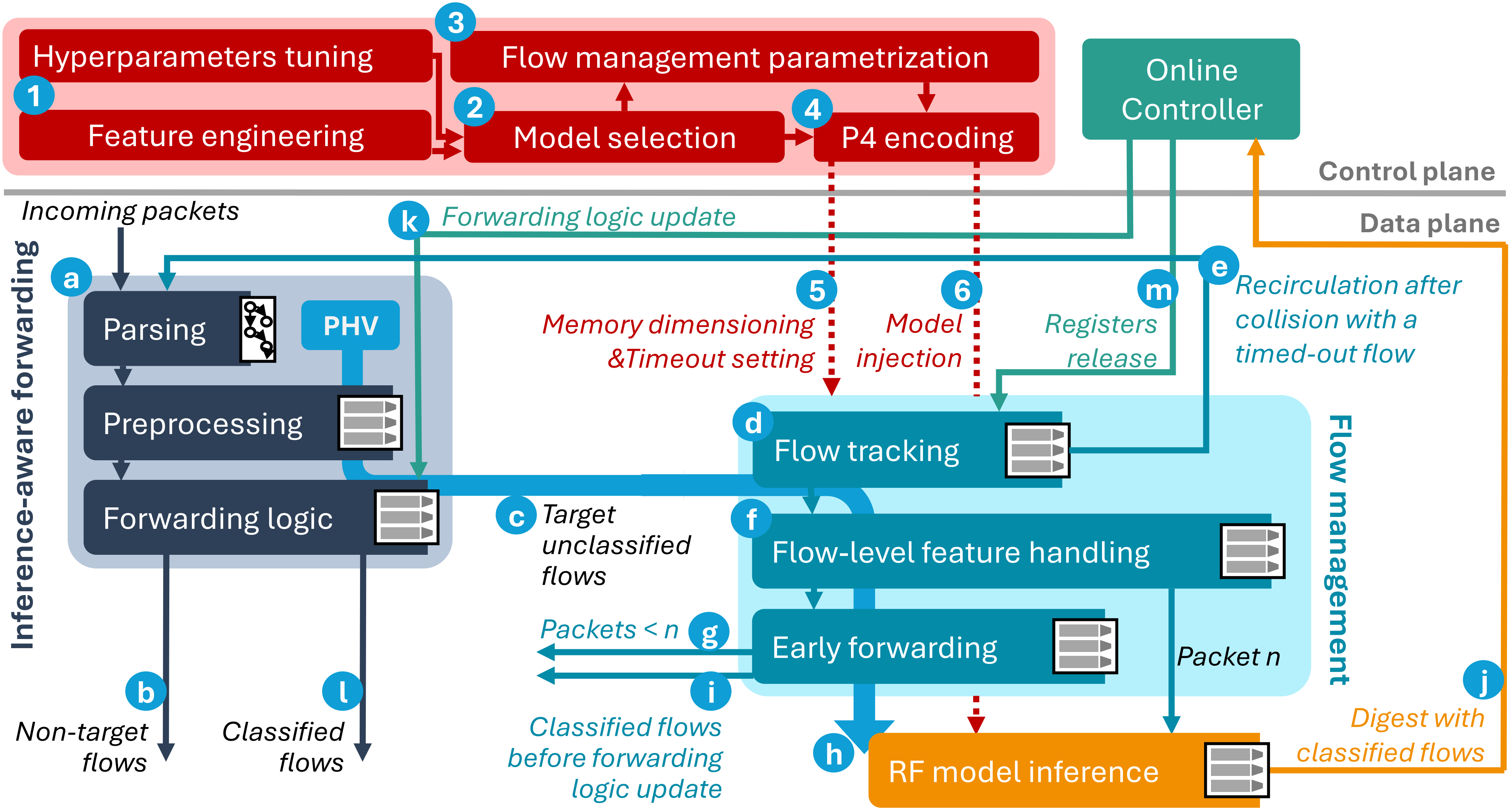


General purpose & open-source

- Is not tied to any use case
- Can convert any stateless solution to flow level
- First open-source implementation of in-switch flow-level RF

Flowrest: Practical stateful flow-level inference

System overview



Flowrest: Practical stateful flow-level inference


```
apply {
        // compute the current time
        meta.now_timestamp = (bit<32>)(ig_prsr_md.global_tstamp[47:20]);  //msec

        //compute flow_ID and hash index
        get_flow_ID(meta.hdr_srcport, meta.hdr_dstport);
        get_register_index(meta.hdr_srcport, meta.hdr_dstport);
        flow_action_table.apply();

        if (meta.f_action != 0) {
            // Recirculated flow because of timeout collision
            if (hdr.recirc.isValid()){
                meta.is_first = 1;
                meta.reg_status = read_reg_status.execute(meta.register_index);
                update_flow_ID.execute(meta.register_index);
                meta.pkt_count = read_pkt_count.execute(meta.register_index);
                meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);

                update_reg_time_occ.execute(meta.register_index);
                // Invalidate the recirculation header
                hdr.recirc.setInvalid();
                hdr.ethernet.ether_type = TYPE_IPV4;
                ipv4_forward(260);
            }
            else{
                // modify status register
                meta.reg_status = read_reg_status.execute(meta.register_index);

                // check if register array is empty
                if (meta.reg_status == 0){ // we do not yet know this flow
                    meta.is_first = 1;
                    update_flow_ID.execute(meta.register_index);
                    meta.pkt_count = read_pkt_count.execute(meta.register_index);
                    meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                    meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                    meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);
                    update_reg_time_occ.execute(meta.register_index);
                    ipv4_forward(260);
                }
```

Get timestamp and compute hashes

Flowrest: Practical stateful flow-level inference


```
apply {
        // compute the current time
        meta.now_timestamp = (bit<32>)(ig_prsr_md.global_tstamp[47:20]);  //msec

        //compute flow_ID and hash index
        get_flow_ID(meta.hdr_srcport, meta.hdr_dstport);
        get_register_index(meta.hdr_srcport, meta.hdr_dstport);
        flow_action_table.apply();

        if (meta.f_action != 0) {
            // Recirculated flow because of timeout collision
            if (hdr.recirc.isValid()){
                meta.is_first = 1;
                meta.reg_status = read_reg_status.execute(meta.register_index);
                update_flow_ID.execute(meta.register_index);
                meta.pkt_count = read_pkt_count.execute(meta.register_index);
                meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);

                update_reg_time_occ.execute(meta.register_index);
                // Invalidate the recirculation header
                hdr.recirc.setInvalid();
                hdr.ethernet.ether_type = TYPE_IPV4;
                ipv4_forward(260);
            }
            else{
                // modify status register
                meta.reg_status = read_reg_status.execute(meta.register_index);

                // check if register array is empty
                if (meta.reg_status == 0){ // we do not yet know this flow
                    meta.is_first = 1;
                    update_flow_ID.execute(meta.register_index);
                    meta.pkt_count = read_pkt_count.execute(meta.register_index);
                    meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                    meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                    meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);
                    update_reg_time_occ.execute(meta.register_index);
                    ipv4_forward(260);
                }
```

Manage recirculated packets and initialize feature registers

Flowrest: Practical stateful flow-level inference


```
apply {
        // compute the current time
        meta.now_timestamp = (bit<32>)(ig_prsr_md.global_tstamp[47:20]);  //msec

        //compute flow_ID and hash index
        get_flow_ID(meta.hdr_srcport, meta.hdr_dstport);
        get_register_index(meta.hdr_srcport, meta.hdr_dstport);
        flow_action_table.apply();

        if (meta.f_action != 0) {
            // Recirculated flow because of timeout collision
            if (hdr.recirc.isValid()){
                meta.is_first = 1;
                meta.reg_status = read_reg_status.execute(meta.register_index);
                update_flow_ID.execute(meta.register_index);
                meta.pkt_count = read_pkt_count.execute(meta.register_index);
                meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);

                update_reg_time_occ.execute(meta.register_index);
                // Invalidate the recirculation header
                hdr.recirc.setInvalid();
                hdr.ethernet.ether_type = TYPE_IPV4;
                ipv4_forward(260);
            }
            else{
                // modify status register
                meta.reg_status = read_reg_status.execute(meta.register_index);

                // check if register array is empty
                if (meta.reg_status == 0){ // we do not yet know this flow
                    meta.is_first = 1;
                    update_flow_ID.execute(meta.register_index);
                    meta.pkt_count = read_pkt_count.execute(meta.register_index);
                    meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                    meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                    meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);
                    update_reg_time_occ.execute(meta.register_index);
                    ipv4_forward(260);
                }
```

Manage first flow packets

Flowrest: Practical stateful flow-level inference


```
apply {
        // compute the current time
        meta.now_timestamp = (bit<32>)(ig_prsr_md.global_tstamp[47:20]);  //msec

        //compute flow_ID and hash index
        get_flow_ID(meta.hdr_srcport, meta.hdr_dstport);
        get_register_index(meta.hdr_srcport, meta.hdr_dstport);
        flow_action_table.apply();

        if (meta.f_action != 0) {
            // Recirculated flow because of timeout collision
            if (hdr.recirc.isValid()){
                meta.is_first = 1;
                meta.reg_status = read_reg_status.execute(meta.register_index);
                update_flow_ID.execute(meta.register_index);
                meta.pkt_count = read_pkt_count.execute(meta.register_index);
                meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);

                update_reg_time_occ.execute(meta.register_index);
                // Invalidate the recirculation header
                hdr.recirc.setInvalid();
                hdr.ethernet.ether_type = TYPE_IPV4;
                ipv4_forward(260);
            }
            else{
                // modify status register
                meta.reg_status = read_reg_status.execute(meta.register_index);

                // check if register array is empty
                if (meta.reg_status == 0){ // we do not yet know this flow
                    meta.is_first = 1;
                    update_flow_ID.execute(meta.register_index);
                    meta.pkt_count = read_pkt_count.execute(meta.register_index);
                    meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                    meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                    meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);
                    update_reg_time_occ.execute(meta.register_index);
                    ipv4_forward(260);
                }
```

```
else { // not the first packet - get flow_ID from register
    bit<32> tmp_flow_ID;
    tmp_flow_ID = read_only_flow_ID.execute(meta.register_index);
    if(meta.flow_ID != tmp_flow_ID){ // hash collision
        meta.age_value = read_reg_time_occ.execute(meta.register_index);
        if (meta.age_value < timeout_threshold){
            meta.final_class = 255;
            ipv4_forward(260);
        }
        else{
            // meta.digest_info = 127;
            meta.final_class = 127;
            recirculate(68);
        }

        ig_dprsr_md.digest_type = 1;        // activating the digest for statistics
    }
    else { // not first packet and not hash collision
        //read and update packet count
        meta.is_first = 0;
        meta.pkt_count = read_pkt_count.execute(meta.register_index);
        //read and update feature registers
        meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
        meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
        meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);

        update_reg_time_occ.execute(meta.register_index);

        // check if # of packets requirement is met
        if(meta.pkt_count == 3){
            // apply feature tables to assign codes
            table_feature0.apply();
            table_feature1.apply();
            table_feature2.apply();
            table_feature3.apply();
            table_feature4.apply();

            // apply code tables to assign labels
            code_table0.apply();
            code_table1.apply();
            code_table2.apply();

            voting_table.apply();
```

Manage has collisions and timed-out flows

Flowrest: Practical stateful flow-level inference


```
apply {
        // compute the current time
        meta.now_timestamp = (bit<32>)(ig_prsr_md.global_tstamp[47:20]);  //msec

        //compute flow_ID and hash index
        get_flow_ID(meta.hdr_srcport, meta.hdr_dstport);
        get_register_index(meta.hdr_srcport, meta.hdr_dstport);
        flow_action_table.apply();

        if (meta.f_action != 0) {
            // Recirculated flow because of timeout collision
            if (hdr.recirc.isValid()){
                meta.is_first = 1;
                meta.reg_status = read_reg_status.execute(meta.register_index);
                update_flow_ID.execute(meta.register_index);
                meta.pkt_count = read_pkt_count.execute(meta.register_index);
                meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);

                update_reg_time_occ.execute(meta.register_index);
                // Invalidate the recirculation header
                hdr.recirc.setInvalid();
                hdr.ethernet.ether_type = TYPE_IPV4;
                ipv4_forward(260);
            }
            else{
                // modify status register
                meta.reg_status = read_reg_status.execute(meta.register_index);

                // check if register array is empty
                if (meta.reg_status == 0){ // we do not yet know this flow
                    meta.is_first = 1;
                    update_flow_ID.execute(meta.register_index);
                    meta.pkt_count = read_pkt_count.execute(meta.register_index);
                    meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                    meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                    meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);
                    update_reg_time_occ.execute(meta.register_index);
                    ipv4_forward(260);
                }
```

```
                else { // not the first packet - get flow_ID from register
                    bit<32> tmp_flow_ID;
                    tmp_flow_ID = read_only_flow_ID.execute(meta.register_index);
                    if(meta.flow_ID != tmp_flow_ID){ // hash collision
                        meta.age_value = read_reg_time_occ.execute(meta.register_index);
                        if (meta.age_value < timeout_threshold){
                            meta.final_class = 255;
                            ipv4_forward(260);
                        }
                        else{
                            // meta.digest_info = 127;
                            meta.final_class = 127;
                            recirculate(68);
                        }

                        ig_dprsr_md.digest_type = 1;        // activating the digest for statistics

                    }
                    else { // not first packet and not hash collision
                        //read and update packet count
                        meta.is_first = 0;
                        meta.pkt_count = read_pkt_count.execute(meta.register_index);
                        //read and update feature registers
                        meta.pkt_len_total = read_pkt_len_total.execute(meta.register_index);
                        meta.pkt_len_max = read_pkt_len_max.execute(meta.register_index);
                        meta.ack_flag_count = read_ack_flag_count.execute(meta.register_index);

                        update_reg_time_occ.execute(meta.register_index);

                        // check if # of packets requirement is met
                        if(meta.pkt_count == 3){
                            // apply feature tables to assign codes
                            table_feature0.apply();
                            table_feature1.apply();
                            table_feature2.apply();
                            table_feature3.apply();
                            table_feature4.apply();

                            // apply code tables to assign labels
                            code_table0.apply();
                            code_table1.apply();
                            code_table2.apply();

                            voting_table.apply();
```

Read features and classify flows

Flowrest: Practical stateful flow-level inference

Evaluation

Use cases

- Intrusion detection based on the CICIDS 2017 dataset – 2 classes
- IoT device identification based on the UNSW IoT traces – 26 classes
- More in the paper

Benchmarks

- Packet-level (PL): Planter [1], Mousika [2], Soter [3]
- Flow-level (FL): pForest [4]
- Hybrid (PL+FL): NetBeacon [5]

[1] C. Zheng and N. Zilberman. Planter: Seeding trees within switches. In SIGCOMM Poster Session, 2021

[2] G. Xie et al. Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation. In IEEE INFOCOM, 2022

[3] G. Xie et al. Soter: Deep learning enhanced in-network attack detection based on programmable switches. In SRDS, 2022

[4] Busse-Grawitz et al. pForest: In-Network Inference with Random Forests. In Arxiv, 2019.

[5] G. Zhou et al. An efficient design of intelligent network data plane. In USENIX Security, 2023.

Flowrest: Practical stateful flow-level inference

Results – Flowrest vs stateless solutions

Dataset	Average	Metric	Planter	Mousika	Soter	Flowrest
CICIDS	Macro	Precision	94.448%	87.920%	94.446%	**99.785%**
		Recall	92.900%	78.359%	92.906%	**98.682%**
		F1-Score	93.625%	81.231%	93.628%	**99.231%**
	Weighted	Precision	94.712%	86.668%	94.713%	**99.700%**
		Recall	94.734%	86.009%	94.74%	**98.556%**
		F1-Score	94.688%	85.015%	94.690%	**99.124%**
UNSW	Macro	Precision	54.822%	67.882%	53.608%	**72.839%**
		Recall	57.523%	80.543%	55.677%	**81.760%**
		F1-Score	48.502%	69.103%	47.498%	**72.277%**
	Weighted	Precision	78.597%	90.166%	78.329%	**91.538%**
		Recall	73.906%	88.285%	72.208%	**89.165%**
		F1-Score	73.055%	88.572%	73.084%	**89.733%**

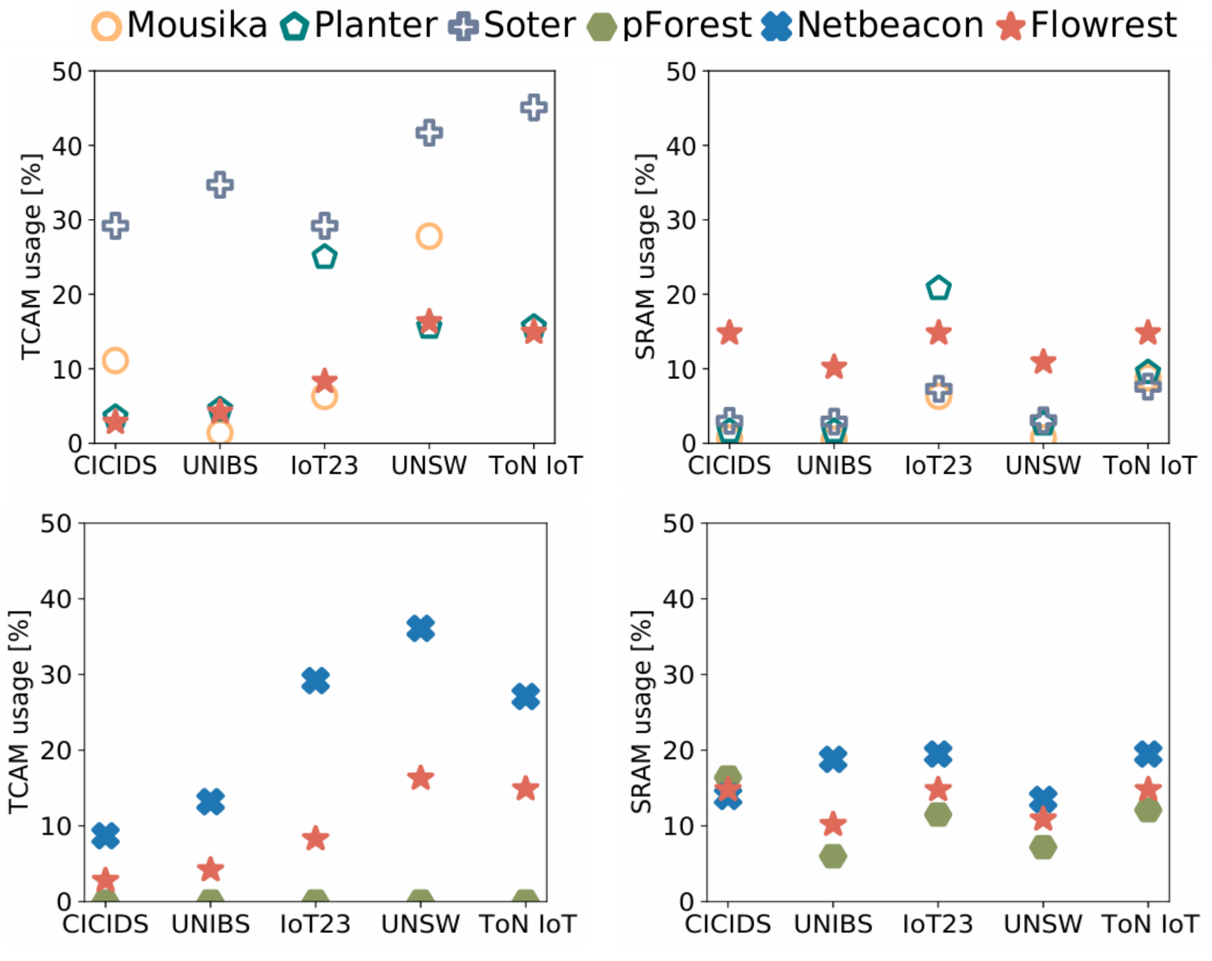
Flowrest: Practical stateful flow-level inference

Results – Flowrest vs stateful solutions

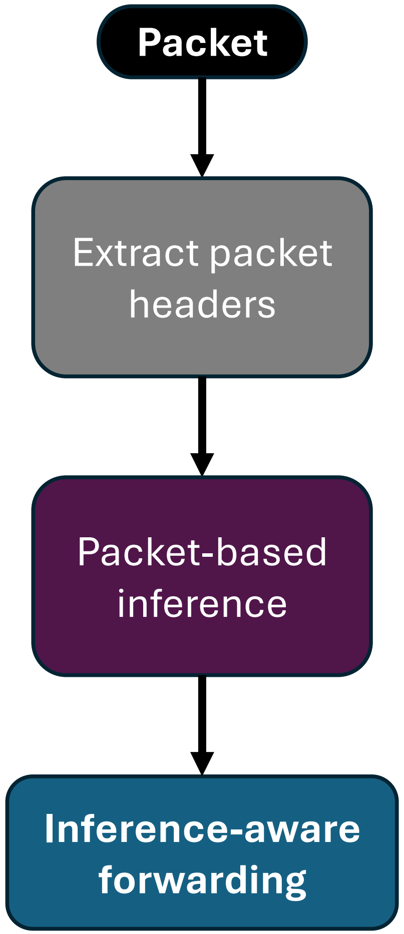
Dataset	Average	Metric	pForest	NetBeacon	Flowrest
CICIDS	Macro	Precision	99.778%	98.251%	**99.785%**
		Recall	98.690%	**98.918%**	98.682%
		F1-Score	**99.231%**	98.576%	**99.231%**
	Weighted	Precision	99.697%	98.816%	**99.700%**
		Recall	98.556%	**98.793%**	98.556%
		F1-Score	99.123%	98.798%	**99.124%**
UNSW	Macro	Precision	14.183%	56.256%	**72.839%**
		Recall	18.412%	66.089%	**81.760%**
		F1-Score	15.200%	53.284%	**72.277%**
	Weighted	Precision	41.582%	81.261%	**91.538%**
		Recall	48.407%	73.394%	**89.165%**
		F1-Score	43.034%	75.470%	**89.733%**

Flowrest: Practical stateful flow-level inference

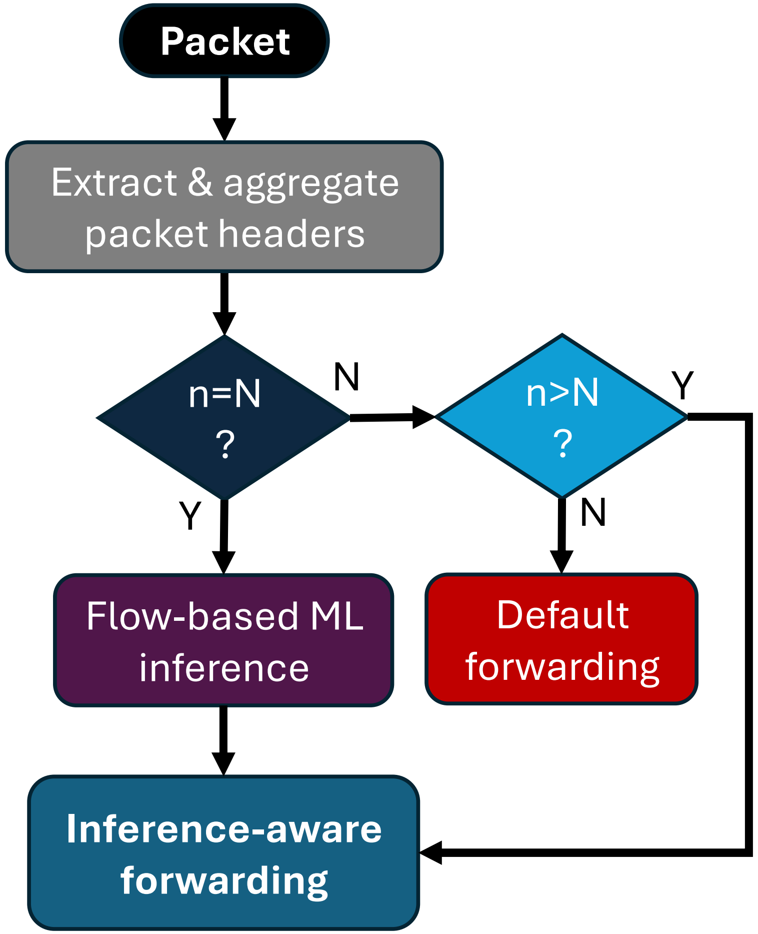
Results – resource usage



In-network ML: from stateless to hybrid approaches



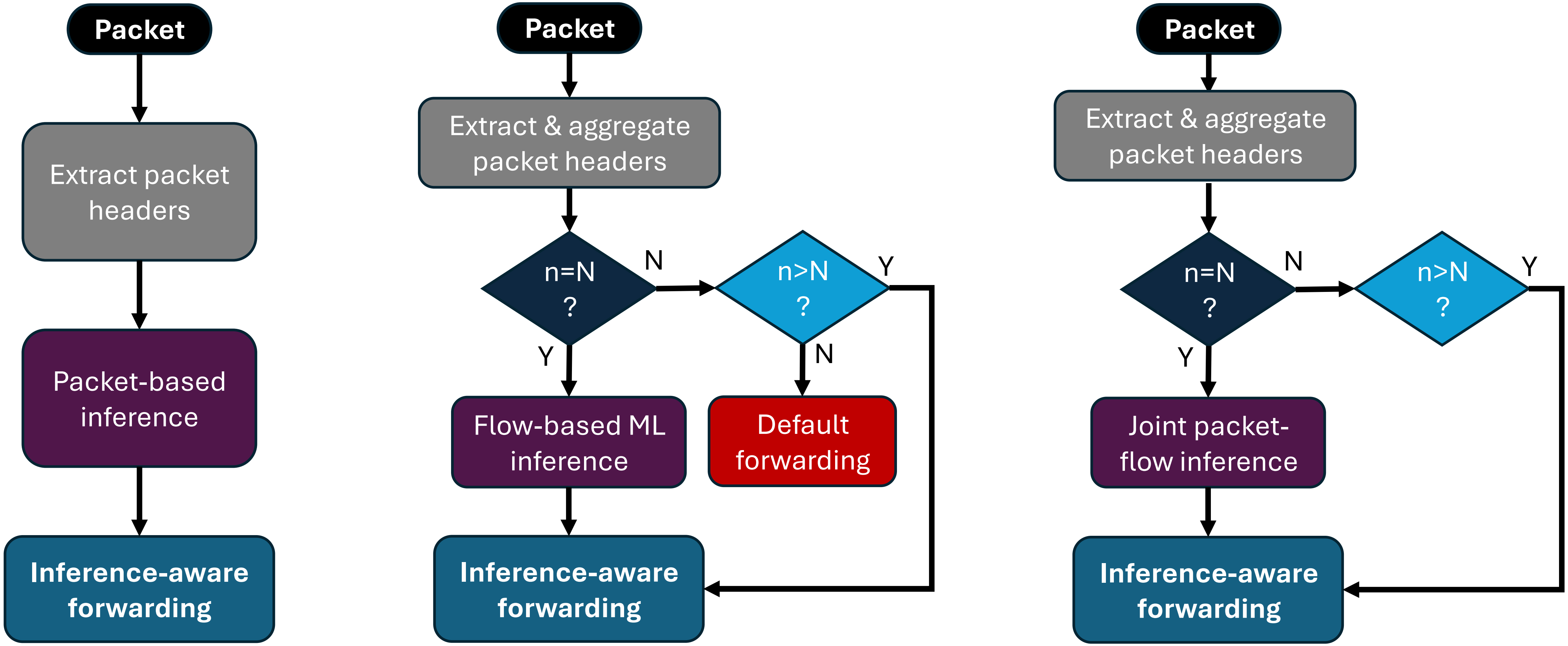
Per-packet (stateless)



Per-flow (stateful)

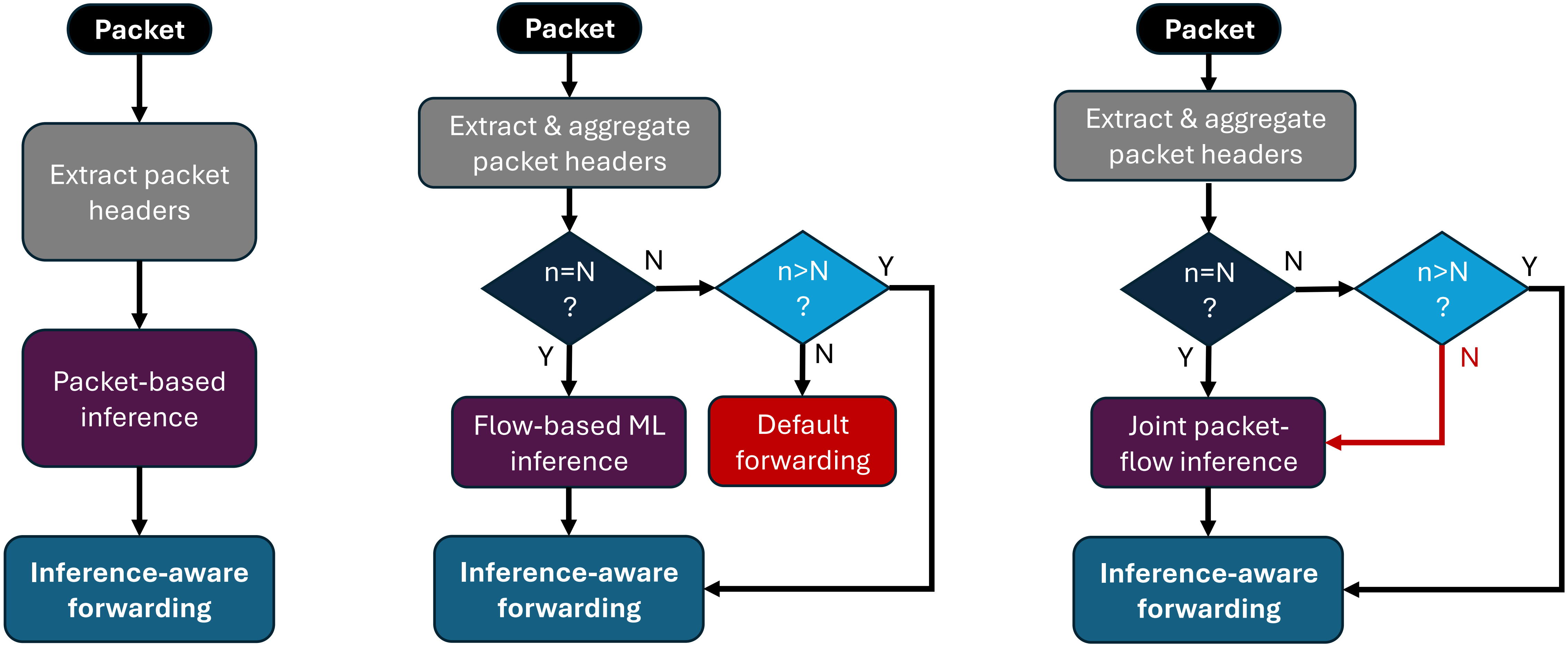
Joint packet-flow (hybrid)

In-network ML: from stateless to hybrid approaches



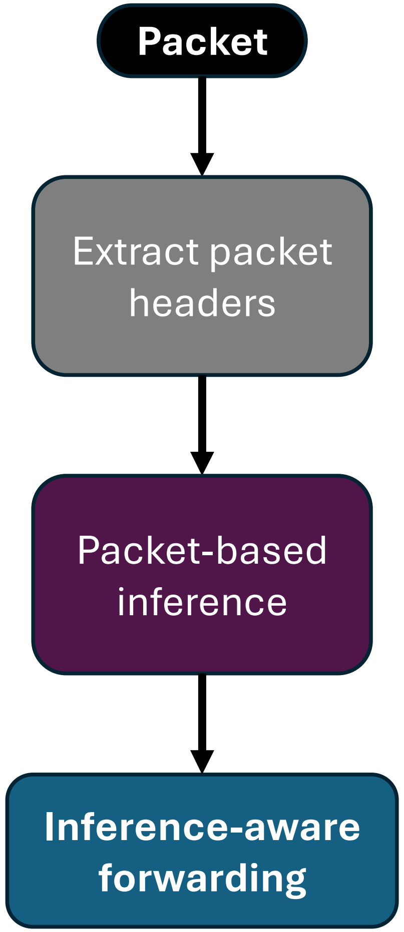
Per-packet (stateless) **Per-flow (stateful)** **Joint packet-flow (hybrid)**

In-network ML: from stateless to hybrid approaches

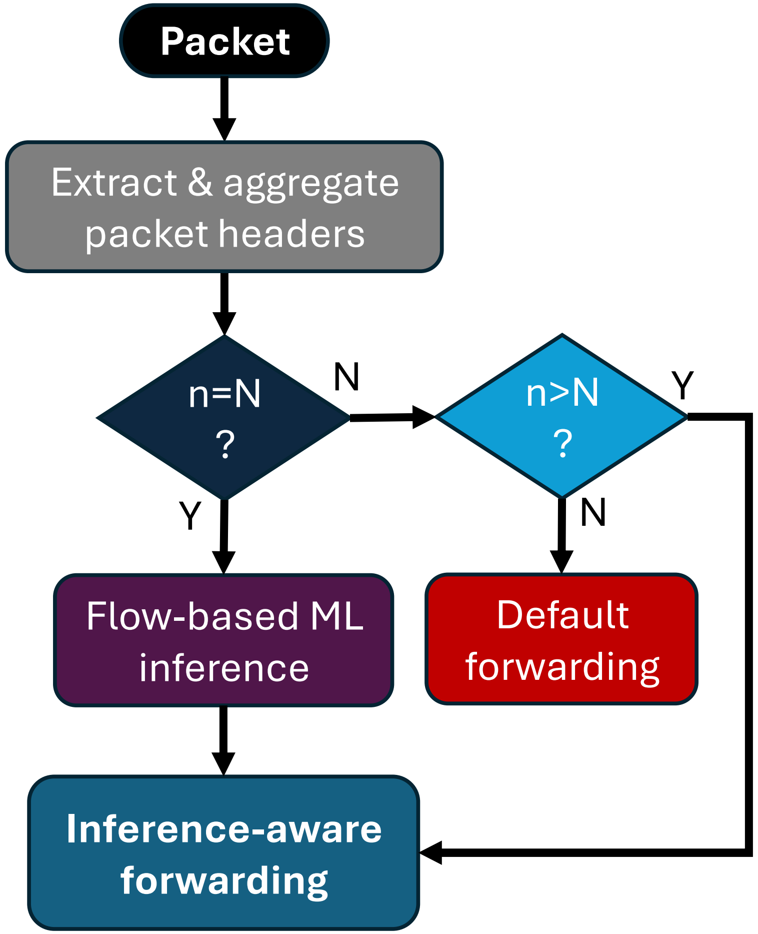


Per-packet (stateless) **Per-flow (stateful)** **Joint packet-flow (hybrid)**

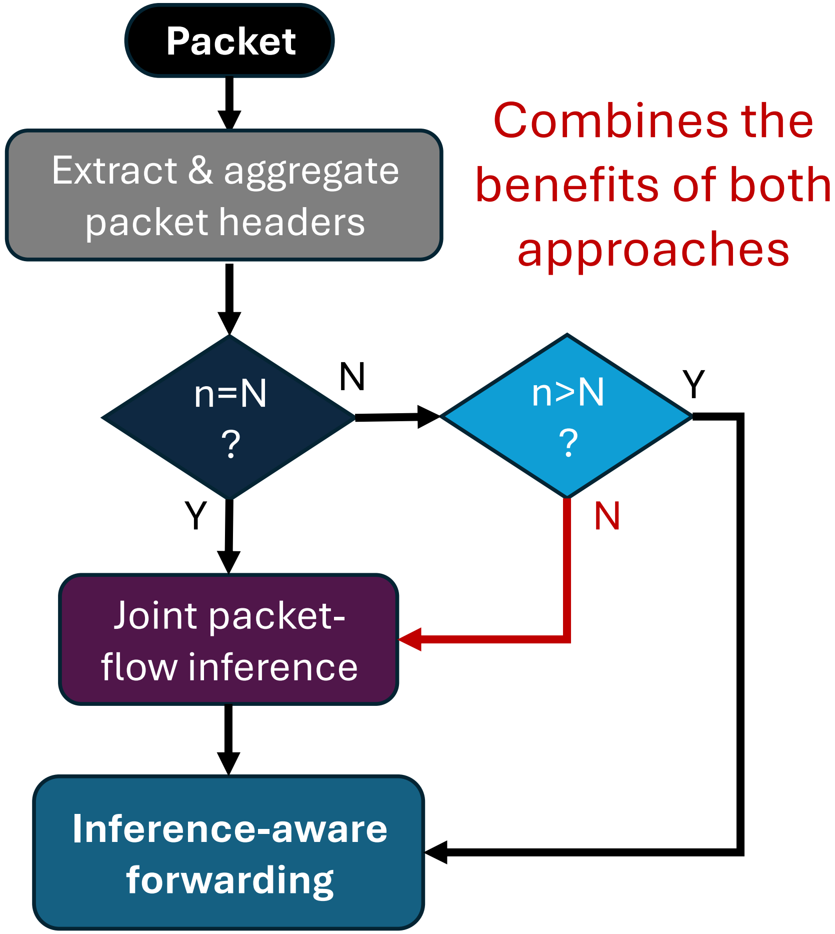
In-network ML: from stateless to hybrid approaches



Per-packet (stateless)



Per-flow (stateful)



Joint packet-flow (hybrid)

Jewel: Hybrid packet-level and flow-level inference

Motivation

Stateless Approaches

- Relatively lower accuracy in complex scenarios
- Cannot use rich FL features

Stateful Approaches

- Leave early packets unclassified when computing flow features
    - *Number of early packets could vary from **2 to 50** packets*
    - *Could be up to between **67.67%** and **98.04%** of the total flow length, respectively*

Jewel: Hybrid packet-level and flow-level inference

Motivation

Stateless Approaches

- Relatively lower accuracy in complex scenarios
- Cannot use rich FL features

Stateful Approaches

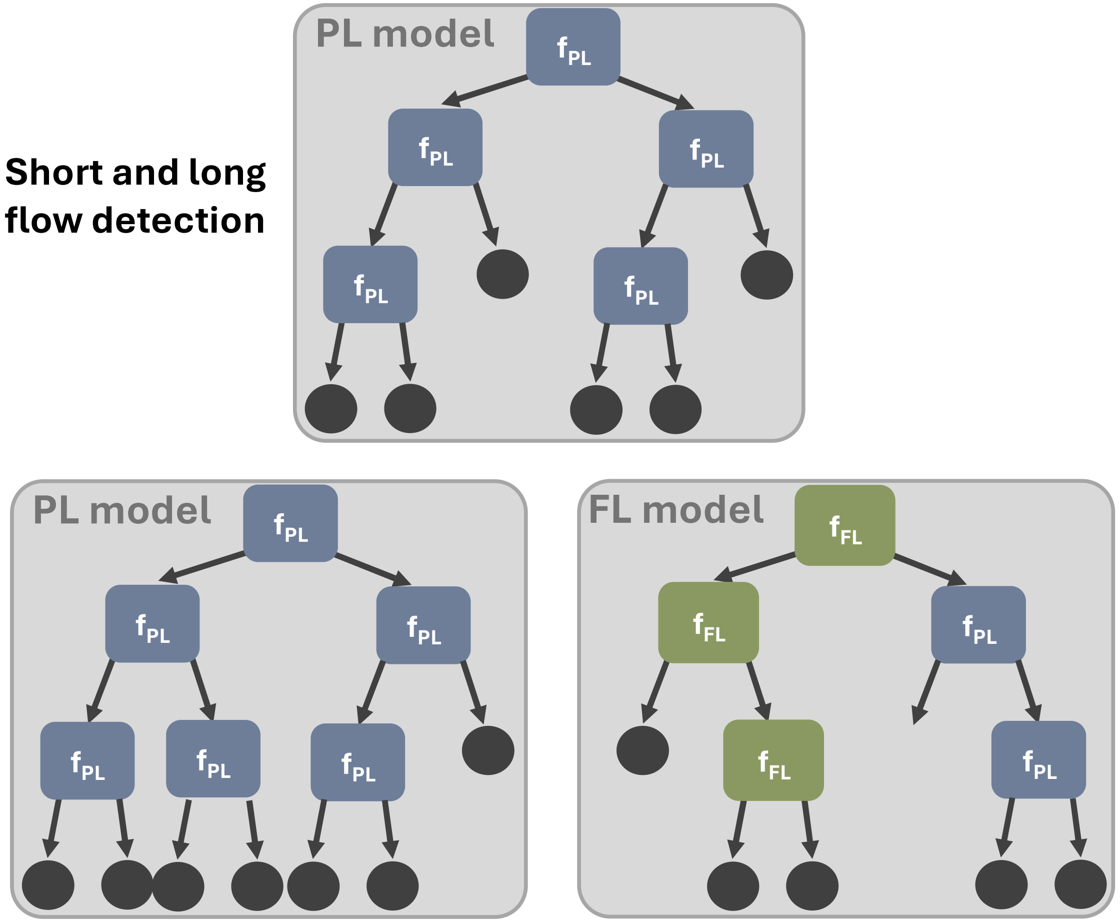
- Leave early packets unclassified when computing flow features
  - *Number of early packets could vary from **2 to 50** packets*
  - *Could be up to between **67.67%** and **98.04%** of the total flow length, respectively*

Hybrid stateless + stateful inference offers the best of both worlds

Jewel: Hybrid packet-level and flow-level inference

Prior approach to joint PL+FL inference, e.g. NetBeacon [1]

Short and long flow detection



Short flows
Early Packets of Long Flows

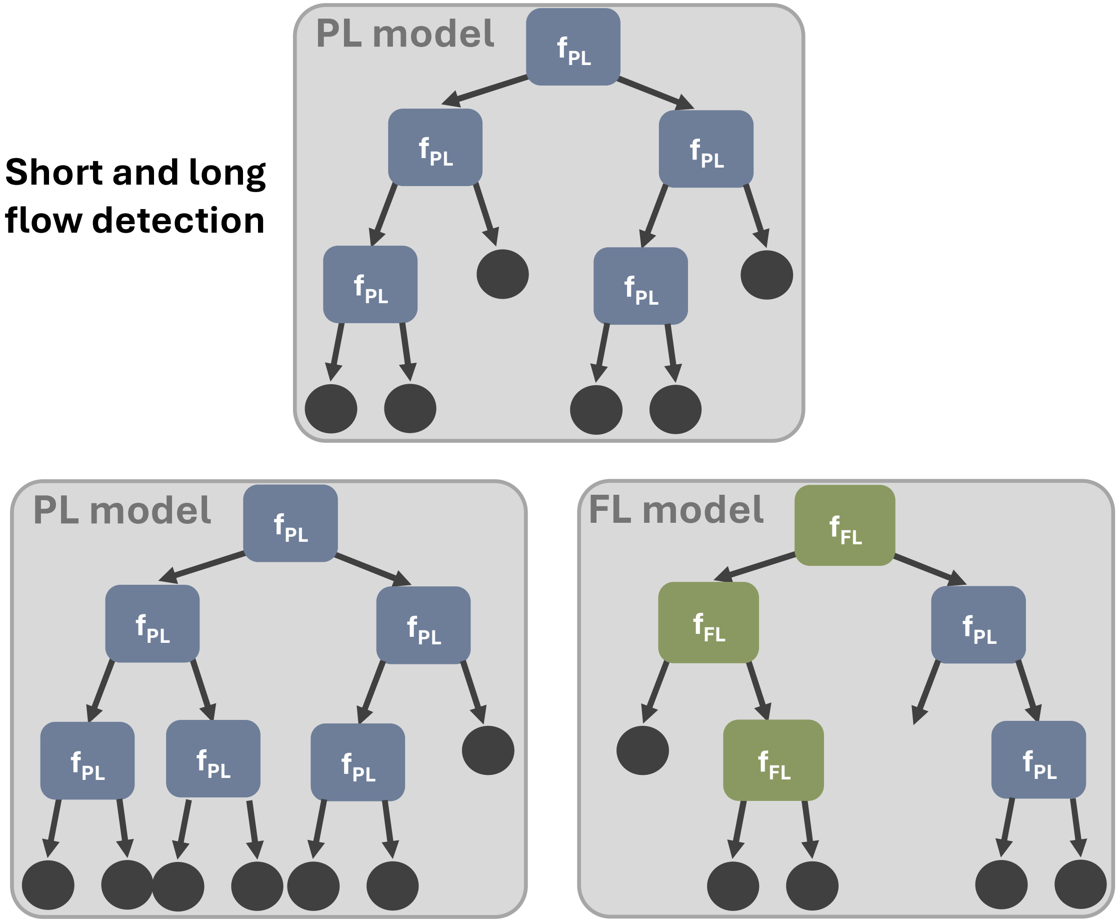
Long Flows

[1] G. Zhou et al. An efficient design of intelligent network data plane. In USENIX Security, 2023.

Jewel: Hybrid packet-level and flow-level inference

Prior approach to joint PL+FL inference, e.g. NetBeacon [1]

Short and long flow detection



Short flows
Early Packets of Long Flows

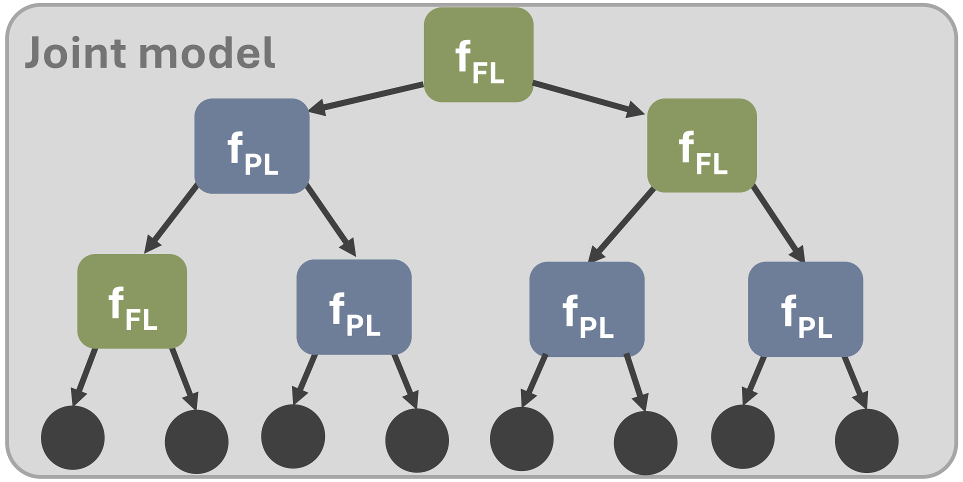
Long Flows

Increased switch memory consumption

[1] G. Zhou et al. An efficient design of intelligent network data plane. In USENIX Security, 2023.

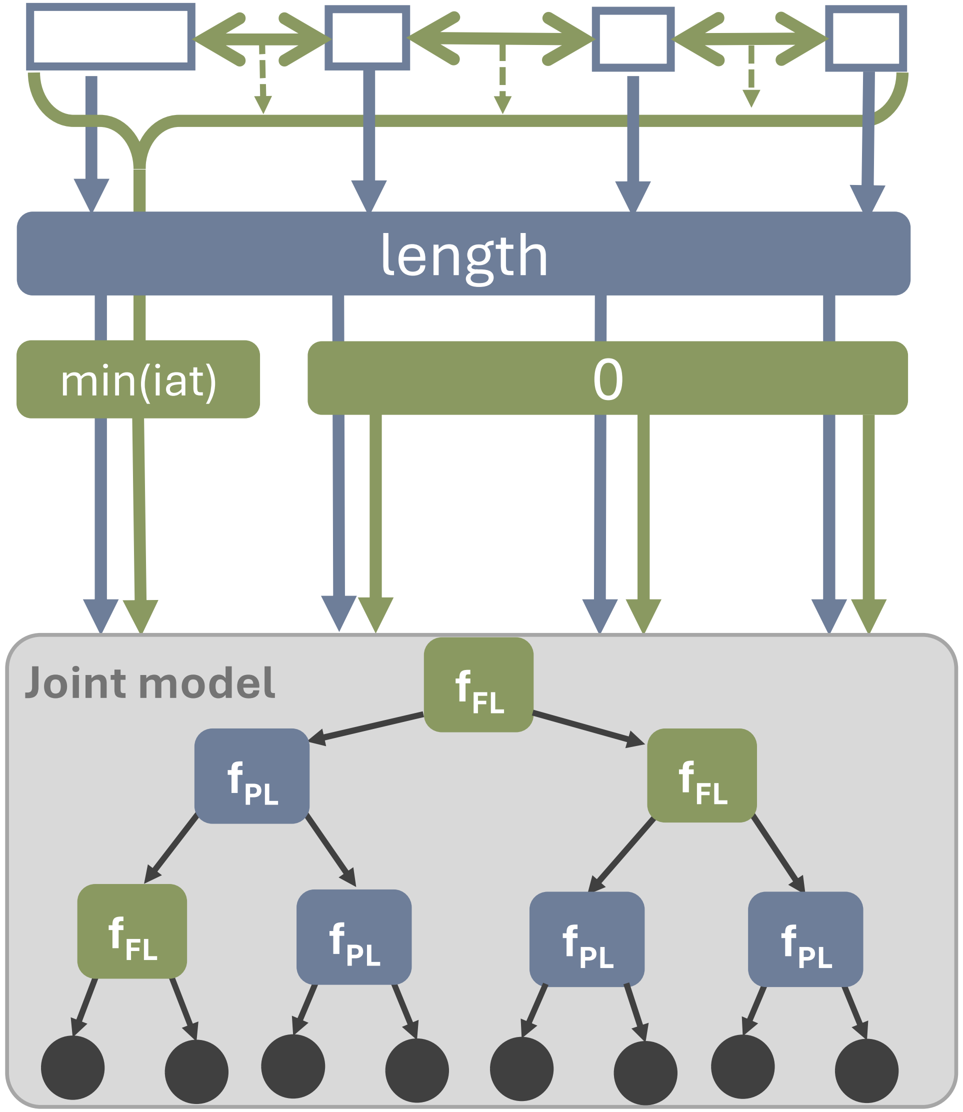
Jewel: Hybrid packet-level and flow-level inference

Our approach: single fully joint PL+FL model



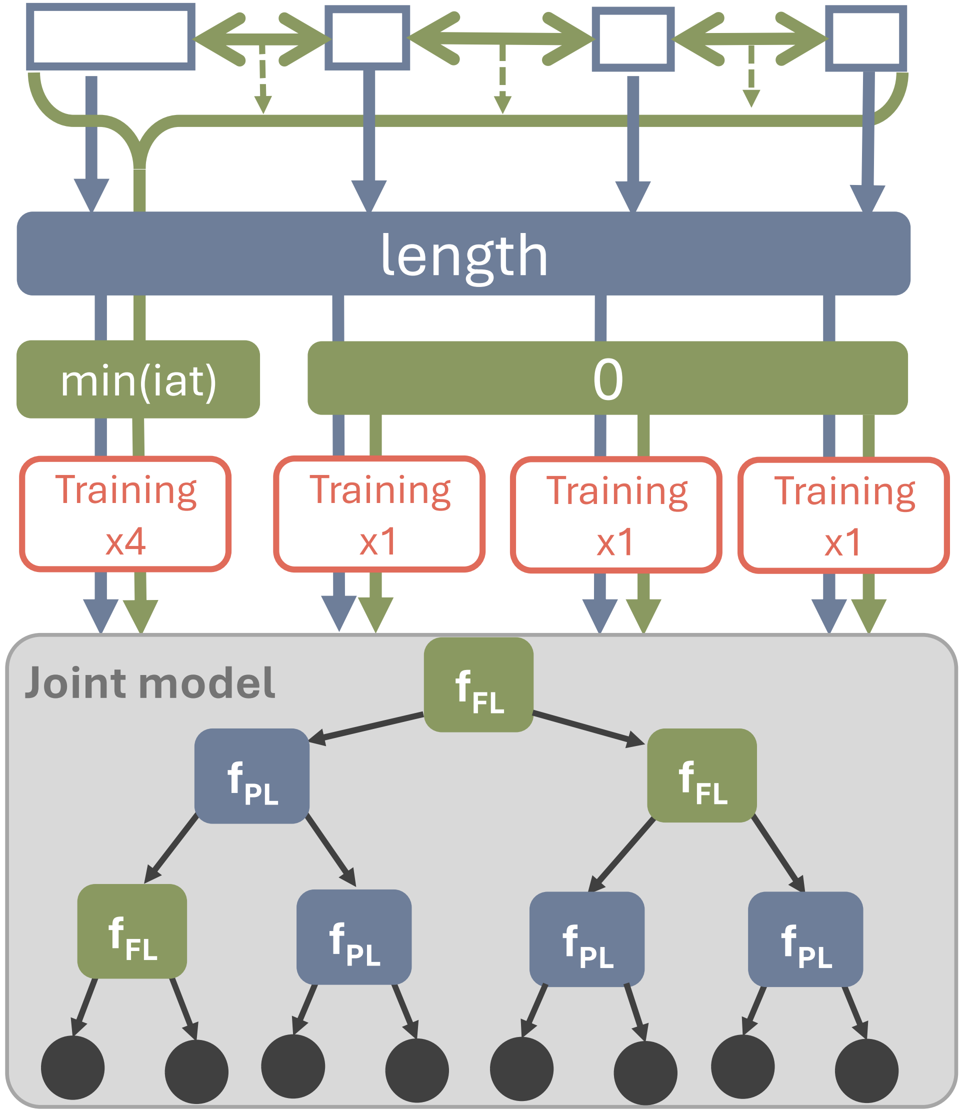
Jewel: Hybrid packet-level and flow-level inference

Our approach: single fully joint PL+FL model



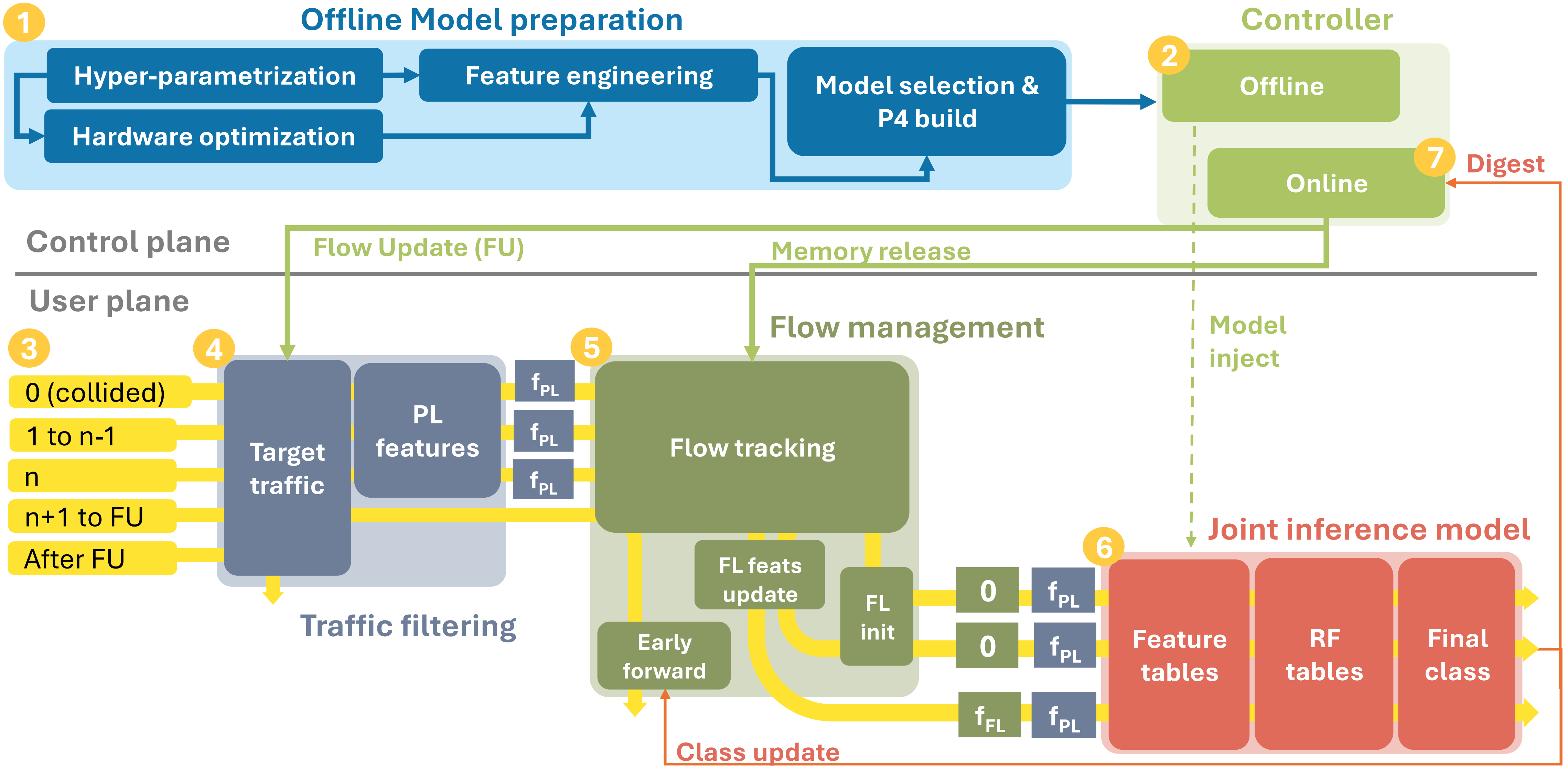
Jewel: Hybrid packet-level and flow-level inference

Our approach: single fully joint PL+FL model



Jewel: Hybrid packet-level and flow-level inference

System overview



Jewel: Hybrid packet-level and flow-level inference

Evaluation settings

Testbed

- Switch: Edgecore switch with an Intel Tofino BFN-T10-032Q chipset
- Servers: 2 DELL servers with AMD EPYC 24-core at 2.8GHz

Use cases

- Intrusion detection: *CIC-IDS 2017 dataset* (binary)
- IoT device classification: *UNSW IoT traces* (multiclass)
- IoT bot classification: *IoT-23 dataset* (multiclass)
- IoT cyberattack classification: *ToN-IoT dataset* (multiclass)

Benchmarks

- Packet-level (PL): Mousika [1], Planter [2]
- Flow-level (FL): Flowrest [3]
- Hybrid (PL+FL): NetBeacon [4]

[1] G. Xie et al. Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation. In IEEE INFOCOM, 2022
[2] C. Zheng and N. Zilberman. Planter: Seeding trees within switches. In SIGCOMM Poster Session, 2021
[3] A. Akem et al. Flowrest: Practical flow-level inference in programmable switches with random forests. In *IEEE INFOCOM,* 2023.
[4] G. Zhou et al. An efficient design of intelligent network data plane. In USENIX Security, 2023.

Jewel: Hybrid packet-level and flow-level inference

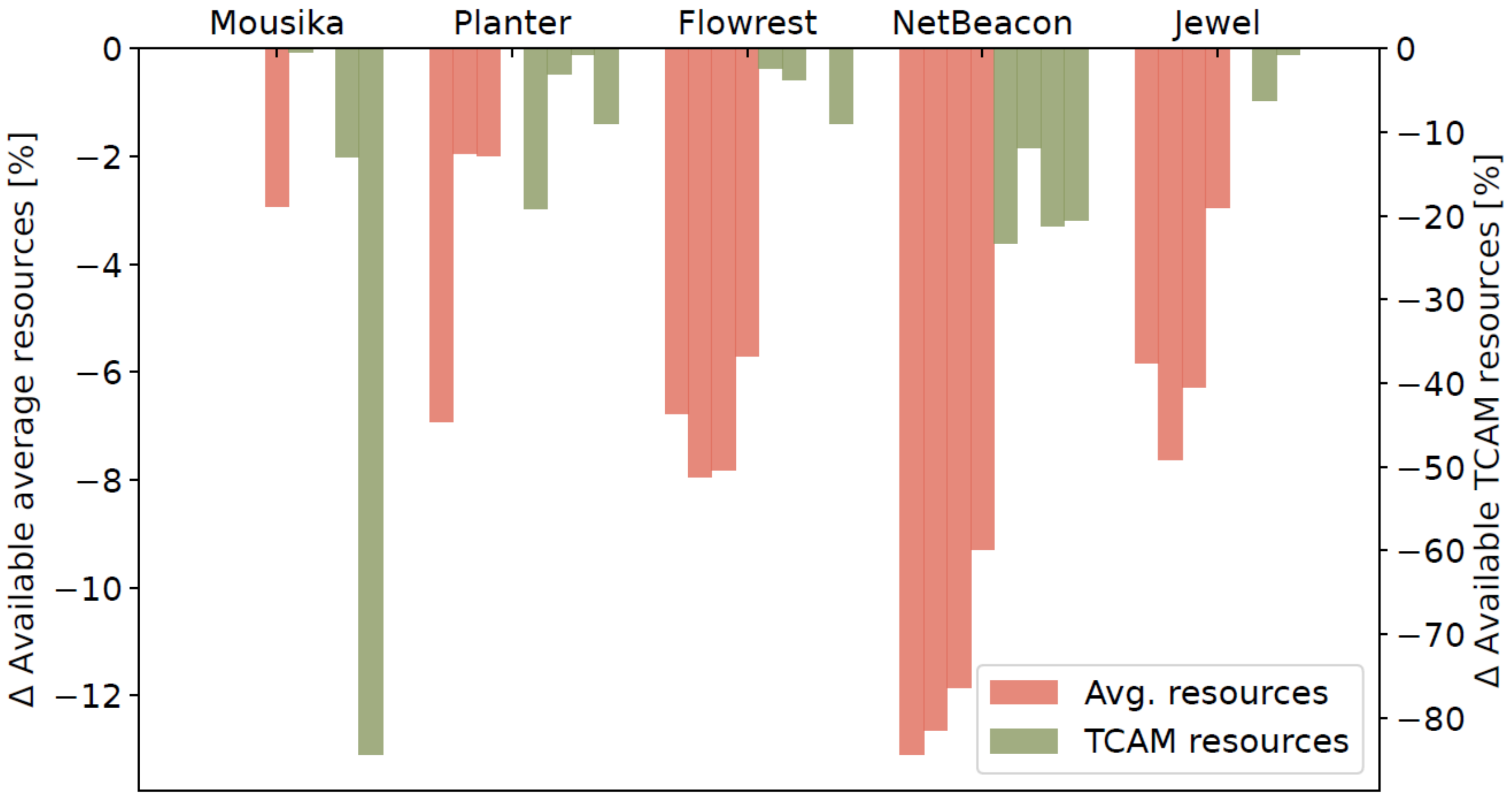
Results – model accuracy

	Mousika	Planter	Flowrest	NetBeacon	Jewel
UNIBS	90.351%	91.560%	96.398%	94.570%	**98.354%**
UNSW	82.003%	79.853%	80.691%	78.594%	**87.317%**
ToN-IoT	27.554%	70.496%	73.461%	70.063%	**75.703%**
IoT23	86.054%	88.147%	82.857%	86.076%	**91.314%**

Accuracy gains in the range **2.0% – 5.3%** over the next best

Jewel: Hybrid packet-level and flow-level inference

Results – resource usage



Jewel achieves high accuracy while not increasing resource usage

In-network inference design trade-offs

Approach	Pros	Cons
Stateless	Early decisions, classifies all packets, low memory footprint	Lower accuracy in complex tasks
Stateful	Higher accuracy	Early packets missed, higher memory footprint
Hybrid	Higher accuracy, classifies all packets	Slightly higher complexity, higher memory footprint

Ongoing and future work

- Diversifying inference targets:
  - *Intel Infrastructure Processing Unit (IPU)*
  - *NVIDIA BlueField-2 Data Processing Unit (DPU)*
- Distributed inference in heterogeneous settings:
  - *Scenarios with multiple models/targets in coordination*
  - *Real-time model drift detection and online learning*
- Use cases of in-network inference:
  - *Healthcare monitoring*
  - *KV cache acceleration, etc*

Conclusions

- In-network ML enables network intelligence at high speed
- Several solutions have been proposed for stateless, stateful, and hybrid inference
- These solutions lay the foundation for many in-network inference use cases that will contribute to the automation of network management
- Future work will pursue further steps towards a more seamless integration of ML into networked systems


"And just as self-driving cars have been "just a few years away" for more than a few years, I suspect that automating the management of physical networks is going to remain out of reach (for most of us) for a while longer."
– Bruce Davie, Systems Approach LLC
The Register, June '24

Acknowledgements

Collaborators

- Prof. Marco Fiore, Research Professor, IMDEA Networks Institute, Madrid, Spain
- Dr. Michele Gucciardo, Research Engineer, NEC Laboratories Europe, Madrid, Spain
- Beyza Bütün, PhD Student, IMDEA Networks Institute, Madrid, Spain

Relevant Publications

[1] A. Akem, B. Bütün, M. Gucciardo, M. Fiore. **Henna: Hierarchical machine learning inference in programmable switches**. In *NativeNI,* 2022.

[2] A. Akem, M. Gucciardo, M. Fiore. **Flowrest: Practical flow-level inference in programmable switches with random forests**. In *IEEE INFOCOM,* 2023.

[3] A. Akem, B. Bütün, M. Gucciardo, M. Fiore**. Jewel: Resource-efficient joint packet and flow level inference in programmable switches**. In *IEEE INFOCOM,* 2024.

[4] A. Akem, B. Bütün, M. Gucciardo, M. Fiore**. Practical and General-Purpose Flow-Level Inference With Random Forests in Programmable Switches**. In *IEEE/ACM Transactions on Networking,* 2025.

Thank you!

This work was supported by

Project **PCI2022-133013 (ECOMOME)**,
funded by MICIU/AEI/10.13039/501100011033 and
the European Union "NextGenerationEU"/PRTR.

The Horizon Europe programme of the European Union,
under grant agreement no. 101139270 **"ORIGAMI"**,
and under grant agreement no. 860239 **"BANYAN"** .

Questions?