Mitigating Hallucination in Security Policy Generation with Large Language Models: A Prompt Ensembling Approach

Jet Wei Goh[1], Kamarul Ridwan Bin Abdul Rahim[1], Nathan Moyses[2], Vetrivel Maheswaran[3], Jaehoon (Paul) Jeong[4]
[1]Singapore University of Technology and Design, Singapore
[2]University of Alicante, Spain
[3]Rochester Institute of Technology, Rochester, NY 14623, USA
[4]Department of Computer Science & Engineering, Sungkyunkwan University, Suwon, Republic of Korea
{goh_jetwei, kamarulridwan_abdulrahim}@mymail.sutd.edu.sg, nrm69@alu.ua.es, vm6923@rit.edu, pauljeong@skku.edu

***Abstract*—Large Language Models (LLMs) have shown promise in translating high-level security intents into machine-readable policies for cloud-based security services. However, direct prompting of LLMs often produces hallucinated or invalid outputs, particularly when generating structured artifacts such as the Interface to Network Security Functions (I2NSF) Consumer-Facing Interface (CFI) policies. In this work, we propose a prompt ensembling approach that mitigates hallucination by decomposing policy generation into a sequence of specialized LLM prompts. Overall, our results show that schema-grounded prompt ensembling and intent filtering can substantially improve the reliability of LLM-driven security policy generation. The proposed design offers a practical path towards deployable policy assistants that translate natural-language intents into syntactically valid, standards-compliant I2NSF CFI policies with minimal manual correction.**

I. Introduction

Network security management in cloud environments requires translating high-level natural language user intents into low-level configurations that enforce desired security policies. The Internet Engineering Task Force (IETF)'s Interface to Network Security Functions (I2NSF) framework addresses this challenge by defining a Consumer-Facing Interface (CFI) YANG data model for expressing policies [1]. The CFI YANG model provides a standardized, vendor-agnostic structure that follows the Event-Condition-Action (ECA) paradigm, as well as incorporating endpoint groups (e.g., users, devices, locations, URLs, voice identifiers) to flexibly target entities, and threat prevention objects (e.g., threat feeds and payload content) to capture dynamic threat intelligence, as shown in Fig. 1.

Together, these components allow users to specify events that trigger security rules, the contextual conditions under which they apply, the endpoint groups they affect, the preventive intelligence they rely on, and the enforcement actions to be taken. This standardized schema ensures that high-level policies can be uniformly interpreted and deployed across Network Security Functions (NSFs).

However, manually authoring I2NSF CFI policies in XML is labour-intensive and error-prone, requiring deep expertise

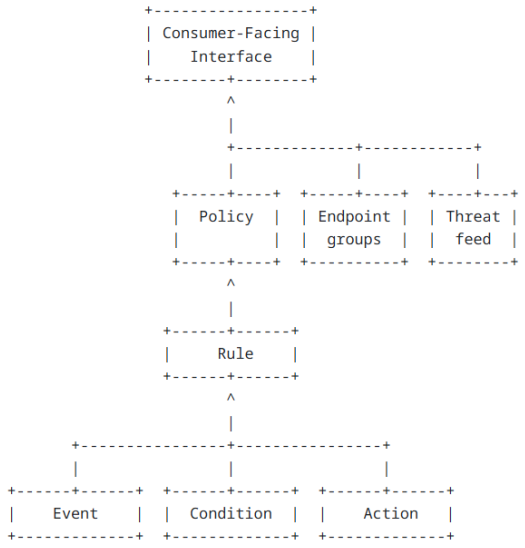

Fig. 1: The CFI YANG model schema.

in the data model and syntax. To simplify policy creation for non-experts, Rodriguez et al. [2] introduced a Security Policy Translator (SPT) that uses a LLM to translate natural-language intents into I2NSF CFI policies. Their work demonstrated the feasibility of integrating LLMs into the I2NSF workflow, but also highlighted important limitations around hallucinations, lack of datasets, and robustness.

II. Problem Formulation

In this work we study the problem of reliably generating I2NSF CFI security policies from natural-language intents using LLMs. Formally, given a user-provided intent describing a desired network security behaviour, our goal is to (i) decide whether the intent is a valid I2NSF-style security intent, and, if so, (ii) produce an XML policy that is both semantically faithful to the intent and syntactically compliant with the I2NSF CFI YANG schema.

While SPT [2] shows that a single LLM call can often produce plausible policies, three key limitations remain:

1) **Schema-level hallucinations:** When uncertain, the LLM frequently invents tags, values, or structures that are not part of the CFI YANG model (e.g., unsupported action names, incorrect tag names, mis-nested `<context>` and `<firewall>` blocks, or missing endpoint-group and threat-prevention sections). These errors cause policies to fail YANG validation or have unintended semantics.
2) **Lack of training data for structured models:** There is no large, publicly available dataset of aligned intent–policy pairs. This makes it difficult to build data-driven approaches such as retrieval over semantically similar intents, supervised fine-tuning, or knowledge-graph embedding models that directly map intents into the I2NSF policy space.
3) **No intent relevance filtering:** The baseline pipeline assumes every input is a well-formed security intent. In realistic deployments, however, a policy assistant will receive a mix of security and non-security queries. Passing off-domain requests straight to the policy generator can produce syntactically plausible but semantically meaningless I2NSF policies, as we illustrate in Section III-B1.

In this paper, we address the above challenges by proposing a prompt ensembling approach for security policy generation.

Our contributions are as follows:

1) We introduce a schema-grounded prompt ensembling pipeline that mirrors the workflow of multiple specialists. While prior work defines prompt ensembling primarily as combining multiple diverse prompts to improve robustness of LLM outputs [3], our approach takes this concept further by treating each LLM prompt as a domain-specific ”expert” aligned with the I2NSF CFI schema. Each stage extracts one component of the security policy from the intent, followed by schema checks, composition and final refinement. This modular architecture improves robustness and interpretability compared to prior single-prompt approaches [2].
2) We augment the pipeline with a schema-grounded intent relevance classifier that filters out off-domain user requests before policy generation. The classifier, implemented as an LLM expert informed by the I2NSF CFI Schema Reference Table, distinguishes between security-related intents that can be mapped to the I2NSF model and irrelevant queries.
3) We construct two synthetic benchmarks for this task, a set of 50 security intents for evaluating I2NSF policy generation, and a 100-intent dataset for intent relevance classification, enabling reproducible comparison of future approaches.

The implementation of the prompt ensembling pipeline, together with the synthetic intent datasets and evaluation scripts, is publicly available in our project repository at https://github.com/jaehoonpauljeong/KICS2026-Group3.

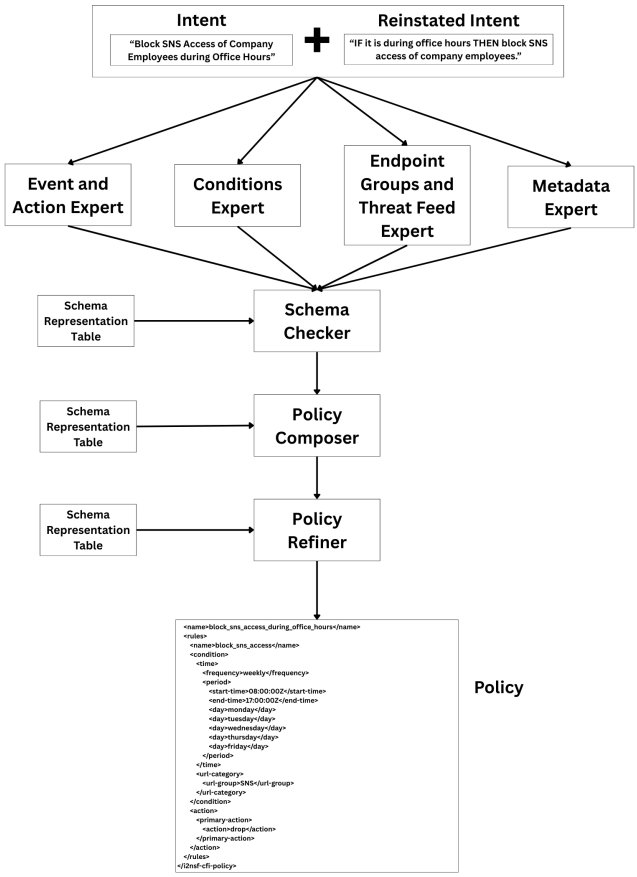

Fig. 2: Architecture of the Prompt Engineering approach.

III. Methodology

A. Prompt Ensembling Pipeline

Rather than relying on a single prompt to directly output an XML policy, our prompt ensembling pipeline decomposes the task into a series of specialized LLM prompts, each acting as an “expert” for a specific sub-task (e.g., event and action extraction, condition analysis, endpoint group identification, threat feeds, metadata generation). These modular outputs are then refined using a Schema Reference Table derived from the official I2NSF CFI YANG model before being composed into a final XML policy. This staged design significantly reduces the risk of hallucination by constraining each step to its own scope, grounding outputs in the schema vocabulary and reducing the cognitive load on the LLM in a single prompt, thereby ensuring that the generated policy is semantically faithful to the intent and syntactically compliant with the I2NSF CFI model. The prompt ensembling approach architecture is shown in Fig. 2.

The pipeline proceeds as follows:

1) **Intent Restatement**: The natural-language intent is reformulated into a normalized IF–THEN structure. This step ensures that the subsequent experts operate on a clean, explicit logical representation of the user’s

requirement, rather than on an ambiguous free-form description. By fixing a single canonical statement of "if X, then Y", we reduce the chance that later prompts reinterpret the intent differently or invent additional goals, which in turn limits high-level semantic hallucinations. The natural-language intent and the restated intent are then passed into each of the experts. For example, the free-form intent "Block SNS access during business hours." is restated as: "IF internal corporate users access SNS (e.g., Facebook, Instagram, X, TikTok) between 09:00 and 18:00, Monday–Friday, THEN drop the traffic at the firewall." This explicit IF–THEN formulation makes the triggering condition and enforcement action clear.

2) **Event and Action Expert**: A policy event–action expert identifies the triggering security events (e.g., access violation) and the actions to be taken (e.g., `drop`, `pass`, `reject`). Isolating events and actions in a dedicated step prevents them from being entangled with other components of the I2NSF CFI YANG model such as conditions or endpoint groups and allows the prompt to explicitly constrain the action vocabulary to those permitted by the I2NSF CFI YANG model.
3) **Conditions Expert**: A conditions expert extracts contextual constraints such as source/destination, protocol, or time-based restrictions that apply to the policy. Separating conditions from events and actions prevents the model from implicitly inferring or fabricating conditions at the composition stage (e.g., adding spurious time windows or protocols). This step reduces structural hallucinations where conditions are omitted, misplaced, or incorrectly merged with other parts of the policy.
4) **Endpoint Group and Threat Feed Expert**: A threat-intelligence expert detects references to endpoint groups (user, device, location, URL, voice) or external threat feeds and expresses them in terms compatible with the I2NSF CFI model. Since omission of endpoint groups and threat-prevention sections is a common failure mode, dedicating an expert to these elements explicitly prompts the LLM to surface all relevant entities instead of silently ignoring them. This reduces omission hallucinations in which policies appear valid but fail to bind to any concrete endpoints or threat feeds.
5) **Metadata Expert**: A metadata expert proposes policy-level descriptors such as policy name, rule name, language, priority usage, and resolution strategy. Centralizing metadata generation avoids having each expert or the composer independently invent names and priorities, which can lead to inconsistent or conflicting identifiers. This step reduces hallucinated or ad hoc metadata fields and enforces coherent, schema-aligned naming across the policy.
6) **Schema Readiness Checker**: The extracted components from the experts are validated against a Schema Reference Table — a CSV-formatted, worded representation of the official I2NSF CFI YANG model. The LLM checks whether extracted values are consistent with the schema's vocabulary. Any mismatches are flagged for correction before composition. This stage directly targets schema-level hallucinations by constraining free-form text to the finite set of allowed values and structures.
7) **Policy Composer**: Using the checked components from the experts and the Schema Reference Table as reference, a policy composition expert generates a complete XML policy. This prompt is carefully instructed to map extracted elements directly into their proper XML tags (e.g., `<firewall>`, `<url-category>`, `<action>`) and to refrain from introducing any new tags or values not present in the validated inputs. By forcing the composer to act as a deterministic mapper rather than a creative writer, this step reduces tag-level and structural hallucinations such as inventing `<actions>` instead of `<action>` or misnesting `<context>` under `<firewall>`.
8) **Policy Refiner**: The composed policy is passed through a final LLM-based refinement step that is guided by the I2NSF CFI schema and the Schema Reference Table. This component rewrites the XML to better match the canonical field order and structure (e.g. moving a misplaced `<context>` block out of `<firewall>`), and removes hallucinated details such as invented IP ranges, MAC addresses, URLs, or logging actions that are not supported by the original intent. This refiner serves as a last LLM-driven correction pass to reduce residual structural and value-level hallucinations before the policy is passed to external tools.

B. Intent Relevance Filtering

1) Risk of Processing Irrelevant Intents: The prompt-ensembling architecture described so far assumes that the natural-language input is a security intent that can, in principle, be expressed as an I2NSF CFI policy. In practice, however, users may submit intents that are entirely unrelated to network security. If such irrelevant intents are fed directly into the policy generation pipeline, the LLM may still hallucinate a seemingly plausible I2NSF CFI policy, despite the fact that no meaningful network control was requested.

We examined two example policies generated from clearly non-security intents using the prompt ensembling pipeline with GPT-4o-mini:

1) Intent 1: "Calculate the total payroll tax due for this month", as shown in Fig. 3.
   The corresponding hallucinated policy fabricates an endpoint group `"payroll-services"` and a rule named `"Current_Month_Payroll_Tax_Calculation"`. It then invents an invalid primary action value `calculate` that is not part of the I2NSF CFI action vocabulary (which only permits predefined actions such as `drop`, `pass`, or `reject`). As a result, yanglint [4] flags the policy as schema-invalid.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint-groups xmlns="urn:ietf:params:xml:ns:yang:ietf-
     i2nsf-cons-facing-interface">
    <user-group>
        <name>payroll-services</name>
        <range-ipv4-address>
            <start>192.0.2.1</start>
            <end>192.0.2.50</end>
        </range-ipv4-address>
    </user-group>
</endpoint-groups>
<i2nsf-cfi-policy xmlns="urn:ietf:params:xml:ns:yang:ietf-
     i2nsf-cons-facing-interface">
    <name>Payroll_Tax_Calculation_Policy</name>
    <language>en-US</language>
    <priority-usage>priority-by-order</priority-usage>
    <resolution-strategy>fmr</resolution-strategy>
    <rules>
        <name>Current_Month_Payroll_Tax_Calculation</name>
        <condition>
            <firewall>
                <source>payroll-services</source>
            </firewall>
            <context>
                <time>
                    <start-date-time>2023-10-01T00:00:00Z</
                         start-date-time>
                    <end-date-time>2023-10-31T23:59:59Z</
                         end-date-time>
                    <period>
                        <start-time>00:00:00Z</start-time>
                        <end-time>23:59:59Z</end-time>
                        <day>monday</day>
                        <day>tuesday</day>
                        <day>wednesday</day>
                        <day>thursday</day>
                        <day>friday</day>
                        <day>saturday</day>
                        <day>sunday</day>
                    </period>
                    <frequency>monthly</frequency>
                </time>
            </context>
        </condition>
        <action>
            <primary-action>
                <action>calculate</action>
            </primary-action>
        </action>
    </rules>
</i2nsf-cfi-policy>
```

Fig. 3: Hallucinated syntactically invalid XML policy for the non-security intent "Calculate the total payroll tax due for this month."

2) Intent 2: "Book a meeting room for tomorrow from 3pm to 4pm", as shown in Fig. 4.
   Here, the LLM synthesizes an endpoint group `"Meeting_Room_Booking_System"` and generates a schema-valid policy with a time window, firewall context, and a valid primary action `invoke-signaling`.

These examples illustrate two distinct failure modes. Firstly, syntactically invalid but obviously wrong policies as shown in Fig. 3. These can be caught by YANG-level validators and therefore less dangerous, though they still waste operator time and erode trust in the system. Secondly, syntactically valid but semantically irrelevant policies as shown in Fig. 4. These are more dangerous as they may be accepted and deployed by the controller, potentially changing how certain flows are handled, even though the user never intended to modify the network at all.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint-groups xmlns="urn:ietf:params:xml:ns:yang:ietf-
     i2nsf-cons-facing-interface">
  <user-group>
    <name>Meeting_Room_Booking_System</name>
    <range-ipv4-address>
      <start>192.0.2.1</start>
      <end>192.0.2.254</end>
    </range-ipv4-address>
  </user-group>
</endpoint-groups>
<i2nsf-cfi-policy xmlns="urn:ietf:params:xml:ns:yang:ietf-
     i2nsf-cons-facing-interface">
  <name>Meeting_Room_Booking_Policy</name>
  <language>en-US</language>
  <priority-usage>priority-by-order</priority-usage>
  <resolution-strategy>fmr</resolution-strategy>
  <rules>
    <name>Book_Meeting_Room_Availability_Check</name>
    <condition>
      <context>
        <time>
          <start-date-time>2023-10-10T15:00:00Z</start-date
               -time>
          <end-date-time>2023-10-10T16:00:00Z</end-date-
               time>
          <frequency>only-once</frequency>
        </time>
      </context>
      <firewall>
        <source>Meeting_Room_Booking_System</source>
      </firewall>
    </condition>
    <action>
      <primary-action>
        <action>invoke-signaling</action>
      </primary-action>
    </action>
  </rules>
</i2nsf-cfi-policy>
```

Fig. 4: Hallucinated but syntactically valid XML policy for the non-security intent "Book a meeting room for tomorrow from 3pm to 4pm."

2) Intent Relevance Filtering with an LLM Classifier: To address this, we introduce a schema-grounded intent relevance classifier as a pre-filter in front of the prompt-ensembling pipeline. This classifier is implemented as an additional LLM prompt "expert" whose only task is to decide whether a given intent is suitable for I2NSF policy generation with the Schema Reference Table. It takes the raw user intent as input and returns a binary label:

1) **Valid** – the intent clearly describes a network security or traffic-control policy that could reasonably be mapped to the I2NSF CFI YANG model (e.g., firewall rules, URL/application access control, threat-feed based blocking, logging/monitoring policies).
2) **Not Valid** – the intent describes some unrelated task (e.g., mathematical calculations, meeting bookings, report generation) that should not be translated into a network policy.

Only intents classified as valid are passed to the prompt ensembling pipeline. Intents classified as not valid are rejected early, thereby preventing the pipeline from hallucinating seemingly plausible policies from irrelevant, off-domain intents.

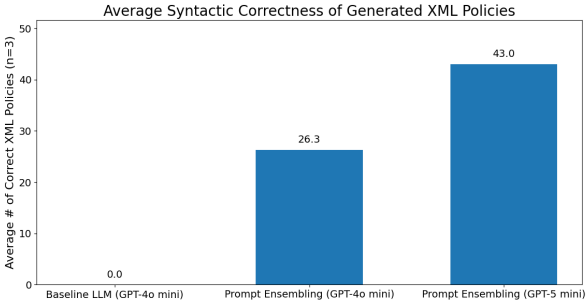

Fig. 5: Average number of syntactically correct XML policies over three runs for each method.

IV. Performance Evaluation

Our evaluation covers three aspects: (1) the syntactic correctness of generated policies, (2) a qualitative comparison of policy outputs for a representative intent, and (3) the accuracy of an LLM-based intent relevance classifier that filters off-domain inputs before policy generation. All experiments use the official I2NSF CFI YANG model as the ground truth for validation. For policy generation we compare a single-step baseline with our prompt ensembling pipeline instantiated with GPT-4o-mini and GPT-5-mini, and for intent relevance filtering we compare the GPT-4o-mini and GPT-5-mini LLM models.

A. Syntactic Correctness Comparison

We created a dataset of 50 synthetic security policy intents, designed to cover a range of scenarios. Each intent was expressed as a one-sentence description of a desired policy (e.g., "Block SNS Access during Business Hours."). We evaluated three configurations, a baseline single-step LLM that directly generates the XML policy from the intent with GPT-4o-mini, our prompt ensembling pipeline instantiated with GPT-4o-mini, and the same pipeline instantiated with GPT-5-mini. For each configuration, we ran the experiment three times over the full set of 50 intents and recorded, for every run, how many generated XML policies were syntactically valid. To assess syntactic validity, we used yanglint [4] to parse each XML and check compliance with the I2NSF CFI YANG schema.

The results are shown in Fig. 5. Across all three runs, the baseline LLM failed to produce a single schema-compliant policy. By contrast, the prompt ensembling pipeline with GPT-4o-mini as the backbone LLM achieved on average 52.6% syntactic correctness, while the same pipeline with GPT-5-mini achieved on average 86% syntactic correctness. The results prove that moving from a single-step baseline to the schema-grounded prompt ensemble yields a large gain in syntactic correctness, and upgrading the underlying LLM model further improves accuracy.

B. Qualitative Output Comparison

We select a representative intent, "Block SNS Access during Business Hours", to highlight the differences between the baseline LLM output and the prompt ensembling approach. Fig. 6 shows excerpts of the XML policies generated by each method, while Table I summarizes their compliance with key aspects of the I2NSF CFI YANG schema.

TABLE I: I2NSF CFI YANG schema properties for the "Block SNS Access during Business Hours" intent.

XML Policy Aspect	Baseline LLM	Prompt Ensembling
Explicit policy metadata (language, priority, resolution)	×	✓
Endpoint groups defined and referenced	×	✓
Valid `<firewall>` node under `<condition>`	×	✓
URL category bound to URL group	×	✓
Time context	Partial	✓
Primary action value (`drop`)	✓	✓
Schema-valid	×	✓

In this example, the baseline LLM captures the high-level idea of dropping SNS traffic during business hours, but still produces a schema-invalid policy. It uses a generic `"All_Devices"` source that is never defined as an endpoint group, omits any URL or destination grouping, and, more critically, introduces a non-existent `<firewall-condition>` element under `<condition>` instead of the schema-defined `<firewall>` node. This mismatch causes `yanglint` [4] to report a validation error ("Node `"firewall-condition"` not found as a child of `"condition"`") and the policy is rejected. The baseline policy also wraps the primary action in an `<actions>` container, which will also cause a validation error. It should have been `<action>`.

The prompt ensembling pipeline, in contrast, using the GPT-5-mini model, produces a fully schema-compliant policy. It defines explicit endpoint groups for corporate users and social networking sites, references these groups from the firewall and URL-category sections, encodes business hours as a recurring weekday period, and includes the optional policy-level metadata. The primary action `drop` is preserved, but now embedded in a structurally correct, reusable policy.

Across similar intents, we observed that single-step outputs often require substantial manual repair, whereas policies generated by the schema-grounded prompt ensembling pipeline are typically ready for deployment or need only minor edits.

C. Evaluation of LLM Classifier for Intent Relevance Filtering

We evaluated the LLM intent relevance classifier on a labeled dataset of 100 synthetic policy intents (50 security-related, 50 irrelevant), and compared two backbone models: GPT-4o-mini and GPT-5-mini. For each intent, the classifier predicts if the intent is either valid or not valid.

GPT-4o-mini correctly classified 91 out of 100 intents, achieving an accuracy of 91%. GPT-5-mini correctly classified 97 out of 100 intents, achieving an accuracy of 97%. These results, summarized in Fig. 7, confirm that an LLM-based pre-filter can effectively prevent off-domain intents from entering the policy generation pipeline, thereby reducing

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<i2nsf-cfi-policy
    xmlns="urn:ietf:params:xml:ns:yang:ietf-i2nsf-cons-
         facing-interface">
    <name>block_sns_access_policy</name>
    <rules>
        <name>block_sns_access_business_hours</name>
        <condition>
            <firewall-condition>
                <source>All_Devices</source>
            </firewall-condition>
            <context>
                <time>
                    <period>
                        <start-time>09:00</start-time>
                        <end-time>17:00</end-time>
                    </period>
                </time>
            </context>
        </condition>
        <actions>
            <primary-action>
                <action>drop</action>
            </primary-action>
        </actions>
    </rules>
</i2nsf-cfi-policy>
```

Listing 1: Baseline LLM XML policy output for the intent "Block SNS Access during Business Hours".

```xml
<?xml version="1.0" encoding="UTF-8"?>
<endpoint-groups xmlns="urn:ietf:params:xml:ns:yang:ietf-
     i2nsf-cons-facing-interface">
  <!-- Source group: Corporate users and devices -->
  <user-group>
    <name>Corporate_Users_Devices</name>
    <ipv4-prefix>192.0.2.0/24</ipv4-prefix>
  </user-group>

  <!-- Destination group: Social networking sites -->
  <url-group>
    <name>Social_Networking_Sites</name>
    <url>https://www.facebook.com</url>
    <url>https://twitter.com</url>
    <url>https://www.linkedin.com</url>
    <url>https://www.instagram.com</url>
  </url-group>
</endpoint-groups>

<i2nsf-cfi-policy xmlns="urn:ietf:params:xml:ns:yang:ietf-
     i2nsf-cons-facing-interface">
  <name>Block_SNS_Access_Business_Hours_Policy</name>
  <language>en-US</language>
  <priority-usage>priority-by-order</priority-usage>
  <resolution-strategy>fmr</resolution-strategy>

  <rules>
    <name>Deny_SNS_Access_During_Business_Hours</name>
    <condition>
      <firewall>
        <source>Corporate_Users_Devices</source>
      </firewall>
      <url-category>
        <url-name>Social_Networking_Sites</url-name>
      </url-category>
      <context>
        <time>
          <period>
            <start-time>09:00:00Z</start-time>
            <end-time>17:00:00Z</end-time>
            <day>monday</day>
            <day>tuesday</day>
            <day>wednesday</day>
            <day>thursday</day>
            <day>friday</day>
          </period>
          <frequency>weekly</frequency>
        </time>
      </context>
    </condition>
    <action>
      <primary-action>
        <action>drop</action>
      </primary-action>
    </action>
  </rules>
</i2nsf-cfi-policy>
```

Listing 2: Prompt ensembling pipeline (GPT-5-mini) XML policy output for the intent "Block SNS Access during Business Hours".

Fig. 6: Generated XML policies for the intent "Block SNS Access during Business Hours" (top: Baseline single-step LLM; bottom: Schema-grounded prompt ensembling pipeline).

the risk of hallucinated but syntactically valid policies being deployed.

D. Performance Considerations

A natural trade-off of the proposed multi-step design is latency, because the prompt ensembling pipeline invokes multiple LLM prompts and a schema lookup step, it is inherently slower than a single-step baseline that calls the model only once. To tackle this, several of the expert stages are relatively lightweight classification that could potentially be delegated to smaller specialised models to reduce overall runtime. In addition, knowledge distillation could be used to train a compact student model that approximates the behaviour of the full ensemble, thereby reducing the number of required model calls at inference time while preserving most of the accuracy benefits.

A second consideration is maintainability. Our method depends on a Schema Reference Table derived from the

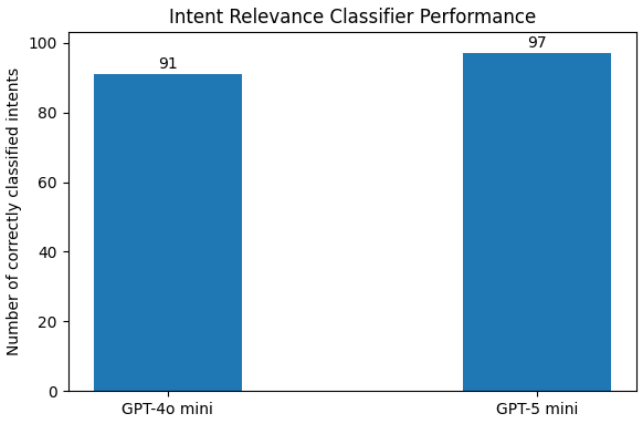

Fig. 7: Number of correctly classified intents for the intent relevance classifier using GPT-4o-mini and GPT-5-mini.

I2NSF CFI YANG model. If the data model were to expand or evolve, the reference table would need to be remade or updated to reflect the new vocabulary. While the I2NSF CFI schema is currently stable and effectively "frozen," this dependence highlights a limitation: the system is not dynamic to schema changes. In future work, one possible remedy would be to automate this process by having the LLM itself parse the YANG specification directly and regenerate the reference table on demand. Although we do not attempt this here, such an approach could reduce maintenance overhead if the schema were to change in the future.

V. Conclusion

We presented a schema-grounded prompt ensembling approach for generating I2NSF CFI security policies from natural-language intents using LLMs. By decomposing policy generation into a sequence of specialized experts for events/actions, conditions, endpoint groups and threat feeds, and metadata, and by validating their outputs against a Schema Reference Table derived from the official I2NSF CFI YANG model, our method substantially reduces the hallucinated tags, unsupported values, and structural errors that arise in single-step prompting. We further introduced an LLM-based intent relevance classifier that acts as a pre-filter in front of the policy generator.

The proposed schema-grounded prompt ensembling pipeline can be generalized beyond security policies to any domain where LLMs must emit precise, structured artifacts, such as other networking configurations or machine-readable specifications in XML, JSON, or fixed-schema formats. It is particularly useful for agentic LLM pipelines, where natural-language instructions must be reliably compiled into deterministic actions. Even when an initial generation violates schema constraints, the combination of schema grounding, validation, and regeneration allows the system to self-correct until a valid artifact is produced.

Future works can focus on the development of a gold dataset of intent to policy mappings, which will allow for the exploration of more powerful alternative learning-based approaches such as knowledge-graph embedding models that directly map intents into the I2NSF YANG space since training such models to reliably produce schema-valid XML would require a substantially larger, curated dataset of aligned intents and policies than is currently available. Additionally, a full functional testing in a live or emulated network can be performed for more rigorous testing. Prior work on the Security Policy Translator [5] hints at the feasibility of integrating I2NSF CFI policies with controllers and NSFs, by building an end-to-end testbed with OpenDaylight [6], RESTCONF-based translation to NSF-facing rules, and data-plane validation in environments such as Mininet [7].

VI. Acknowledgement

The authors would like to thank the I2NSF Working Group for providing the open data model and the anonymous reviewers for their valuable feedback. Note that Jaehoon (Paul) Jeong is the corresponding author.

References

[1] J. P. Jeong, C. Chung, T.-J. Ahn, R. Kumar, and S. Hares, "I2NSF consumer-facing interface YANG data model," Internet-Draft draft-ietf-i2nsf-consumer-facing-interface-dm, 2023, [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-i2nsf-consumer-facing-interface-dm/.

[2] M. L. Rodriguez, J. A. Berasategui, and J. P. Jeong, "Security policy generation for cloud-based security services using large language model," in *Proc. KICS Winter General Conf.*, 2025, [Online]. Available: http://iotlab.skku.edu/publications/domestic-conference/KICS-2025-Winter-LLM-Based-Security-Policy-Generation.pdf.

[3] S. Schulhoff, "Prompt ensembling," DiVeRSe and AMA for Accurate LLM Results, 2023, [Online]. Available: https://learnprompting.org/docs/reliability/ensembling.

[4] CESNET, "libyang: YANG data modeling language library," 2024, [Online]. Available: https://github.com/CESNET/libyang.

[5] P. Lingga, J. P. Jeong, J. Yang, and J. Kim, "SPT: Security policy translator for network security functions in cloud-based security services," *IEEE Trans. Dependable Secure Comput.*, 2024.

[6] OpenDaylight Project, "Opendaylight," 2025, [Online]. Available: https://www.opendaylight.org/.

[7] Mininet Project, "Mininet network emulator," 2025, [Online]. Available: https://mininet.org/.