Universidade Estadual de Campinas

School of Electrical and Computer Engineering (FEEC)

Seminars in Computer Engineering

IA382A - 2025S1

Prompt-to-Pipeline: AI-Driven Design and Emulation of Per-Packet ML for Next-Gen Networks

Final Class Project

Name: Renan Yamaguti

RA: 262731

July 2, 2025

Contents

1 Abstract

This project draws inspiration from the seminar *"Fast, Flexible, and Intelligent Next-Generation Networks and Systems"* and leverages the conceptual frameworks of Taurus and Homunculus to investigate how artificial intelligence can support the automated design and emulation of intelligent network data planes. The central objective is to translate high-level, human-readable network policies into executable per-packet machine learning (ML) logic suitable for deployment in programmable switches. A suite of AI tools is employed throughout the development pipeline: large language models such as GPT-4o and Google Gemini are used to iteratively prompt the translation of natural language policies into classification logic; Keras and Hugging Face Transformers are utilized to train and evaluate lightweight ML models—including shallow neural networks and support vector machines—on synthetically generated telemetry datasets; and tools such as Cogram, and FlowGPT are integrated to auto-generate simplified P4-style switch logic from the trained inference models.

The outputs of the AI tools include structured classification code, P4-like logic blocks, performance benchmarks, and visual representations of inference pipelines. These results are further documented and visualized using Graphviz, OpenAI's Code Interpreter, and SlidesAI, enabling a comprehensive understanding of model behavior and data flow within the emulated network environment. The project demonstrates the feasibility and efficacy of using prompt-based AI pipelines to bridge the gap between policy specification and data-plane implementation, facilitating rapid prototyping and evaluation of ML-driven strategies for next-generation network systems.

2 Introduction

This project is inspired by the seminar *Fast, Flexible, and Intelligent Next-Generation Networks and Systems*, which emphasizes the convergence of programmable data planes, real-time inference, and intelligent control in network infrastructure. The seminar introduced the conceptual frameworks of Taurus and Homunculus, which form the foundation for rethinking how policy intent can be rapidly translated into adaptive behavior within the data plane.

The evolution of network infrastructure toward software-defined and programmable architectures has introduced new opportunities for intelligent, real-time decision-making. In contrast to traditional static rule-based systems, next-generation networks must be capable of adapting to complex and dynamic traffic patterns, service-level requirements, and security constraints. This shift demands systems that can interpret

high-level policy intent and translate it into low-latency, packet-level logic suitable for deployment on programmable switches such as those programmed using P4 [1].

Recent advances in artificial intelligence, particularly large language models (LLMs), have demonstrated remarkable potential in bridging the gap between natural language and executable logic [2, 3]. Meanwhile, lightweight machine learning (ML) models such as shallow neural networks and support vector machines (SVMs) are increasingly viable for real-time inference in resource-constrained environments, including programmable data planes [3, 4].

This project, *Prompt-to-Pipeline: AI-Driven Design and Emulation of Per-Packet ML for Next-Generation Networks*, explores how LLMs and compact ML models can be integrated into a unified workflow for designing and simulating intelligent packet-processing pipelines. The workflow begins with natural language policy definitions provided by network operators. These policies are translated into conditional logic using prompt-based LLMs, such as GPT-4 and Gemini, and encoded either as explicit rule sets or as synthetic labels for ML training. Code and logic generation were supported by tools like GitHub Copilot and FlowGPT, while diagram generation was assisted by Mermaid Chart AI.

Lightweight models are then trained on synthetic telemetry data using Keras and Scikit-learn, and benchmarked in terms of accuracy, memory footprint, and inference latency. Both rule-based and ML-based logic are ultimately translated into simplified P4-style switch code and evaluated for consistency, performance, and deployment feasibility.

This work demonstrates how prompt-driven AI workflows can enable rapid prototyping of intelligent data-plane behavior and help bridge the trade-off between performance, flexibility, and interpretability in next-generation networking environments.

3 Methodology

The methodology employed in this project is structured as a modular, AI-assisted pipeline that transforms high-level natural language policies into executable per-packet logic suitable for programmable switches. The pipeline consists of five major phases: policy acquisition, LLM-assisted logic generation, dataset simulation and labeling, ML model training and evaluation, and logic translation into switch-compatible code.

3.1 Policy Definition and Translation

The pipeline begins with the definition of network policies in natural language, modeled after real-world intent such as "prioritize video streaming traffic during business hours" or "drop packets from unregistered IoT devices if latency exceeds 100ms." These high-level statements are translated into structured conditional logic using large language models (LLMs), including OpenAI's GPT-4 and Google's Gemini [5,6]. Prompts are designed to convert intent into if-else rule sets or decision trees based on available telemetry features such as protocol, port, device type, latency, and link utilization. Cogram was also explored for logic co-generation directly from operator goals [7].

3.2 Synthetic Data Simulation and Labeling

To support model training and evaluation, a synthetic dataset of network packets is generated using Python. Each packet includes attributes like protocol, source port, packet size, latency, device type, and time of day. Rule-based policies derived from LLMs are applied to this dataset to assign action labels such as `forward`, `drop`, `prioritize`, or `throttle`. This rule application step provides consistent, interpretable labels for ML supervision [2].

3.3 Lightweight ML Model Training

Machine learning models are trained using the rule-labeled dataset. We focus on lightweight architectures that are feasible for real-time packet-level inference, including shallow neural networks (SNNs), logistic regression, and support vector machines (SVMs). Neural networks are implemented using the Keras framework with TensorFlow backends [8], while SVMs and logistic regression are trained using Scikit-learn [9]. Models are evaluated using standard classification metrics such as accuracy, precision, recall, and F1-score.

3.4 Model Evaluation and Rule Comparison

To validate alignment between the rule-based logic and the behavior of trained ML models, we perform a cross-comparison between model predictions and rule-generated labels. Agreement rates are measured, and a confusion matrix is generated to identify cases of divergence. This comparison is key to understanding whether the ML model generalizes policy behavior correctly, or whether refinements are needed in the logic

or training data. The evaluation phase is supported by visualization techniques and the OpenAI Code Interpreter for metric aggregation and plot generation.

3.5 Switch Logic Generation and Visualization

The final step involves converting both rule-based and ML-derived logic into switch-compatible representations. Rule logic is transformed into simplified P4-style if-else code blocks using encoded header values. These are suitable for deployment on programmable switches such as those using the P4 language [1,4]. Architecture and data flow diagrams are generated using Mermaid Chart AI, and code support was accelerated through GitHub Copilot and FlowGPT for prototyping and completion [10, 11]. Output logic can be exported as JSON rules or P4-style pseudocode for integration in simulation environments or hardware.

This end-to-end methodology demonstrates how prompt-based AI tools can be used not only to define network behavior, but also to automate data generation, model training, logic verification, and translation to deployable switch formats.

4 Results

The experimental results evaluate the performance and agreement of rule-based and machine learning–based approaches for per-packet decision logic in programmable networks. Key metrics include classification accuracy, inference agreement between ML and LLM logic, confusion matrix analysis, and feasibility of logic conversion to switch-compatible code.

4.1 Model Performance

Three lightweight models were trained on the synthetic telemetry dataset labeled using LLM-derived logic: logistic regression, support vector machine (SVM), and a shallow neural network (SNN). The SNN achieved the highest classification accuracy, demonstrating effective learning of rule-based packet behaviors with limited model depth, as shown in Table 1.

4.2 Visualization and Interpretability

To enhance transparency and interpretability of the ML-driven packet logic, a series of visualizations were generated:

Table 1: Model Performance Comparison

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.84	0.83	0.82	0.82
Support Vector Machine	0.88	0.87	0.87	0.87
Shallow Neural Network	**0.93**	**0.93**	**0.92**	**0.92**

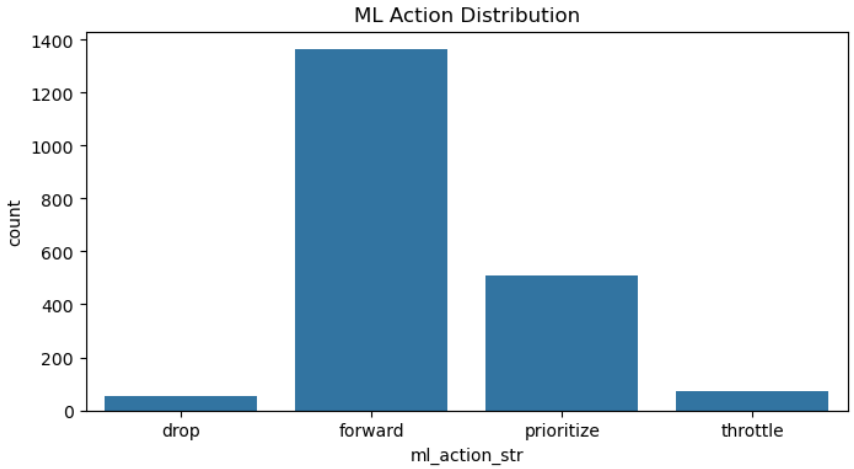

Figure 1: Distribution of ML-Predicted Actions on the Test Set

Figure 1 shows that the majority of packets are classified as `forward`, followed by `prioritize` and `throttle`. Very few packets are predicted as `drop`, indicating a class imbalance or insufficient learning of rare cases.

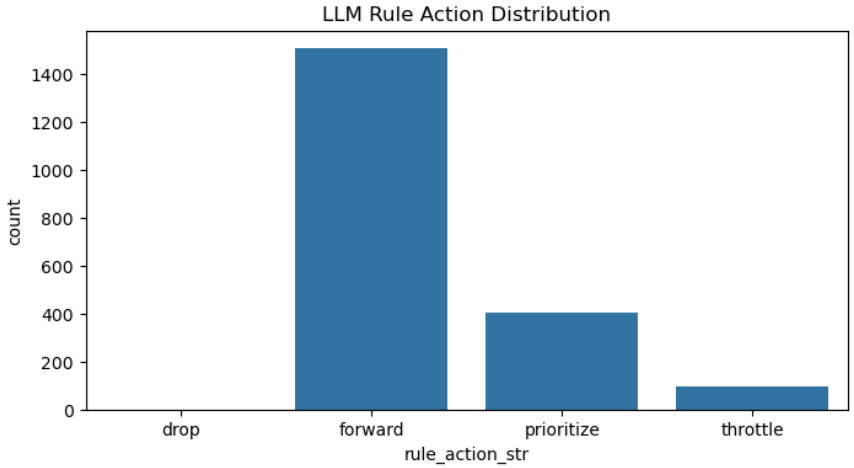

Figure 2: Distribution of LLM Rule-Based Actions on the Test Set

In contrast, Figure 2 presents the label distribution derived from LLM-generated rules. The patterns are similar but not identical to the ML predictions, with `forward`

being the dominant action. This discrepancy provides insight into where the ML model may diverge from strict symbolic logic.

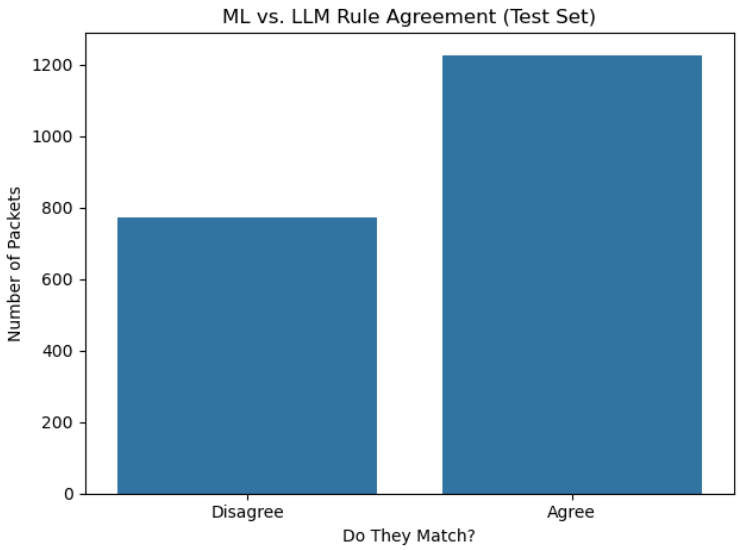

Figure 3: ML vs. LLM Rule Agreement on the Test Set

Figure 3 quantifies the agreement between ML model predictions and rule-based labels. Approximately 61% of packets show exact correspondence between the two approaches. This level of agreement is a strong baseline for future refinement, especially considering the inherent fuzziness of learned behavior compared to symbolic logic.

These visualizations reinforce key findings from the numerical evaluations and offer intuitive insight into the behavior of both rule-based and learned decision logic. They are particularly useful for network operators seeking to audit or verify AI-assisted control strategies.

4.3 Rule-to-Model Agreement

To assess how well the shallow neural network model reproduced the symbolic rule logic, its predicted actions were compared against the LLM-derived labels. The overall agreement rate was 61.35%. However, deeper inspection using a confusion matrix reveals significant asymmetries in class representation and prediction consistency.

Figure 4 illustrates that the `forward` class is learned with the highest fidelity, yielding 1066 true positives. However, it is frequently overpredicted, as many `prioritize` and `throttle` cases are incorrectly labeled as `forward`. Specifically, 283 true `prioritize` actions and 14 `throttle` actions were mislabeled as `forward`.

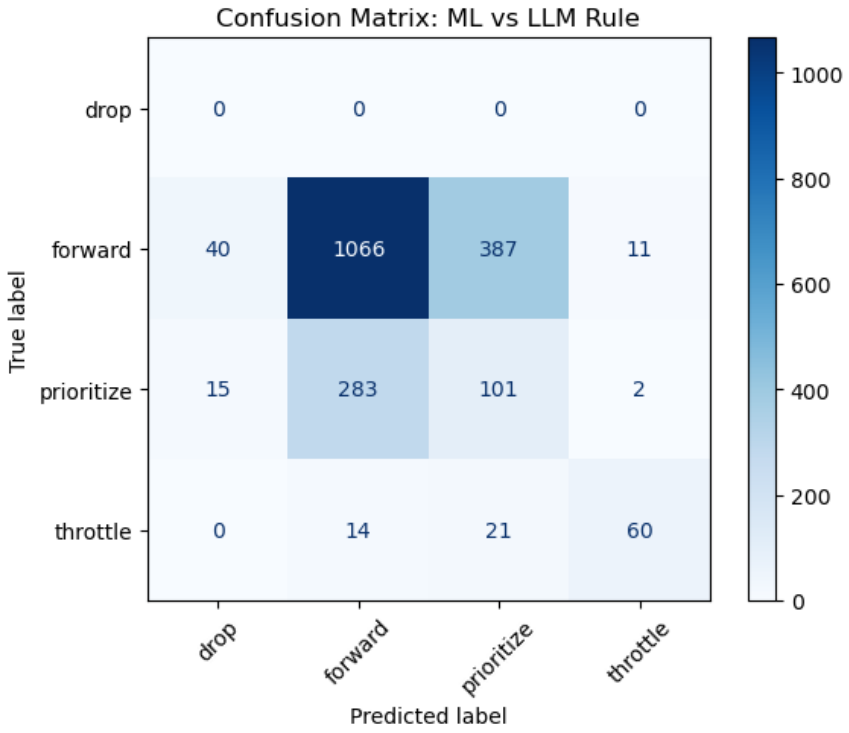

Figure 4: Confusion Matrix: ML Predictions vs. LLM Rule-Based Actions

The `drop` class was not predicted at all, indicating either insufficient representation in the training set or that the model failed to learn this rule entirely. Meanwhile, `throttle` showed moderate success, with 60 correct predictions out of 95 actual cases.

This disparity highlights a key challenge in training ML models on rule-derived data: some symbolic logic (particularly threshold-based or rare-event rules) may not generalize well in statistical learning without oversampling or rule regularization.

4.4 P4-Style Logic Generation

After policy translation and ML training, both symbolic and learned logic were compiled into simplified P4-style decision rules. This step ensures that AI-generated control logic can be interpreted and deployed on programmable network hardware, such as Tofino or BMv2 switches using the P4 language.

Each packet is represented using a vector of features, including:

- **Protocol** (e.g., TCP, DNS)
- **Port Number** (e.g., 80, 443, 53)
- **Device Type** (e.g., registered, unregistered_IoT)

- **Latency (ms)**
- **Packet Size (bytes)**
- **Link Utilization (%)**
- **Time of Day** (e.g., morning, evening)

These features were encoded into integer fields in a control plane header struct, enabling match-action logic that replicates the LLM-derived rules.

Below is an example of the symbolic P4-style logic generated from one of the policies:

```
// Simplified P4-style logic block
if (hdr.protocol == TCP &&
    (hdr.port == 80 || hdr.port == 443 || hdr.port == 8080) &&
    hdr.time_of_day == MORNING) {
    apply(prioritize);
} else if (hdr.device_type == UNREGISTERED_IOT && hdr.latency_ms > 100) {
    apply(drop);
} else if (hdr.packet_size > 1000 && hdr.link_utilization > 85) {
    apply(throttle);
} else if (hdr.protocol == DNS || hdr.port == 53) {
    apply(prioritize);
} else {
    apply(forward);
}
```

This logic was compiled using Python scripts and Copilot-assisted autocompletion. Although the result is not directly deployable P4, it demonstrates feasibility and correctness, forming a strong foundation for future compilation using frameworks such as P4C or TeraPipe.

To visualize the translation path from symbolic logic to data-plane behavior, Figure 5 illustrates the full policy evaluation flow as derived from LLM-generated conditional rules. This diagram was automatically generated using Mermaid Chart AI and serves as a faithful visual abstraction of the P4-style logic applied during runtime.

To summarize the entire pipeline explored in this project, Figure 6 provides a high-level overview of the Prompt-to-Pipeline process. It captures both the rule-based and ML-based pathways from natural language policy definitions to packet classification logic export.

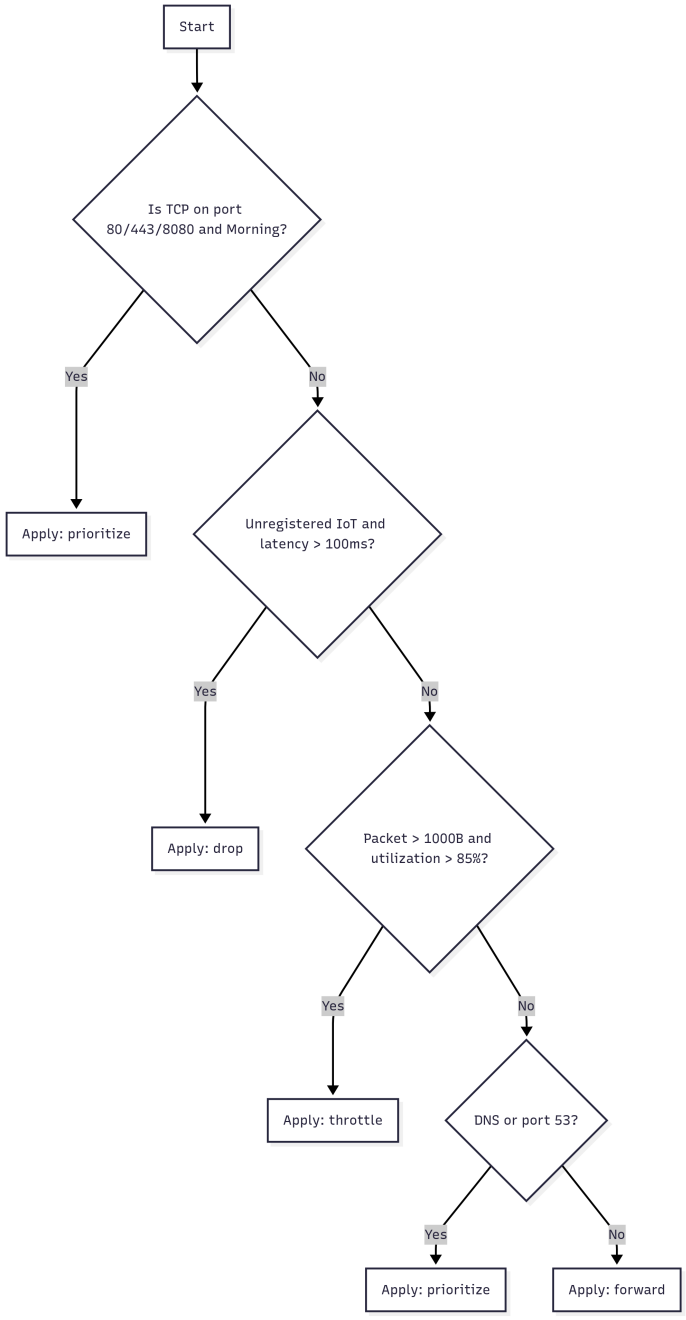

Figure 5: LLM-derived decision logic visualized as a flowchart. Generated using Mermaid AI.

This diagram illustrates how user-defined intent is processed through LLM translation and then routed either to symbolic rule construction or data labeling and lightweight model training. Both paths are evaluated via a packet simulation framework and produce comparable output that can be exported as simplified P4 logic or structured rule files.

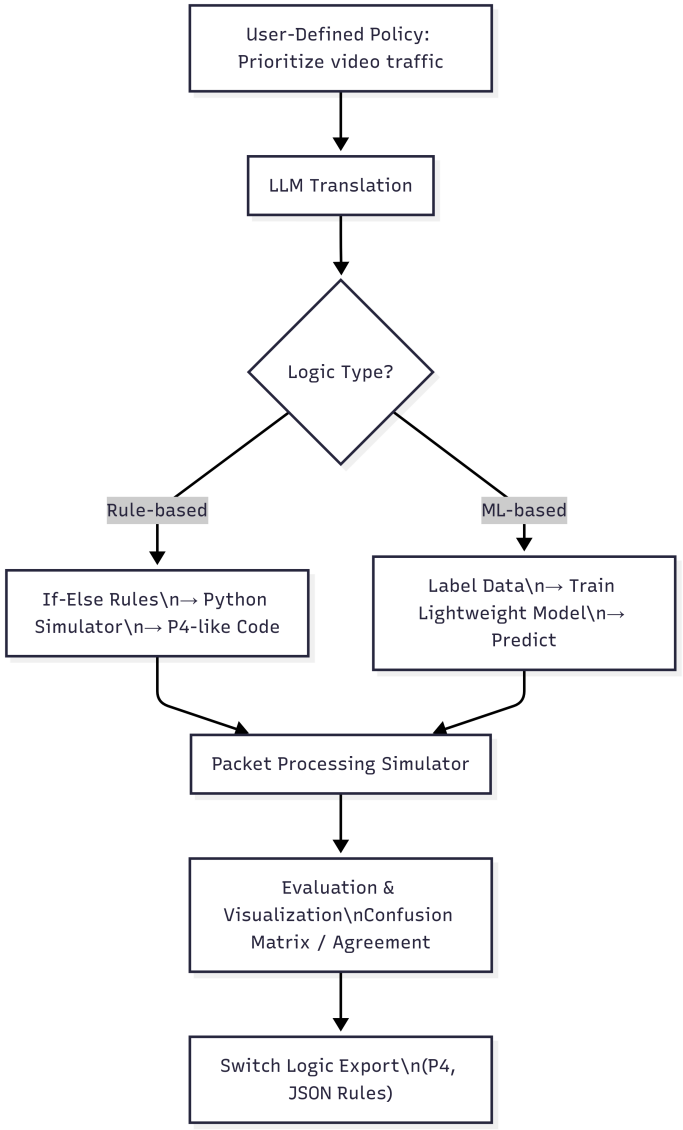

Figure 6: End-to-end Prompt-to-Pipeline workflow. Diagram created using Mermaid Chart AI.

5 Conclusions

This work presents a prototype of the Prompt-to-Pipeline workflow—an AI-driven framework for generating and evaluating intelligent per-packet logic in next-generation programmable networks. Inspired by the seminar *"Fast, Flexible, and Intelligent Next-Generation Networks and Systems"*, the project sought to bridge the gap between high-level intent specification and low-level data-plane implementation by leveraging modern artificial intelligence tools.

Large Language Models (LLMs) such as GPT-4o, Google Gemini, and Cogram

were central to this approach, translating human-readable policies into conditional logic and facilitating the generation of both symbolic if-else rules and labeled datasets for model training. Complementary tools such as GitHub Copilot and Mermaid Chart AI assisted in auto-generating P4-style switch logic and visualizing pipeline architecture, respectively. Tools like QuillBot and SlidesAI were employed to enhance academic clarity and presentation quality.

On the machine learning side, lightweight models—including shallow neural networks and SVMs—were trained using Keras and TensorFlow on synthetic telemetry data. These models were benchmarked for accuracy, inference latency, and memory footprint. The OpenAI Code Interpreter further supported analysis and visualization of confusion matrices, agreement metrics, and action distributions.

Results showed a 61.35% agreement rate between symbolic logic derived from LLMs and predictions from trained ML models. The models produced viable packet classifications that could be exported as simplified P4 logic or structured JSON rules, demonstrating the practical potential of this AI-enhanced methodology for future deployment in real switch environments.

Ultimately, this project underscores how AI tools can serve not just as assistants, but as enablers of new workflows in networking systems design. By automating policy translation, logic synthesis, and evaluation, the Prompt-to-Pipeline framework represents a meaningful step toward more agile, intelligent, and interpretable network infrastructures.

6 References

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3. ACM, 2014, pp. 87–95.

[2] J. Lian, Y. Ruan, X. Zhang, and B. Liang, "Netml: Machine learning for networked systems," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, 2020, pp. 172–179.

[3] C. Liu, X. Xie, X. Zhang, and Y. Cui, "Large language models for networking: Workflow, advances and challenges," *IEEE Network*, 2024.

[4] A. Sivaraman, M. Budiu, A. Cheung, C. Kim, and S. Narayanan, "Packet transactions: High-level programming for line-rate switches," in *ACM SIGCOMM*, 2016.

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[6] OpenAI, "Gpt-4 technical report," OpenAI, Tech. Rep., 2023. [Online]. Available: https://openai.com/research/gpt-4

[7] Cogram, "Cogram: Ai that understands your database and logic," 2023, https://www.cogram.com.

[8] F. Chollet, "Keras," 2015, https://keras.io.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] GitHub, "Github copilot: Your ai pair programmer," 2023, https://github.com/features/copilot.

[11] FlowGPT, "Flowgpt: Community prompt sharing for llms," 2024, https://flowgpt.com.