Beyond Make-and-Model: Simplifying IoT Network Management with Behavioral Metaclasses

ALEKSANDAR PASQUINI, RAJESH VASA, and IRINI LOGOTHETIS, A2I2, Deakin University, Australia
HASSAN HABIBI GHARAKHEILI, School of Electrical Engineering and Telecommunications, University of New South Wales, Australia
ALEXANDER CHAMBERS and MINH TRAN, Information Sciences Division, Defence Science & Technology Group, Australia

Systematically managing the cyber health of Internet of Things (IoT) networks at scale requires understanding the functions and behaviors of connected devices. Current IoT device management approaches rely on classifying these behaviors based on the make-and-model of devices. However, these make-and-model classes proliferate over time and do not generalize well to unseen devices or networks. To overcome this, we propose a method to classify IoT behaviors into higher-abstraction metaclasses. Our contributions are three-fold: (1) We develop a method based on Large Language Models (LLM) to derive metaclasses, grouping 125 IoT make-and-model types into 7 functionality-based classes, using public datasets collected from five IoT networks. (2) We train PacketTransformer – a GPT-2 classifier – on data from one network and evaluate its generalizability on four unseen networks. In our cross-network setting, classification using metaclasses achieves up to 38% higher accuracy than make-and-model classification. (3) We investigate performance-improvement strategies beyond the multiclass baseline by training separate models for each metaclass and by calibrating stacked one-class outputs. These experiments improve performance in several settings, with the strongest one-class results achieving balanced accuracies of 71% for Cameras, 75% for Speakers and 79% for Sensors.

CCS Concepts: • **Networks** → **Network manageability**; • **Security and privacy** → Network security; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: IoT, Make-and-Model, Functionality Groups, Metaclasses, GPT-2, Device Identification, Out-of-Distribution Detection

ACM Reference Format:
Aleksandar Pasquini, Rajesh Vasa, Irini Logothetis, Hassan Habibi Gharakheili, Alexander Chambers, and Minh Tran. 2025. Beyond Make-and-Model: Simplifying IoT Network Management with Behavioral Metaclasses. 1, 1 (June 2025), 30 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The rapid proliferation of Internet of Things (IoT) devices has expanded the dynamics of networked systems. Network administrators face mounting challenges in managing thousands of diverse devices and identifying anomalous behaviors.

Authors' Contact Information: Aleksandar Pasquini, aleksandar.pasquini@deakin.edu.au; Rajesh Vasa, rajesh.vasa@deakin.edu.au; Irini Logothetis, rena.logothetis@deakin.edu.au, A2I2, Deakin University, Geelong, VIC, Australia; Hassan Habibi Gharakheili, h.habibi@unsw.edu.au, School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, NSW, Australia; Alexander Chambers, alexander.chambers@defence.gov.au; Minh Tran, minh.tran7@defence.gov.au, Information Sciences Division, Defence Science & Technology Group, Edinburgh, SA, Australia.

Manuscript submitted to ACM

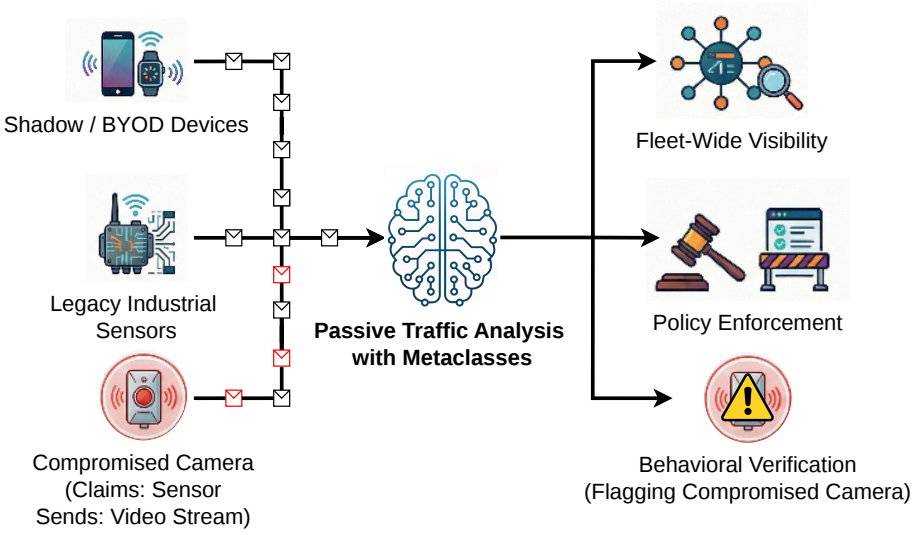

Fig. 1. Use cases of passive traffic analysis with metaclasses: (1) discover and label unknown/shadow IoT devices at scale, (2) enforce network policy by function (allow/deny/quarantine by metaclass), and (3) validate expected behavior to detect compromises or misconfigurations (*e.g.,* a "sensor" that starts streaming video)

While machine learning algorithms have proven effective for device identification [31] and for monitoring behavioral deviations [40], current approaches emphasize make-and-model classes. These classes rely on the maker (manufacturer), model (type and series), and connectivity features, such as Wi-Fi, Bluetooth, and Zigbee protocols, to distinguish devices.

This level of granularity, while precise, present three critical problems: (1) make-and-model classifiers require extensive training datasets capturing nuanced profiles for hundreds of distinct device types, (2) they fail to generalize to new devices or network environments without costly retraining, and (3) they create excessive cognitive load for administrators, who must reason about and maintain large numbers of device-specific security policies.

These limitations are amplified in common deployment settings where endpoint-declared identity is unavailable, unreliable, or operationally infeasible. These environments are illustrated in Fig. 1. In Bring-Your-Own-Device (BYOD) environments, users connect personal IoT devices without IT oversight. Administrators need visibility into these shadow devices and the ability to enforce policy without requiring device cooperation or pre-registration. In legacy and heterogeneous networks, many deployed devices (*e.g.,* older industrial sensors and consumer electronics) do not implement modern discovery standards such as W3C Web of Things [19] or oneM2M [28], making protocol-based identification incomplete or unavailable. Additionally, in security monitoring and behavioral verification, even when a device declares its identity through a discovery protocol, traffic analysis is necessary to verify that observed behavior matches its claimed functionality; for example, a device advertising itself as a temperature sensor but exhibiting sustained video-streaming traffic patterns warrants investigation. Finally, at larger scales, where network providers manage millions of customer devices, they cannot rely on endpoint cooperation for device identification; classification at network ingress points enables fleet-wide visibility for capacity planning and threat detection. Across these scenarios, the shared requirement is to understand device functionality from traffic alone without device modification, user intervention, or protocol compliance.

To overcome these limitations, higher-abstraction classes have been created. For instance, a device can be labeled "Camera" instead of "Nest Dropcam", as illustrated in Fig. 2. In this paper, classes are defined as the set of possible

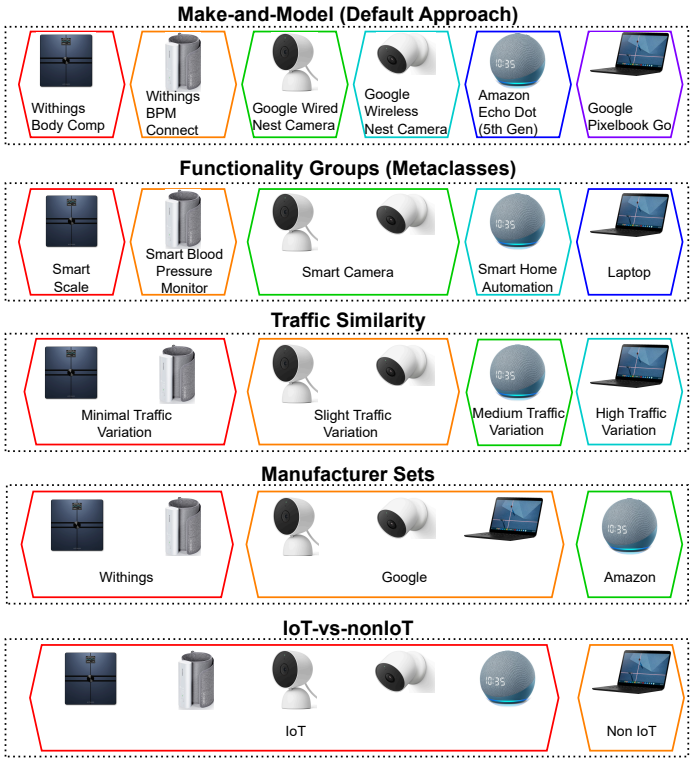

Fig. 2. Different methods of generating labels for the same set of devices. This paper will focus on Make-and-Model and Functionality Group classes.

categories (*e.g.*, IoT, Google, Smart Scale), while labels refer to the specific class assigned to an individual device instance. Various strategies can be employed to construct these classes and assign corresponding labels for device classification.

Previous studies have explored classification at a higher-abstraction level. While this approach mitigates challenges associated with the make-and-model classes, the studies fall short in key areas. Many evaluate their approaches within single networks [3, 5, 23, 32, 37], failing to test true generalizability to unknown devices. Additionally, class creation strategies lack consistency across studies [3, 4, 32, 37], producing varying numbers of classes even when using identical datasets.

We propose using functionality-group-based classes generated by a Large Language Model (LLM) to ensure scalable mapping. Functionality groups reduce the number of classes by organizing devices according to their function within the network. In the remainder of this work, we refer to these groups as metaclasses. This approach simplifies the class space while preserving meaningful insights into expected network behaviors and security requirements, supporting accurate and interpretable device classification [34, 43]. This also enables administrators to set and more critically manage appropriate policies while allowing classifiers to remain robust across different environments [18, 21].

This paper makes three contributions[1]: Our **first** contribution develops a novel LLM-based method to automatically map IoT device make-and-model types to higher-abstraction metaclasses, evaluating various LLMs by comparing their

[1]The code for this paper can be found here: https://github.com/AleksandarPasquini/Beyond-Make-and-Model-Simplifying-IoT-Network-Management-with-Behavioral-Metaclasses

Table 1. Previous works' class creation strategies. Our method falls under Functionality Groups.

Research Work	Class Type	Created by	# of Classes	Dataset
[42]	Manufacturer Set	Predefined Heuristic	4	Aalto and Private Dataset
[29]	IoT-vs-nonIoT	Predefined Heuristic	2	UNSW and Private Dataset
[6]	IoT-vs-nonIoT	Predefined Heuristic	2	UNSW, CIC-IDS2017, YourThings, ACI and Private Dataset
[12]	IoT-vs-nonIoT	Predefined Heuristic	2	UNSW, LSIF and Private Dataset
[21]	Traffic Similarity	K-nn Clusters	23	Aalto
[4]	Traffic Similarity	Traffic Fingerprint Similarity	3	UNSW
[8]	Traffic Similarity	Statistical Variance	4	Private Dataset
[23]	Functionality Groups	Research Team (Human)	9	Private Dataset
[5]	Functionality Groups	Research Team (Human)	7	Private Dataset
[17]	Functionality Groups	Research Team (Human)	6	Private Dataset
[2]	Functionality Groups	Research Team (Human)	11	Private Dataset
[20]	Functionality Groups	Research Team (Human)	7	UNSW
[32]	Functionality Groups	Research Team (Human)	14	UNSW
[3]	Functionality Groups	Research Team (Human)	4	UNSW
[37]	Functionality Groups	Research Team (Human)	8	UNSW
Our Approach	Functionality Groups	LLMs	7	Aalto, CICIoT-2022, Deakin, UNSW and LSIF

outputs to human-generated classifications. We also measure how consistently each device is assigned to the same label. Our **second** contribution demonstrates that our LLM-derived metaclasses enable a classifier to achieve higher accuracy on new networks and unseen device types compared to make-and-model classifiers. Our PacketTransformer model, based on GPT-2 [33], outperforms other machine learning approaches with minimal human intervention. Our **third** contribution investigates performance-improvement strategies beyond multiclass metaclass classification. We model each metaclass independently using perplexity-based and contrastive objectives, and we evaluate a stacked ensemble with MLP calibration to determine when these alternatives improve cross-network performance.

The structure of this paper is as follows: §2 reviews prior studies focused on creating classes for IoT devices. Our proposed strategy for generating metaclasses is detailed in §3. In §4, we present the PacketTransformer training methodology and evaluate performance in classifying devices based on their metaclasses. §5 develops adaptations to the PacketTransformer model that improve generalization to unseen devices and enable rejection of those outside the target metaclass. Finally, the paper concludes in §6.

2 Related Work

There are standardized approaches for IoT device description and discovery, such as W3C Web of Things (WoT) [19], oneM2M [28], and the CoRE Resource Directory [1]. These frameworks allow devices to advertise capabilities and services through structured metadata. Our traffic-based classification approach complements these standards rather than replacing them. These standards are not always available, and many consumer IoT devices and legacy industrial devices do not implement discovery protocols. In these cases, traffic analysis can help identify devices. Additionally, standards-based discovery typically requires active device participation, while traffic analysis operates passively at the network level, enabling classification without any device cooperation or modification. This is particularly valuable for security monitoring, where compromised or misconfigured devices may behave differently from their declared specifications. Our traffic analysis approach is therefore most valuable in heterogeneous environments where standards adoption is incomplete, or where independent verification of device behavior is required for security assurance.

Previous studies on IoT traffic analysis have explored various approaches to higher-abstraction classification, as summarized in Table 1. Despite IoT devices typically performing single functions and exhibiting consistent communication patterns well-suited for characterization, mapping their semantic behavior to higher-level abstract concepts remains ambiguous. As a result, class creation approaches are inconsistent. This variation reveals a fundamental trade-off: fewer classes reduce informational specificity, while more classes approach make-and-model at the cost of generalizability. To

better illustrate how previous works have constructed their classes, we categorize metaclass generation strategies into four primary approaches: IoT-vs-nonIoT, Manufacturer Sets, Behavioral Similarity, and Functionality Groups.

2.1 IoT-vs-nonIoT

The IoT-vs-nonIoT strategy divides devices into two broad categories based on standard definitions: IoT devices versus non IoT devices. The former are embedded devices with network connectivity performing specific functions via sensors/actuators, whereas the latter are general-purpose computing devices [11]. Studies leverage either traffic features such as Transmission Control Protocol (TCP) window size [6, 12] or neural networks [29] to create behavioral profiles. While these approaches demonstrate high accuracy within known device sets, they show degraded performance with unseen devices [29] and provide minimal practical utility for network administrators managing complex device behaviors.

2.2 Manufacturer Sets

Manufacturer sets enable brand-level device identification, allowing for market trend analysis and brand-specific behavioral tracking. This can be easily done via Media Access Control (MAC) address inspection, however, it is vulnerable to spoofing. Traffic pattern analysis provides an alternative but fails to account for functional similarities across brands, creating potential confusion. For example, [42] grouped Google, TP-Link, and WeMo devices into a single manufacturer class (WE-TP-CAST) based solely on traffic pattern similarities despite functional differences.

2.3 Traffic Similarity

Traffic similarity approaches map devices based on traffic pattern characteristics, providing insights into operational behaviors without requiring predefined labels. This unsupervised approach enhances adaptability to new devices but produces classes that lack immediate human interpretability, may contain subjective boundaries, and require post-processing to infer meaningful similarities.

Prior work by [4] grouped devices based on communication characteristics such as flow direction, packet size, and inter-arrival times, identifying three primary groups: G1 (devices using local DNS resolvers), G2 (devices with distinct internal/external communication patterns), and G3 (highly dynamic devices interacting with various home automation systems). However, these groups often contained functionally unrelated devices (*e.g.,* printers and sleep monitors in G1) and failed to capture broader operational similarities, limiting their administrative utility.

In another study, [8] employed the Cu index to measure behavioral predictability, creating four classes ranging from highly predictable (C1, *e.g.,* sensors) to low-predictability devices (C4, *e.g.,* streaming devices). However, this approach requires extensive long-term data collection to establish stable values, limiting practical application in real-time or resource-constrained environments. Similarly, AuDI [21] created fingerprints from 30-minute flow features and used K-Nearest Neighbours (K-NN) clustering to identify 23 communication pattern groups. However, 16 groups contained only single devices, requiring administrators to invest time interpreting these narrowly defined categories rather than focusing on network defense.

2.4 Functionality Groups

Functionality-based classification maps devices according to their primary purpose (Cameras, Lights, Sensors) and is done by humans, offering the most administratively relevant information. Understanding functional distinctions

(*e.g.,* sensors versus cameras) guides bandwidth allocation and vulnerability mitigation decisions that other approaches might obscure.

Studies have explored functionality groups classification using varied methodologies. ProfilIoT [23] analyzed TCP sessions with GeoIP enrichment while [3, 5, 17, 32] utilized statistical features including traffic volume, packet length, and protocols. In other work, [2] examined non-IP traffic from ZigBee and Z-Wave devices and leveraged inter-arrival times to preserve privacy by avoiding payload analysis. Despite their utility, studies such as [2, 3, 5, 17, 23, 32, 37] rely on manual, expert-driven metaclass creation, introducing inconsistencies and scalability challenges. For example, [32] created 14 classes from the UNSW dataset while [37] derived only 8 from the same data, with neither providing clear justification for their classification decisions. This subjectivity manifests in ambiguous categorizations, such as [17] placing the Netatmo weather station under “home automation" when it could logically fit under “appliance" or merit its own category as in [32].

2.5 Our Novelty

Our methodology advances the current device classification methods in three key dimensions. **First**, we pioneer the use of LLMs for generating functionality-based metaclasses, eliminating human subjectivity. Unlike [26], which uses LLMs for direct device classification, we prompt LLMs to generate metaclasses and map devices to these labels using minimal textual descriptors, demonstrating effective classification without device experts or extensive manual intervention. **Second**, we rigorously evaluate metaclass-trained model generalization across unseen devices and networks by training on one network and testing on others. This aspect is not addressed by prior metaclass-based studies, which introduce metaclasses but do not test transfer across networks. There are a few studies, such as FedHome [35], which address poor generalization across heterogeneous environments but they use different techniques instead of metaclasses. We also directly compare make-and-model and metaclasses classifiers under optimal conditions for make-and-model testing, demonstrating that LLM-derived metaclasses effectively trade specificity for transferability. This approach enables network administrators to achieve robust classification with basic device identifiers and minimal retraining. **Finally**, we develop a one-metaclass-per-model approach with two training strategies: a perplexity-based method evaluating how well device packets fit learned class distributions using a packet-generation layer and perplexity thresholds, and a contrastive learning method using triplet loss to cluster similar packets while separating different metaclasses. Both approaches enhance adaptation to novel devices without retraining, with contrastive learning further reducing training data requirements. While these models may exhibit overfitting when training data is limited (*e.g.,* single-device metaclasses) and sensitivity to device operational states (*e.g.,* setup vs. active phases), they demonstrate strong out-of-class packet rejection and effective generalization when provided with diverse training sets.

3 How Effective are Large Language Models at Mapping and Generating Metaclasses?

This section explores how LLMs can be employed to map and generate metaclasses from a list of labels (*i.e.,* IoT device make-and-models). An immediate question is whether the same grouping could be obtained directly from traffic, for example, by applying unsupervised clustering (*e.g.,* K-means over traffic-derived features) without involving an LLM. We argue that LLM-based metaclass generation offers distinct advantages. Firstly, LLM-generated labels such as “Cameras” and “Sensors” are immediately meaningful to administrators, whereas cluster indices require post-hoc interpretation. Secondly, the clustering method groups devices by traffic similarity, which may not reflect functional equivalence (*e.g.,* two devices with similar packet sizes may serve entirely different purposes). Thirdly, LLMs leverage extensive training on device descriptions and specifications, encoding domain knowledge that pure traffic analysis cannot access.

Finally, LLM-based labels remain stable across different networks, whereas clustering results depend on the specific traffic samples available. However, clustering-based validation of our LLM-generated metaclasses represents a valuable direction for future work, as traffic-based clusters that align with semantic metaclasses would provide independent validation of functional groupings.

3.1 Dataset of Make-and-Model Labels

Before generating metaclasses, a dataset containing a diverse range of device make-and-models is needed. A variety of device types and functionalities ensures the creation of robust metaclasses. To achieve this, five datasets from distinct networks were chosen: UNSW [39], LSIF [7], Aalto [24], CICIoT-2022 [9], and Deakin [30].

We also acknowledge that some of these captures are older (from 2016), which limits how broadly the results can be generalized to current IoT ecosystems. At the same time, this temporal spread provides a useful challenging test for the classifiers. If the metaclass models learn higher-level functional patterns rather than narrow device-specific signatures, they should be more robust to changes introduced over time by firmware updates, hardware revisions, and evolving traffic behaviors.

The devices included in the datasets are listed in the Appendix (Table 16). In total, there are 139 device names in the dataset, but for metaclass generation, only unique ones are considered. Thirteen duplicated names were identified, reducing the number of labels to 125. Similar names across different datasets were retained, as it is unclear whether these represent the same device model. For example, HP Envy and HP Printer are both printers, but they may not be identical models.

To generate the metaclasses, the prompt to the LLM will include only the make-and-model labels, each constructed from the device name and manufacturer name, when available. Some devices, such as “Smart_Lamp” in the LSIF dataset, lacked sufficient metadata to identify the device name and manufacturer. Despite this, these incomplete make-and-model labels are still valuable, as they reflect real-world scenarios where human users do not always have perfect information about the devices in their networks. Thus, these labels will serve as a useful test for LLMs.

3.2 Mapping Make-and-Model Labels to Human-Generated Metaclasses using LLMs

We employ LLMs to generate metaclasses due to their ability to process vast amounts of textual data and extract meaningful patterns. LLMs are well-suited for this task because their extensive training enables them to analyze device descriptions, infer high-level abstractions, and accurately map device make-and-models (names) to appropriate classes. Additionally, their scalability enables efficient handling of large datasets.

Before generating the LLM-based metaclasses, we conducted an initial experiment to evaluate the models’ ability to replicate human-generated classifications. Specifically, we assessed how effectively LLMs could map device make-and-models to metaclasses previously assigned by human annotators. While various LLMs can perform this task, we focused on four prominent models from different organizations: GPT-o1, LLaMA 3.3, Claude 3.7 Sonnet, and Gemini Flash 2.0. These models are the most current publicly available versions as of March 2025.

To evaluate the accuracy and consistency of LLM-generated mappings, we leveraged metaclasses created by humans in prior studies using the UNSW dataset, specifically those in [3, 20, 32, 37] (highlighted in yellow in Table 1). We unified these prior metaclasses into seven metaclasses, as shown in Table 2. This specific number reflects the statistical median of metaclasses used in the referenced studies and aligns with the classification schemes in [20, 37]. Note that one of the eight original metaclasses in [37] was excluded, since it pertained to non-IoT devices. Other studies, such as

Table 2. Mapping of metaclasses from prior studies to our unified metaclasses.

Our Unified Metaclasses	Metaclasses in [20]	Metaclasses in [32]	Metaclasses in [3]	Metaclasses in [37]
Hubs	Hubs	Smart Assistant Hub	Hubs	Hubs
Other	Electronics	Photo frame Printer Multimedia	Electronics	Electronics
Cameras	Cameras	Camera	Cameras	Cameras
Switches and Plugs	Switches & Triggers	Switches and Plugs	Switches & Triggers	Switches and Triggers
Sensors	Air quality sensors	Motion Sensor Weather station Smoke Alarm		Air quality sensors
Healthcare	Healthcare devices	Sleep sensor Blood Pressure meter Smart scale		Health
Lighting	Light Bulbs	Light bulbs		Lamps

Table 3. Summary of metaclass labels assigned to each IoT make-and-model by human annotators versus LLMs. There is full agreement except for Belkin Wemo Motion Sensor.

Device Make-and-Model	Our Metaclass	ChatGPT-o1	LLaMA 3.3	Claude 3.7	Gemini 2.0
Dropcam	Cameras	Cameras	Cameras	Cameras	Cameras
Samsung SmartCam	Cameras	Cameras	Cameras	Cameras	Cameras
Amazon Echo	Hubs	Hubs	Hubs	Hubs	Hubs
Belkin Wemo Switch	Switches and Plugs	Switches and Plugs	Switches and Plugs	Switches and Plugs	Switches and Plugs
Insteon Camera	Cameras	Cameras	Cameras	Cameras	Cameras
Netatmo Welcome	Cameras	Cameras	Cameras	Cameras	Cameras
Withings Smart Baby Monitor	Cameras	Cameras	Cameras	Cameras	Cameras
Smart Things	Hubs	Hubs	Hubs	Hubs	Hubs
Withings Aura Smart Sleep Sensor	Healthcare	Healthcare	Healthcare	Healthcare	Healthcare
TP-Link Day Night Cloud Camera	Cameras	Cameras	Cameras	Cameras	Cameras
HP Printer	Other	Other	Other	Other	Other
Netatmo Weather Station	Sensors	Sensors	Sensors	Sensors	Sensors
Triby Speaker	Other	Other	Other	Other	Other
LiFX Smart Bulb	Lighting	Lighting	Lighting	Lighting	Lighting
Nest Dropcam	Cameras	Cameras	Cameras	Cameras	Cameras
PIX-STAR Photo-Frame	Other	Other	Other	Other	Other
iHome Plug	Switches and Plugs	Switches and Plugs	Switches and Plugs	Switches and Plugs	Switches and Plugs
TP-Link Smart Plug	Switches and Plugs	Switches and Plugs	Switches and Plugs	Switches and Plugs	Switches and Plugs
Withings Smart Scale	Healthcare	Healthcare	Healthcare	Healthcare	Healthcare
NEST Protect Smoke Alarm	Sensors	Sensors	Sensors	Sensors	Sensors
Blipcare Blood Pressure Meter	Healthcare	Healthcare	Healthcare	Healthcare	Healthcare
Belkin Wemo Motion Sensor	Switches and Plugs	Sensors	Sensors	Sensors	Sensors

[3, 32], used different levels of granularity, which we reconciled by our unified scheme. Our final metaclass definitions (Table 2) are based on shared device functionality and abstraction.

Using this unified scheme, we relabeled the device make-and-model from prior work to establish a consistent ground truth. For each make-and-model, we selected the statistical mode (*i.e.,* the most frequently assigned metaclass) as its final label, shown in the second column of Table 3. For example, the Amazon Echo had been previously labeled as both "Hubs" and "Smart assistant". After consolidation, "Hub" emerged as the statistical mode and was used as the definitive label for comparison against LLM outputs.

To generate the LLM-based mappings, we used prompt 1. We tested several prompt formats, but this format was chosen to ensure a dictionary-style output, facilitating easier parsing and analysis.

Prompt 1 Mapping IoT Device Make-and-Models to Metaclass Labels

```
Map the following IoT device make-and-models into one of the seven labels in metaclasses. Assign
a corresponding integer value to each device make-and-model.

metaclasses = {
  1: "Hubs",
  2: "Other",
  3: "Cameras",
  4: "Switches and Plugs",
  5: "Sensors",
  6: "Healthcare",
  7: "Lighting"
}
make_and_model = {
  "SmartThings":
  "Amazon Echo":
  "Netatmo Welcome":
  . . .
  "Nest Dropcam":
}
```

Table 3 presents the mapping results for each device make-and-model, comparing LLM-generated metaclass to those assigned by humans. Overall, the LLMs demonstrated perfect internal agreement across all device mappings. Moreover, their outputs showed near-perfect alignment with the unified human-assigned metaclasses in the second column.

The only disagreement occurred with the Belkin Wemo Motion Sensor. While all LLMs categorized this device as the "Sensors" metaclass, our mapping assigned it to "Switches and Plugs". This discrepancy mainly arises from how we interpreted and aggregated labels from prior studies. In our unified scheme, the "Sensors" metaclass is broader than in the original works. As seen in Table 2, previous studies used narrower and more specific labels such as "Air quality sensors" and designated the Motion Sensor under the "Switches and Triggers" category. However, among the original references, there was some inconsistency. For example, [32] labeled the device as "Motion Sensor" instead of a switch-type device. This inconsistency in human annotations highlights the complexity of device categorization and further underscores the utility of LLMs in creating consistent, high-level abstractions.

3.3 Generating Metaclasses with LLMs

In the previous subsection, we showed that LLMs can effectively map device make-and-model names to human-created metaclasses. However, in practical, real-world settings, it may not always be feasible for humans to define such metaclasses in advance. This limitation motivates our next line of investigation: evaluating the ability of LLMs to autonomously generate metaclasses and corresponding device-to-metaclass mappings.

The key challenge in this approach is consistency. When tasked with generating metaclasses, LLMs may produce a different number of metaclasses across separate runs. In our initial trials, the number of generated metaclasses varied from nine to nineteen. One contributing factor is the LLM's temperature hyperparameter, which controls the randomness of token sampling. While a temperature of zero enforces deterministic behavior, other mechanisms, such as top-k/top-p filtering, nondeterministic Graphics Processing Unit (GPU) calculations, and varying random seeds, can

Prompt 2 Generating Metaclasses and Mapping IoT Devices

```
Generate seven metaclasses and map each of the following IoT devices to a metaclass. Output a
dictionary with the metaclass name as keys and the device integers as values.
label = {
  1: "Gosuna_Socket",
  2: "Minger_LightStrip",
  3: "Smart_LightStrip",
  . . .
  125: "GALAXY Watch5 Pro"
}
```

still introduce some randomness. To preserve fairness and generality, we retained each model's default temperature to reflect typical "out-of-the-box" performance without additional tuning.

Rather than adjusting the temperature, consistency can be achieved by explicitly specifying the number of metaclasses in the prompt. However, this introduces a trade-off between the number of metaclasses and the information provided by each metaclass. Too few metaclasses limit the information available to end users, whereas using an excessive number of metaclasses undermines their usefulness by introducing unnecessary granularity. We therefore selected seven metaclasses based on three considerations. First, seven is the statistical median across prior human-defined classification schemes (ranging from 4 to 14 in Table 1), aligning our design with established practice. Second, cognitive research [25] suggests that approximately seven categories approach the upper bound of what humans can reliably manage in short-term memory, supporting practical usability for network administrators. Third, seven provides sufficient granularity to separate functionally distinct device types (*e.g.,* cameras vs. sensors) while remaining general enough to support cross-device generalization within each metaclass. While the globally optimal number of metaclasses for all deployment scenarios remains an open research question, this choice balances these practical and theoretical considerations.

For this experiment, we used the full set of 125 make-and-model labels and to process them, the LLMs were given prompt 2. As with our earlier prompt 1, a dictionary output format was requested to facilitate subsequent analysis. However, in this case, device identifiers (integers) were used instead of full make-and-model labels to reduce output token length. This prompt was repeated 10 times, with each run conducted in a fresh session to prevent previous context from influencing the results.

Once the output was received, two validation checks were performed. The first ensured that exactly seven metaclasses had been generated; if this condition was not met, the failure was recorded and the session was restarted. The second check confirmed that each device was mapped exactly once. If the output did not meet the requirements, the LLM was prompted accordingly. If there were unmapped devices, it was prompted with:
`Please classify these devices:{<unmapped-dev>}`. If any devices had multiple labels, the prompt was:
`Please ensure {<dup.-dev>} only have one label`.

We grouped the LLM-generated metaclasses based on shared conceptual abstractions using the same logic used for human-defined metaclasses. This mapping is presented in the Appendix (Table 17). For example, if a device was labeled as "Plugs and Switches" in one run and "Smart Plugs & Switches" in another, we considered these as conceptually equivalent. Devices were then relabeled to reflect their corresponding general concept. To assess the consistency of the LLMs, we employed Fleiss' Kappa, which is a statistical measure of inter-rater agreement for categorical data that accounts for chance agreement. The metric ranges from 1 (perfect agreement, where each device is consistently assigned

Table 4. Consistency of LLMs in generating metaclasses and mapping device make-and-models to metaclasses across the 10 runs. ChatGPT-o1 is the most consistent model.

Metric	ChatGPT-o1	LLaMA 3.3	Claude 3.7	Gemini 2.0
Fleiss Kappa	**86**%	48%	81%	83%
Metaclass Generation Failures ($\neq 7$)	**0**%	50%	10%	**0**%
Unmapped Devices	**0**%	9%	1%	3%
Multi-Labeled Devices	2%	8%	**1**%	7%

Table 5. The top 3 most problematic device make-and-models for LLM mapping. Mid-list positions and uncommon names contributed to the classification difficulties.

Device Make-and-Model	LLaMA 3.3	Gemini 2.0
Atomi Coffee Maker	1	7
D-Link DCHS-161 Water Sensor	6	2
Belkin Wemo switch	6	1

to the same metaclass across all ten runs) to 0 (no agreement, where each device is assigned to different metaclasses in every run).

The results of the repeated prompting experiments are presented in Table 4. ChatGPT demonstrated the highest consistency, achieving a Kappa score of 0.86, and also had the fewest task failures. This performance is likely due to its longer inference times. As shown by work in [41], increasing inference-time computation can boost an LLM's performance, sometimes enabling smaller models to outperform much larger ones. However, this comes at the cost of higher Application Programming Interface (API) usage. In contrast, LLaMA 3.3 yielded the weakest results with a low Kappa score and frequent errors. This underperformance is likely due to its smaller size (70 billion parameters). However, its key advantage is its local deployability, thereby avoiding API costs altogether.

The top three devices that the models (*i.e.,* Gemini and Llama) struggled to map are shown in Table 5. Two main factors may contribute to this challenge. The first is positional bias, a known limitation of transformer-based architectures. These models distribute attention across input tokens in a way that can inadvertently undervalue items located far from the boundaries of a list. Many of the unmapped devices are in the middle of the device list (*e.g.,* Atomi Coffee Maker is item 66), making them more susceptible to reduced model attention. The second factor relates to the representation of the device and brand pairs in the LLM's training data. While "Coffee Maker" is a common term, enabling accurate mappings for similar devices, the brand name "Atomi" may be underrepresented or inconsistently encoded in the underlying training data. As a result, the LLM may not form a strong association between "Atomi Coffee Maker" and the appropriate "Appliances" metaclass, leading to the LLM not assigning it to any metaclass. Addressing these issues may involve fine-tuning LLMs on domain-specific IoT datasets or reordering the device list to ensure more uniform attention across all entries.

Mapping devices into multiple metaclasses is justified since certain devices, such as the Amazon Echo Show 8, serve multiple roles, functioning as both a hub and a smart speaker. Ideally, labels would reflect a proportion of time a device spends on each function (*e.g.,* 80% streaming, 20% hub operations), offering more details than a single-category assignment (*e.g.,* Smart Speakers). However, this paper does not consider multilabel classification, as most IoT devices perform only one primary function.

Based on our analysis, we selected GPT-o1's metaclasses and associated mappings for use in subsequent experiments. The consistency of GPT-o1 across runs (Fleiss' Kappa of 0.86) means that different runs produce nearly identical classifications, minimizing the impact of run selection on downstream classifier performance. For reproducibility, we

provide the complete device-to-metaclass mappings in Appendix Table 18, with color-coding indicating metaclass assignments. Among the datasets, only UNSW and CICIoT-2022 included devices spanning all metaclasses. We chose UNSW for training our classification models, as its relatively small device set offers a more rigorous test of the model's ability to generalize across metaclasses.

4 Are Classifiers Trained with Metaclasses Capable of Generalization Beyond the Training Data Distribution?

In this section, we investigate whether the LLM created metaclasses improve the generalization performance of classification models. Specifically, we evaluate whether metaclass-based classifiers outperform make-and-model classifiers when evaluated on traffic from different networks. We also assess which model architectures and packet encoding strategies are most effective for learning generalizable metaclass representations. To do this, we train a range of models on packet-level traffic data and test their ability to classify unseen devices using the five datasets introduced in §III-3.1.

Our Packet Data: Each dataset provides Packet Capture (PCAP) files containing IoT device traffic, typically covering idle and interactive states. The exception is the Aalto dataset, which only includes traffic captured during the setup phase. Despite the focus of Aalto, we include it as robust classifiers should be capable of identifying devices across operational phases. Additionally, all traffic is benign, ensuring that no malicious activity skews the learned behavior profiles.

For the training dataset, we cap the training contribution of each MAC address to the first 100,000 packets to control data volume and avoid over-representing any single device, thereby encouraging models to learn generalizable patterns rather than device-specific behaviors. To focus the signal on functionality, we restrict processing to packets that contain an Internet Protocol (IP) layer. This is consistent with prior findings that IP-layer packets provide the most useful information for packet classification [31]. However, at inference time, the classifier operates on a single packet. All our models are designed to classify a device from a single IP packet, which minimizes traffic requirements and makes the approach robust to packet loss.

4.1 Packet Processing and Classifier Models

Since our models require only one packet, classification can be performed immediately upon observing the first eligible packet for a MAC address and repeated as needed (*e.g.,* on subsequent packets or at periodic intervals) to smooth out transient variability and support continuous monitoring. Moreover, the use of single-packet inputs enables near-real-time operation with low compute overhead by avoiding stateful buffering, reassembly, or per-flow feature aggregation. While some pipelines require non-trivial preprocessing [31], we mitigate this by transforming packets into the smallest possible structured representations. Additionally, encodings, such as image-based and compression-based representations, can suppress device-specific noise and emphasize shared statistical structure that is more likely to transfer across networks, especially when paired with metaclass labels that reduce the effective class complexity.

Another advantage of our approach is that it does not rely on payload decryption. The packet headers remain observable under common encryption regimes. As encryption increases, payload-dependent cues diminish, reducing the discriminative signals available to any method that relies on deep packet inspection. However, the proposed encodings are compatible with header-based features and therefore remain applicable in encrypted settings. In fully encrypted scenarios, performance depends on how much discriminative behavioral signal is retained in observable fields. In practice, many IoT deployments still expose a mixture of encrypted and partially encrypted traffic due to legacy

protocols, ancillary services, or misconfigurations [22, 36]. Taken together, these design choices support the feasibility of deployment in real-world networks.

To evaluate the effectiveness of metaclass representations with single packet inputs, we implemented a variety of classifiers, namely 1-Dimensional Convolutional Neural Network (1D CNN), 2D CNN, Random Forest, Gradient Boosting, and our novel PacketTransformer model. Each model requires distinct pre-processing techniques. Following established packet encoding methodologies [31], we adapt three packet encoding methods to suit the respective models: byte-based encoding for 1D CNN, image-based encoding for the 2D CNN, and compression-based encoding for the Random Forest and Gradient Boosting classifiers. In addition, we introduce a new packet tokenization approach based on the GPT-2 transformer to capture semantic patterns within packets. We compare these encoding strategies and evaluate their impact on classifier performance. The following section will describe these models and the packet processing used by each model.

4.1.1 Byte-based. Byte-based packet encoding involves minimal transformation, making it a straightforward approach for preparing packet data. Each byte is converted to its integer value (0-255). Packets are then padded or truncated to a fixed length of 100 bytes, covering headers and most payload content, to standardize input.

A 1D CNN was chosen to train on these inputs. The 1D CNN architecture begins with an embedding layer mapping each byte to a 64-dimensional vector. This is followed by a convolutional layer (128 filters, kernel size 15) to capture local byte-level patterns, a 1D max-pooling layer for dimensionality reduction, and a 64-dimensional dense layer for higher-level feature abstraction, concluding with a classification layer. This lightweight model learns metaclass patterns for classification. Further details are in [31].

4.1.2 Image-based. The image-based approach transforms packet data into image representations, enabling the use of computer vision models for classification. We encoded packets as grayscale images. Each packet was first converted into a list of byte-derived integers, then padded or truncated to a fixed sequence length of 1000. These sequences were reshaped into a 2D array of dimensions 25×40, forming a single-channel image suitable for input into a 2D CNN model.

2D CNN models are well-suited to capture spatial relationships within data. The 2D byte arrangement allows learning of local patterns and global contextual dependencies. Our architecture, based on [13], comprises three convolutional blocks (Conv2D, batch normalization, ReLU). Filter counts start at 16 and double per block (to 64 in the third, which uses a stride of 2). The output is flattened and passed through two dense layers (128 and 64 units), followed by a final output layer with N_classes units and softmax activation for multi-class classification.

4.1.3 Compression-based. This approach uses Principal Component Analysis (PCA) to reduce packet dimensionality by retaining high-variance components and removing collinear ones. Packets are converted to integer lists, padded/truncated to 1000 bytes, then PCA projects them into 64-dimensional feature vectors. This compression captures structural patterns, yielding computationally lightweight, task-agnostic representations.

The resulting compact, structurally encoded 64-dimensional vectors are suitable for classical statistical machine learning models. We employed Random Forest (used in [24]) and Gradient Boosting (used in [5]) classifiers. Hyperparameters follow the configurations established in prior IoT classification studies [31], where they proved effective for IoT device classification.

4.2 PacketTransformer: Tokenization-based

Unlike the preceding strategies, tokenization incorporates packet semantics into the input representation. By associating field names with their data, this method introduces structural information, enabling models to learn intra-packet

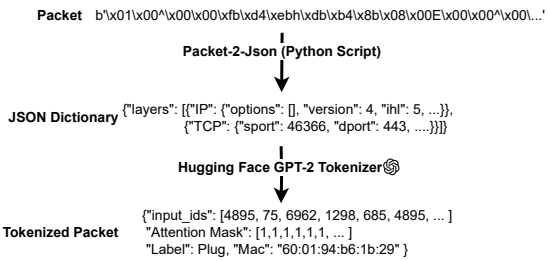

Fig. 3. Our packet tokenization pipeline. This input is only used with the PacketTransformer model.

Table 6. PacketTransformer architecture, where N is the batch size. $num\_classes$ depends on if the classifier is using make-and-model or metaclass labels.

Input	$[2D]_{N\times1000}$		
Attention_mask	$[2D]_{N\times1000}$		
Layer	**Output Shape**	**# of Parameters**	**Hyperparameters**
TokenAndPositionEmbedding	$[3D]_{N\times1000\times64}$	3280448	$vocab\_size = 50257$ $sequence\_length = 1000$ $embedding\_dim = 64$
TransformerDecoder	$[3D]_{N\times1000\times64}$	33472	$intermediate\_dim = 128$ $num\_heads = 2, dropout = 0.2$ $normalize\_first = True$ $decoder\_padding\_mask = attention\_mask$
TransformerDecoder	$[3D]_{N\times1000\times64}$	29344	$intermediate\_dim = 96$ $num\_heads = 2, dropout = 0.2$ $normalize\_first = True$ $decoder\_padding\_mask = attention\_mask$
TransformerDecoder	$[3D]_{N\times1000\times64}$	25216	$intermediate\_dim = 64$ $num\_heads = 2, dropout = 0.2$ $normalize\_first = True$ $decoder\_padding\_mask = attention\_mask$
LayerNormalization	$[3D]_{N\times1000\times64}$	128	
GlobalMaxPooling1D	$[2D]_{N\times64}$	0	$input = LayerNormalization$
GlobalAvgPooling1D	$[2D]_{N\times64}$	0	$input = LayerNormalization$
Concatenate([MaxPool,AvgPool])	$[2D]_{N\times128}$	0	
Classification	$[2D]_{N\times num\_classes}$	$num\_classes \times 128 + num\_classes$	$units = num\_classes, activation = softmax$

relationships more effectively. This structured, context-aware approach, contrasting with lower-level raw byte inputs, is expected to enhance classifier recall and generalizability.

To generate these semantic representations, each packet is processed as shown in Fig. 3. Packets are converted into JSON-serializable dictionaries by extracting fields (and their names as keys) from each protocol layer, starting from IP. Byte-type fields become hexadecimal strings, list-type fields become lists of strings, and other data types are cast to strings. Finally, the entire payload is appended as a hexadecimal string.

The resulting JSON text is processed by the Hugging Face GPT-2 tokenizer [15]. This tokenizer employs byte-pair encoding to segment the text into variable-length subword tokens derived from frequent byte pairs, which allows it to represent uncommon patterns by decomposing them into smaller units. Tokens are then mapped to numerical identifiers from its vocabulary. The `add_special_tokens` parameter was set to true, adding tokens, such as <|endoftext|>, to mark sequence ends. Token sequences were padded or truncated to a fixed length of 1000 tokens, and an attention mask was generated to differentiate real tokens (1s) from padding (0s).

The GPT-2 tokenizer's vocabulary, primarily designed for natural language, may not be optimal for network-specific terms or hexadecimal strings. However, given the nature of the transformer architecture at learning relationships

Table 7. *Similar Dataset.* Each row corresponds to a distinct make-and-model label, and each color represents a unique metaclass label. Devices within the same row are expected to share similar traffic patterns.

	UNSW (Training)	**Deakin** (Testing)
Plugs	TP-Link Smart plug	TOPERSUN Smart Plug
Cameras	Netatmo Welcome	Netatmo Smart Indoor Security Camera
	TP-Link Day Night Cloud camera	TP-Link Tapo Pan/Tilt Wi-Fi Camera
	Samsung SmartCam	SAMSUNG Pan/Tilt 1080P Wi-Fi Camera
Speakers	Amazon Echo	Echo Dot (4th Gen)
Sensors	NEST Protect smoke alarm	NEST Protect Smoke Alarm
	Netatmo weather station	Netatmo Weather Station
	Withings Smart scale	Withings Body+ (Scales)
	Belkin wemo motion sensor	Perfk Motion Sensor
	Blipcare Blood Pressure meter	Withings Connect (Blood Pressure)
Appliances	PIX-STAR Photo-frame	Pix-Star Easy Digital Photo Frame
	HP Printer	HP Envy

between tokens, and needing a baseline to start with, we did not develop packet-specific tokenization vocabularies in this research.

Finally, we developed PacketTransformer, a modified GPT-2 model (architecture in Table 6). It employs a decoder-only design with layers of masked multi-head self-attention and feed-forward sublayers. Self-attention helps identify patterns indicative of metaclasses, while learnable positional embeddings encode sequence order, and layer normalization stabilizes training.

4.3 Make-and-Model versus Metaclasses

We hypothesize that metaclasses offer greater generalizability than make-and-model labels, which rely on device-specific training data and thus limit applicability to unseen devices. The assumption that make-and-model models generalize well to seen devices across different networks is not well-validated for IoT. Metaclasses should achieve superior generalization across diverse device types. To test this, we trained models on a UNSW dataset subset to classify devices in a Deakin dataset subset. Two evaluations compared make-and-model and metaclass classifier accuracy, using identical hyperparameters for all five models, varying only the selected devices and labels.

The first experiment uses devices that are present in both datasets, as shown in Table 7 (Similar Dataset). This dataset contains 12 distinct make-and-model labels, each represented by at least one device in both the UNSW and Deakin datasets. In the table, rows correspond to devices with the same make-and-model label, while the colors indicate devices belonging to the same metaclass. The second experiment uses the entire UNSW training set and tests the classifiers on both the similar dataset. This experiment aims to assess the impact of training with a broader and potentially noisier set of devices on classifier performance. Table 8 and Table 9 present the performance of make-and-model labels and metaclass labels classification, respectively, across the two train-test dataset pairs.

These experiments represent a best-case scenario for make-and-model classifiers, yet none exceeded 30.21% accuracy (1D CNN). In all cases, performance improved when using metaclasses. As shown in Table 9, metaclass models consistently outperformed those in Table 8, demonstrating the strength of their learned latent representations. A reason for the poor performance of the make-and-model classes is many of the devices perform the same function. This leads to their network traffic packets exhibiting similar characteristics, resulting in overlapping class boundaries within the latent space and making it difficult for make-and-model models to differentiate between them. By combining such overlapping classes into broader metaclasses, the model benefits from more explicit class boundaries, increasing

Table 8. The accuracy of make-and-model classification. 1D CNN was the best-performing model, but all accuracies are low.

Train Dataset : Test Dataset	PacketTransformer	1D CNN	2D CNN	Random Forest	Gradient Boosting
UNSW Similar : Deakin Similar	10.70%	30.21%	18.64%	8.86%	6.61%
UNSW Full : Deakin Similar	8.17%	24.57%	19.82%	5.67%	4.67%

Table 9. The accuracy of metaclass classification. Up to a 6 times increase compared to make-and-model labels. PacketTransformer is now the best-performing classifier.

Train Dataset : Test Dataset	PacketTransformer	1D CNN	2D CNN	Random Forest	Gradient Boosting
UNSW Similar : Deakin Similar	36.33%	32.39%	28.62%	26.02%	29.76%
UNSW Full : Deakin Similar	46.83%	36.15%	32.36%	25.45%	23.33%

prediction accuracy. In addition, the use of fewer metaclass labels reduces the likelihood of misclassification among closely related classes, further boosting prediction performance.

These findings are broadly consistent with FedHome [35], which also identifies poor generalization across heterogeneous smart-home environments. However, FedHome addresses this problem through federated learning, where local gateway models are aggregated into a global model. In contrast, our goal is not to improve transfer through distributed optimization, but to improve it by changing the label space itself. By grouping devices into behavioral metaclasses, we encourage classifiers to learn higher-level functional patterns rather than highly specific device signatures, making generalization across environments more feasible without relying on federated infrastructure.

However, the results show that functional similarity alone does not guarantee strong transfer across networks. Devices with the same functionality are expected to exhibit similar traffic patterns, yet none of the models achieved high accuracy (maximum accuracy of 46.83%), suggesting discrepancies in traffic patterns between the two networks. One reason is that manufacturers release hardware revisions under the same model name. Newer units can ship with different chipsets, radios, or firmware that subtly change runtime behavior and their network traffic signatures [14]. These discrepancies can also arise from variations in network configurations or software updates on the devices themselves. For example, differing Maximum Transmission Unit (MTU) settings can lead to varying levels of packet fragmentation; a smaller MTU can cause larger packets to be split, altering their structure. Additionally, quality-of-service (QoS) settings can modify packet headers to prioritize or manage traffic differently across networks. Software updates can likewise change device behavior, as demonstrated in [40]. The multi-year gap between the two data collection periods likely had multiple software updates for many of the devices that introduced a range of behavioral changes. Thus, these behavioral variations undermine the consistency assumed by the classifiers.

The Full-Similar scenario yield lower classification accuracy than the Similar-Similar test because the model is trained on a broader set of device labels than in the test set. As a result, the model may assign probability mass to irrelevant classes, thereby increasing the chance of misclassification. This, again, causes the model to learn overlapping class boundaries, especially for devices with similar traffic patterns. Although the test set remains unchanged, the increased label diversity during training adds complexity and weakens classification signals, further degrading the performance. This effect is more pronounced with make-and-model classification, as the models must learn a larger number of classes. In contrast, models trained on metaclasses experience only a minor drop in accuracy due to the reduced number of class boundaries.

Therefore, effective classifiers should limit the number of labels to those necessary for the task, striking a balance between granularity and interpretability. Metaclass classifiers naturally achieve this balance. As a result, they not only generalize better to previously unseen devices, but they also adapt more effectively to new network environments. With fewer class boundaries and less dependence on device-specific identifiers, metaclass-based models can recognize

Table 10. Metaclass classification accuracy with the UNSW dataset as the training set. While the average accuracy is lower than 50%, PacketTransformer is the best model for generalization.

Testing Data	Pretrained GPT-2	Encoder	PacketTransformer	1D CNN	2D CNN	Random Forest	Gradient Boosting
LSIF	16.86%	24.63%	29.34%	7.46%	**31.51%**	17.84%	19.34%
Aalto	5.33%	10.11%	8.20%	12.33%	9.42%	13.86%	**18.65%**
Deakin	20.10%	31.90%	**34.38%**	33.78%	33.19%	26.63%	23.36%
CICIoT-2022	13.04%	**41.56%**	40.70%	28.39%	25.36%	23.75%	17.35%
Micro-Average	15.69%	34.31%	**35.72%**	24.04%	28.70%	22.79%	19.38%
Macro-Average	13.83%	27.05%	**28.16%**	20.49%	24.87%	20.52%	19.68%

functional patterns, enabling accurate classification even in unfamiliar network contexts. In contrast, a make-and-model classifier, tailored to specific device signatures, is more prone to misclassification or failure when presented with novel devices. This flexibility makes metaclass models more robust and practical for real-world deployment, as they reduce the need for constant retraining with each newly introduced device.

4.4 Comparison of Metaclass Models

We next evaluate which model performs best at learning metaclass behavior profiles. In this experiment, all available datasets are utilized. The UNSW dataset is used exclusively for training, while the remaining four datasets serve as test sets to assess the model's generalization ability.

As shown in Table 10, no model surpassed 50% accuracy on any single test dataset. The best-performing model varied across test datasets: PacketTransformer, 2D CNN, and Gradient Boosting each achieved top accuracy on at least one test dataset. The Gradient Boosted classifier demonstrated highest consistent performance, with no accuracy falling below 17%. However, PacketTransformer achieved the highest micro- and macro-average accuracies, suggesting the strongest overall performance.

While these metaclass accuracies on unseen networks (Table 10) do not exceed 50%, they generally surpass the make-and-model models' performance under similar cross-network generalization tests (Table 8, max 30.21% but often much lower, especially in the "Full" scenario). Make-and-model classifiers struggle with unseen device behaviors or network contexts due to their reliance on specific training signatures. Metaclasses capture broader, more transferable traffic patterns, which helps explain their stronger cross-network performance; however, the sub-50% accuracies reflect additional factors.

The limited diversity of IoT devices in the UNSW training set restricts representational learning, posing classification challenges. The dataset (see Table 18 for device distribution per metaclass) may not capture the full variability within each metaclass, as traffic patterns, protocols, and power profiles differ across manufacturers and device types. For example, different power profiles can affect traffic patterns due to differences in transmission frequency, protocol choice, and sleep behavior. These variations impact packet timing, size, and frequency, complicating consistent classification. This narrow representation leads to the metaclass classifiers overfitting and poor generalization, especially for devices at metaclass boundaries. These issues are compounded by intra-metaclass heterogeneity; for instance, the "Appliance" metaclass groups devices such as printers, photo frames, and coffee makers, which exhibit diverse network behaviors. Additionally, multi-year gaps between dataset collections (2016–2024) capture multiple software update cycles, introducing behavioral changes in devices that confuse the classifiers, further reducing classification accuracy.

These results indicate that metaclass classifiers at this stage are not sufficient as standalone decision tools. At most, they suggest a hypothesis that metaclass predictions may serve as a coarse first-stage filter within a broader pipeline, with later stages or local calibration refining the final decision. We do not directly validate such an operational workflow

in this paper. Instead, the remaining experiments examine whether predictive performance can be improved through alternative learning objectives, different model architectures, and post-hoc calibration.

4.4.1 Pretrained GPT-2 versus PacketTransformer. We next explored whether fine-tuning a large, pretrained GPT-2 model could outperform our PacketTransformer (a smaller, custom-trained GPT-2 variant). A pretrained GPT-2 offers a robust sequence-modeling foundation, having been exposed to a broad range of textual data, which may include textual packet representations. This could potentially reduce the need for packet-specific tokenization. Fine-tuning such a model on our task may allow it to capture nuances among IoT metaclasses that smaller or purely discriminative models might overlook.

For this comparison, we used the GPT-2 model released by openAI, available on the Hugging Face library [15]. The original GPT-2 models were trained on a filtered version of the Common Crawl dataset, which contains a wide range of diverse text on the Internet. We fine-tuned the pretrained GPT-2 model on our tokenized UNSW packet data, using the corresponding metaclass labels to adapt it for the IoT packet classification task.

PacketTransformer outperformed the fine-tuned, pretrained GPT-2 across all datasets (see Table 10, “Pretrained GPT-2" vs “PacketTransformer" columns). In some cases, the performance gap was substantial. For example, on the CICIoT-2022 dataset, accuracy improved from 13% to 40%, and on the Deakin dataset, from 20% to 34%. These results indicate that training a GPT-2 architecture from scratch on packet data yields specialized embeddings and attention patterns that are better aligned with the structural characteristics of IoT traffic, such as protocol types, headers, and payload formats. The domain mismatch between Internet text (used for GPT-2 training) and packet-level data was too significant for fine-tuning alone to bridge. Additionally, GPT-2 has more substantial memory requirements than PacketTransformer. This all suggests PacketTransformer’s size and architecture are well-suited to the UNSW dataset, and the domain-specific training from scratch is more beneficial than fine-tuning a very large, general-purpose model in this context.

4.4.2 Different Attention Mechanisms. We also developed an encoder-based transformer. Unlike decoders, which use causal attention (where each token attends only to preceding tokens), encoders apply bidirectional attention, allowing each token to attend to all others in a sequence. Input sequences are mapped to latent representations through this mechanism, and a classification layer operates on a pooled embedding to produce predictions. To isolate the impact of the attention mechanism, we modified the original PacketTransformer architecture by replacing its decoder layers with encoder layers using bidirectional attention. All other hyperparameters were kept constant, including the use of pre-layer normalization (unlike the more common post-layer normalization employed in many encoders, such as BERT [10]).

Compared to the classifiers in Table 10, the encoder-based model (third column) achieved the highest accuracy on the CICIoT-2022 dataset (41.56%), slightly outperforming our decoder-based PacketTransformer (fourth column) by less than 1%. It also surpassed PacketTransformer by about 2% on the Aalto dataset. However, PacketTransformer outperformed the encoder model on average. These results suggest that causal attention, as used in the decoder-style architecture of PacketTransformer, may offer a marginal advantage for packet classification tasks.

5 How Can the Generalization Ability of Classifiers be Improved?

While the PacketTransformer achieved the highest performance, its accuracy remains limited. The goal of this section is therefore explicitly improvement-oriented: we evaluate whether alternative learning objectives and post-hoc calibration can improve cross-network performance beyond the multiclass baseline.

As shown in earlier experiments, multi-class classifiers struggle when encountering entirely new device types. This is due to the complexity of learning multiple decision boundaries simultaneously, which increases the risk of overfitting to the training distribution. If a new device's characteristics (*e.g.,* novel protocols or usage patterns) are underrepresented, the model is likely to misclassify it. Addressing this would require retraining the full model with each new class, which is not scalable. Multi-class classifiers also face challenges in imbalanced settings. While we limit devices to the first-seen 100,000 packets, some devices do not emit many packets. For example, the Lights metaclass contains only 88,000 training packets from a single device, while the Cameras metaclass includes 676,000 packets from seven devices. Because multi-class models are trained using a softmax-based loss that jointly optimizes across all classes, dominant classes contribute more to the gradient updates. This can result in rare class patterns being underemphasized or ignored, ultimately reducing the model's ability to generalize.

One-class models offer a more scalable and robust alternative. Each model is trained independently to capture the behavior of a single metaclass, eliminating the need to model inter-class boundaries and reducing sensitivity to class imbalance. These models can be retrained individually as new device types emerge, avoiding the overhead of full model retraining. Additionally, because one-class models learn only from positive examples, each gradient step directly reinforces the defining features of the target class. This focused learning strategy should enhance generalization and improve performance on unseen devices.

5.1 Classification with Perplexity

To test the generalizability of one-class classifiers, we tested two approaches. For our first approach, we employ perplexity [38] inspired by NLP practices. In our experiment, each packet is treated analogously to a sentence, with individual bytes/tokens corresponding to words. Classifying packets based on perplexity involves three main steps, as illustrated in Fig. 4. In what follows, we explain these steps.

5.1.1 Predict Tokens with PacketTransformerGen. To evaluate whether one-class models outperform their multiclass counterparts, we trained a new variant of PacketTransformer. Instead of using the original multiclass classification layer (the last layer shown in Table 6), we replaced it with a packet-generating layer. This change aligns with the inherent design of decoder layers, which are specifically designed for sequence generation. Decoder layers learn token-level dependencies left-to-right, allowing the model to predict the next token within a sequence of tokens in a single packet.

PacketTransformer with a generative layer (PacketTransformerGen) still takes a single tokenized packet as input. Each packet consists of 1000 token positions, each containing a token from a vocabulary of 50,257, as defined by the GPT-2 tokenizer [15]. However, as the final token does not have a corresponding next-token target, it is excluded from prediction, yielding input sequence of length 999, such as $[100, 3, 75, 1, ..., 98]_{1\times999}$ as denoted by Input Tokens in Fig. 4. Consequently, the generative layer outputs a 2D tensor of logits with shape 999×50257, where each row corresponds to the model's prediction at a given position (0-998) for the next token in the sequence. For example, $[-8, -89, 78, ...]_{1\times50257}$, the first row in the matrix is the predicted logits that indicate the next possible token for the first token value 100.

5.1.2 Calculate the Perplexity. After obtaining the logit matrix from PacketTransformerGen, the softmax function [27] is applied to transform the logits in each row into a probability distribution, such as $[0.02, 0.05, 91.56, ...]_{1\times50257}$. The cross-entropy [16] is then computed by taking the negative logarithm of the probability assigned to the ground-truth token at each position (*i.e.,* Next Tokens[2]: $[3, 75, 1, ..., 98, 23]_{1\times999}$). The resulting token-level entropies can be averaged

[2]This is the input tokenized packet in Fig. 4, with the first token excluded.

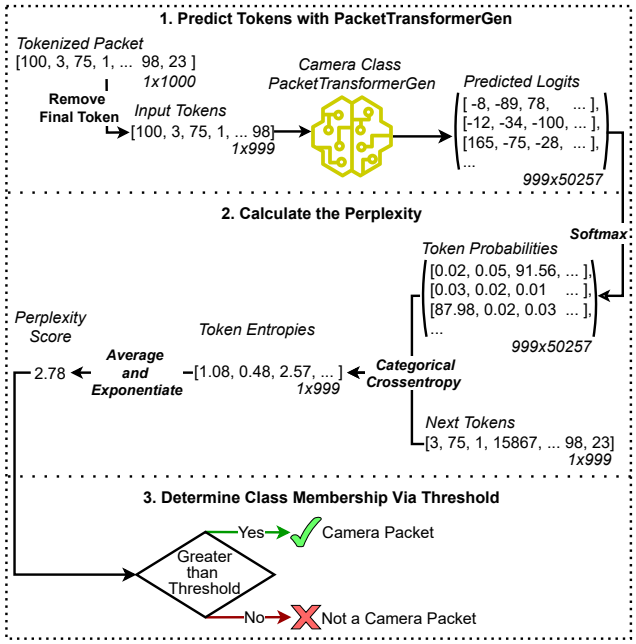

Fig. 4. Our perplexity-based inference pipeline: Classification is no longer performed directly by the PacketTransformer. Instead, predictions are made by thresholding against the training distribution.

and exponentiated to produce a single score, which quantifies how well a packet aligns with the learned distribution of a given metaclass. This score, known as perplexity (*e.g.,* 2.78 in Fig. 4), indicates how "surprising" the model finds the input data (*i.e.,* how well it aligns with the model's learned distribution). Perplexity can also be interpreted as the number of probable tokens the model considers. For example, a perplexity of 3 means that, on average, there were three likely token candidates observed across all positions in the input sequence of tokens. The perplexity formula penalizes models that assign low probability to the true next tokens in a sequence. Therefore, perplexity can serve as a classification decision criterion.

To assess whether the models effectively learned the true token distributions, Fig. 5 presents the average perplexity at each token position for packets belonging to the Cameras metaclass in the UNSW dataset. Two distinct regions are observed. The first region, spanning positions 0–180, has a generally increasing perplexity. At the beginning (0-25), the perplexity is one because all packets consistently start with the "Layers", "IP", "Options" tokens (see Fig. 3), resulting in no variability at those positions. The Perplexity later spikes due to inconsistencies in header values. These spikes are followed by noticeable drops at positions that are more likely to correspond to the field tokens (such as "version", "ttl" and "proto"), reflecting the repetitive nature of such fields across packets. Additionally, the tokenization encoding produces slight positional variations, contributing to higher perplexity. Packets with identical byte length can map to different token positions because the segmentation process can choose different subword boundaries. For instance, "12 34 56 78" might be encoded as the two tokens [500, 501], but switching the 56 to 57 changes the decomposition to [500, 505, 503], shifting all subsequent indices. Furthermore, the header size itself is variable. For example, a minimum IP/User Datagram Protocol (UDP) header (28 bytes) and IP/TCP header (40 bytes) expand to roughly 102 and 139 tokens, after which tokens encode payload bytes. Yet headers can grow to 68 bytes for IP/UDP and 120 bytes for IP/TCP, extending

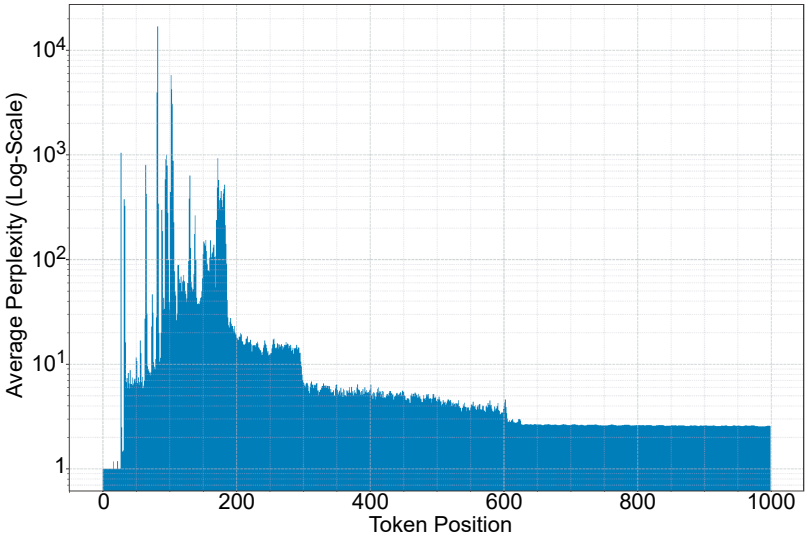

Fig. 5. The Camera-class model's perplexity distribution after 676,000 packets. The low perplexity in most positions suggests the model has learned to anticipate the correct tokens, particularly toward the end of the sequence.

the header boundary to around token positions 430 and 644, respectively. The resulting overlap of header and payload tokens between positions 100-180 produces the heightened perplexity observed in that segment. The second region spans the positions 181–998, where the perplexity initially plateaus at approximately 15 before gradually decreasing beyond position 300. This indicates that although some packets extend to the full 1000-token length, most terminate earlier, resulting in lower perplexity at later token positions.

5.1.3 Determine Class Membership Via Threshold. To classify new devices using perplexity, each packet is evaluated by computing its perplexity score under each class-specific model. This score is then compared against a predefined threshold: if the perplexity falls below the threshold, the packet is assigned to that class; otherwise, it is not. The thresholds are determined after model training by reprocessing the training set and recording the perplexity of each training packet. For each metaclass, we compute the perplexity percentiles [0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 0.99], which are then used to segment the corresponding test perplexities. While the absolute perplexity values at these percentiles vary across metaclasses due to packet variability, the use of consistent percentile ranks ensures uniformity across all models.

Individual Model Evaluation: The full results can be seen in our repository[3]. The tables organized by class per dataset. Four metrics are reported: True Positive Rate (TPR), True Negative Rate (TNR), F1 and Balanced Accuracy (BA). In a one-class classification setting, the TPR represents the share of in-class samples the model correctly accepts, while the TNR is the share of out-of-class samples it correctly rejects. The weighted F1 score is the harmonic mean of precision and TPR (recall), averaged across classes with weights proportional to their support. It captures the trade-off between false positives and false negatives. The BA score measures how well the model correctly identifies both in-class and out-of-class samples. It is calculated as the average of the TPR and TNR.

[3]https://doi.org/10.26187/deakin.31361119

Table 11. A summary of the perplexity-based classifier results, showing the best balanced accuracy per class and dataset. The Sensors metaclass was the best performing because of a diverse training set.

Dataset	Plugs	Cameras	Speakers	Sensors
LSIF	0.47	0.70	1.00	1.00
Aalto	0.50	0.58	1.00	0.63
Deakin	0.47	0.54	0.62	0.59
CICIoT-2022	0.49	0.59	0.68	0.75
Average	0.48	0.61	0.75	0.77

The PacketTransformerGen results shows that, generally, as the perplexity threshold approaches the 99th percentile, the TPR approaches the perfect rate of 1. This trend is desirable because it indicates that the metaclass packets from other networks are similar to those seen during the training. Among the datasets, LSIF is the most similar to UNSW, achieving the highest TPR at the lowest threshold. This can be attributed to the relatively small time gap of three years between them. Larger time gaps (*e.g.,* 8 years between UNSW and Deakin) can introduce greater variations in device behavior, leading to lower performance. Additionally, as the threshold increases, the TNR decreases toward 0. This is due to the higher threshold, leading the model to assign both in-class and out-of-class packets to the target metaclass. For example, Network Time Protocol (NTP) packets are similar across all metaclasses and are more likely to be closer to the training distribution than some metaclass-specific packets, leading to false positives.

To identify the best threshold, we examine the BA and F1 scores. It is important to note that the optimal threshold varies between datasets and metaclasses. For instance, the ideal threshold for the Cameras metaclass ranges from 10 in the Deakin dataset to 70 in the Aalto dataset. This highlights that a universal threshold is not feasible. Therefore, we recommend that network administrators calibrate the classifier using data from their own networks before deployment to ensure optimal performance.

For certain metaclasses, the optimum threshold performs no better than random guessing. For example, the Hubs classifier consistently predicts that all packets are out-of-class, resulting in a maximum BA of 0.5. The high F1 scores (0.92) is attributable to the absence of Hubs class packets in the datasets. The low BA indicates that the UNSW SmartThings device displays traffic patterns that do not overlap with those of the rest of the Hubs class. This highlights the need to train classifiers with various devices per metaclass to improve generalization. Similar trends can be seen in the Lights, Plugs and Appliances metaclasses. For these metaclasses, a low threshold (around the 30th percentile of the training data) is more optimal. Increasing the threshold beyond this point results in a disproportionate increase in false positives, nullifying gains in the TPR. This effect is exemplified in datasets that lack devices from certain metaclasses. For example, the Deakin dataset contains no "Light" devices, and applying a high threshold in such cases reduces overall accuracy without providing any benefit.

However, a higher threshold can be applied to the Cameras and Sensors metaclasses, as their more diverse training sets lead to higher TPRs. Speakers also perform well despite limited training data, as the test devices are mostly very similar to Amazon Echos. Table 11 shows the best BA achieved for Plugs, Cameras, Speakers, and Sensors across datasets. On average, the models correctly recognize seven out of ten packets. This level of performance allows users, particularly those unfamiliar with their network assets, to start developing a decent device catalog from the first packet received.

5.2 PacketTransformer with Contrastive Learning

Our second method uses a contrastive learning approach. As illustrated in Fig. 6, packet classification using this technique involves two steps:

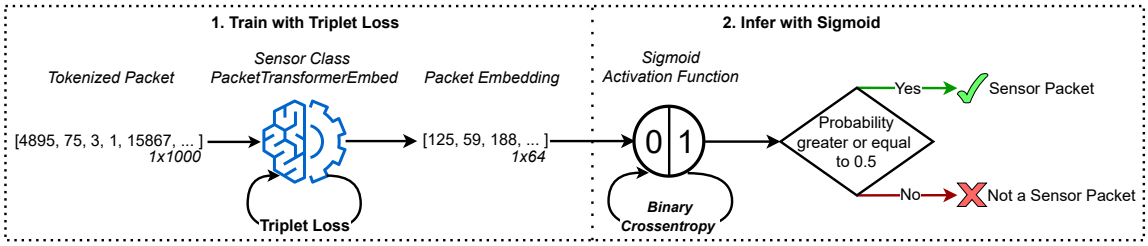

Fig. 6. Contrastive Pipeline. PacketTransformer is trained in two stages: triplet loss for embedding separation, followed by binary cross entropy for the final sigmoid layer.

5.2.1 Train with Triplet Loss. Contrastive learning trains a model to bring similar data points closer together in an embedding space while pushing dissimilar ones farther apart. To support this approach, we modified the PacketTransformer by replacing the final softmax layer with an embedding layer. This new model (PacketTransformerEmbed) is then trained on its embedding outputs using triplet loss.

Triplet loss is made up of three components:

- Anchor: a reference packet selected from the target class.
- Positive example: another packet from the same class as the anchor.
- Negative example: a packet from a different class, representing a dissimilar instance.

The loss function encourages the model to minimize the distance between the anchor and the positive in the embedding space, while maximizing the distance between the anchor and the negative. Specifically, it enforced a margin-based constraint: the anchor-positive distance must be smaller than the anchor-negative distance by at least a predefined margin. If this condition is not met, the model incurs a loss and adjusts the embeddings accordingly. Over time, this process shapes the embedding space such that packets from the same class form tight clusters, and those from different classes are well separated. This process enhances generalization in downstream classification tasks.

5.2.2 Infer with Sigmoid. After the model completes its training, its layers are frozen, and a dense layer with a sigmoid activation function is appended. This final layer determines whether an input packet belongs to the target class. A packet is classified as belonging to the class if its sigmoid output probability is greater than 0.5. The model is then fine-tuned by training only this final dense layer, while the rest of the network remains unchanged. After this step, the model is ready for evaluation.

For the experiments, we varied the margin between 0.1 and 1 in steps of 0.1. Since out-of-distribution device packets should already be separated out from in-distribution packets, we avoided using margins greater than 1. This prevents excessive separation of in-distribution packets from different networks. Also, since the model learns relational patterns rather than modeling each instance in isolation, less training data is required. We use a maximum of 400,000 packets (half from the target class and half from other classes), randomly selected with a fixed seed. This ensures consistent evaluation across different margin values.

Individual Model Evaluation: The PacketTransformerEmbed table[4] shows the performance of the contrastive learning models. Generally, the TPR is lower than the TNR. For example, in the Cameras metaclass, the TPR is 0.2 lower than TNR. This means that models are effective at rejecting out-of-class packets but struggle to recognize in-class packets. This imbalance gets more pronounced as the margin increases because the gradients further push negative pairs apart. While this enhances the TNR, it also excludes borderline positives and lowers the TPR. Such borderline cases are

[4]https://doi.org/10.26187/deakin.31361119

Table 12. A summary of contrastive classifier performance, showing the best balanced accuracy per metaclass and dataset. On average, the contrastive classifier performed as well as or better than the perplexity-based classifier.

Dataset	Plugs	Cameras	Speakers	Sensors
LSIF	0.68	0.83	1.00	0.89
Aalto	0.62	0.41	0.93	0.52
Deakin	0.70	0.64	0.58	0.62
CICIoT-2022	0.85	0.69	0.70	0.84
Average	0.77	0.71	0.75	0.79

common because devices with similar functions often exhibit modified behaviors across different networks, presenting distribution shifts that the models have never seen. Additionally, a drop in the performance of all metrics is observed when testing the Aalto dataset. This degradation is consistent with the results from the perplexity-based classifier. As with the multiclass classifier, one-class models struggle to correctly classify packets from devices in unfamiliar or shifted operating states.

While overall results are mixed, contrastive models demonstrate strong performance in specific scenarios. Table 12 presents these peak performances, which can be attributed to two main factors. First, high performance is observed when a dataset contains no devices from a particular metaclass. For example, the Aalto dataset contains no Speakers, yet achieves a BA of 0.93, primarily due to a very low false positive rate. Second, the presence of diverse device types in the training set enhances the model's generalization. For example, the Sensors model, trained on six distinct sensor types, achieved an average BA of 0.79.

Interestingly, the contrastive model outperforms the perplexity-based classifier in the Plugs metaclass. Its training objective clusters same-class packets tightly in the embedding space, allowing even unseen packets to remain close to the class centroid despite minor variations in header or payload. In contrast, the perplexity-based model tends to over-penalize novel byte patterns, leading to misclassification. This suggests that contrastive learning offers slightly greater robustness in scenarios with limited training data.

5.3 Ensembling the Models

Since individual models achieve high accuracy for certain metaclasses, combining them through ensembling is expected to yield better performance than a single multiclass classifier. We therefore created two model ensembles. Each ensemble uses a simple decision rule: for contrastive models, we assign each packet to the class with the highest predicted probability; for perplexity-based models, it is assigned to the class with the lowest perplexity score. Additionally, if a packet's score fails to meet any of the thresholds, it is discarded. Again, we evaluated performance across the same range of margin and perplexity thresholds.

Contrary to our expectations, the ensemble models did not outperform the best multiclass classifier across all datasets, as shown in Tables 13 and 14. Even when performance improvements were observed, they remained modest. The largest gain was achieved with the contrastive ensemble on the LSIF dataset, yielding a 4% improvement. This finding indicates that selecting classes based solely on the highest probability (contrastive) or lowest perplexity (perplexity-based) is not a robust classification strategy. For the perplexity ensemble, a key issue is the prevalence of low-entropy packets, those that are common across multiple devices. In such cases, all models tend to assign low perplexity values, making it difficult to distinguish the correct class and resulting in a high false positive rate. The contrastive ensemble suffers from the opposite problem: an excess of false negatives. As the margin increases, the models become more contrastive, and often none of them confidently claim a packet as their own, leading to many discarded predictions.

Table 13. PacketTransformer contrastive ensemble performance across margin values. Bolded values indicate better performance than the multiclass PacketTransformer. The poor accuracy is primarily driven by high false negative rates and suboptimal final prediction selection.

Margin	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
LSIF	15.84%	19.49%	15.42%	**34.05**%	15.14%	24.51%	21.66%	10.97%	12.92%	15.87%
Aalto	3.51%	**9.77**%	**9.48**%	**9.68**%	6.40%	7.39%	5.95%	**10.37**%	5.68%	**8.56**%
Deakin	24.95%	27.20%	25.76%	30.46%	26.83%	25.74%	26.26%	25.64%	24.55%	26.30%
CICIoT-2022	25.18%	34.63%	30.91%	29.77%	24.58%	34.32%	38.88%	35.09%	25.92%	26.60%

Table 14. PacketTransformer perplexity ensemble performance across threshold levels. Bolded values indicate better performance than the multiclass PacketTransformer. The poor accuracy is mainly attributed to high false positive rates and reliance on outdated traffic patterns during training.

Perplexity Percentile	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**99**
LSIF	5.21%	6.03%	2.30%	7.28%	6.52%	7.28%	13.85%	12.92%	9.61%	**29.49**%
Aalto	0.98%	2.71%	5.49%	**9.38**%	**10.61**%	**9.91**%	**12.18**%	**14.29**%	**14.15**%	**12.83**%
Deakin	2.98%	4.18%	8.44%	13.35%	9.34%	7.20%	19.89%	26.39%	22.45%	19.63%
CICIoT-2022	0.17%	1.18%	2.27%	7.91%	8.37%	7.80%	17.76%	17.42%	16.69%	20.36%

Table 15. Combined PacketTransformer ensemble, only incorporating models for Cameras, Sensors, Speakers, along with an Unknown class prediction. It outperforms the standalone multiclass model on most datasets. Performance is further enhanced after MLP calibration.

Dataset	**Multiclass PacketTransformer**	**Combined PacketTransformer with UNSW trained MLP**	**Combined PacketTransformer with CICIoT-2022 trained MLP**
LSIF	29.34%	**48.03**%	**59.12**%
Aalto	8.20%	**52.46**%	**65.02**%
Deakin	34.38%	**38.33**%	25.10%
CICIoT-2022	40.70%	32.60%	**57.26**%

We next refine the ensemble using two strategies. First, we limit ensembling to the most reliable one-class models in our experiments (*i.e.,* Cameras, Speakers, and Sensors), excluding others such as Hubs (sparse training data) or Appliances (high intra-class diversity), which add computational cost without clear performance benefit. Second, we stack both model types and use a Multilayer Perceptron (MLP) as the final predictor. The MLP receives each packet's perplexity score and contrastive probability as input, making the perplexity threshold irrelevant for classification. However, the margin used in contrastive learning is still important. We select a margin of 0.3 to reduce the high false positive rate from the perplexity models while retaining the sensitivity to avoid rejecting packets near class boundaries. Together, these features form a six-dimensional representation that is fed to a lightweight MLP classifier trained on the UNSW dataset. The MLP uses class-balanced weights, ℓ_2 regularization and layer normalization to mitigate overfitting. Packets with MLP probabilities below 0.8 are labeled as "unknown" (-1), covering devices outside the Speakers, Sensors, and Cameras metaclasses.

The results are summarized in Table 15. This stacked ensemble is the strongest improvement strategy evaluated in the paper. Relative to the multiclass PacketTransformer, it improves accuracy on LSIF (from 29.34% to 48.03%), Aalto (from 8.20% to 52.46%), and Deakin (from 34.38% to 38.33%) when paired with the UNSW-trained MLP. Its main failure case is CICIoT-2022, where it underperforms the multiclass model (32.60% versus 40.70%), primarily because the Plugs metaclass is excluded from the ensemble. To address this limitation, we calibrated the MLP using 90% of the CICIoT-2022 traffic data. Retraining only the MLP is more efficient than retraining the underlying perplexity and contrastive models. Therefore, the UNSW-trained PacketTransformer models were unchanged, and only their output scores were used to

train the MLP. This calibration increased CICIoT-2022 accuracy to 57.26% and improved performance on LSIF and Aalto, although it decreased Deakin, suggesting that the calibration benefit remains dataset-dependent.

Even after MLP calibration, the maximum accuracy is 65.02%, highlighting the difficulty of combining predictions from multiple models. The challenge arises from the limited information provided by a single packet and from disagreements among classifiers due to distribution shift. We therefore do not claim that the resulting system is a validated standalone operational tool. Rather, these experiments show that one-class modeling and stacked calibration can improve performance over the multiclass baseline in several settings and may support a coarse first-stage filter in a hierarchical pipeline. Direct validation of that utility remains future work.

6 Conclusion

This paper presented a novel framework to improve the classification of IoT devices by introducing functionally-based metaclasses derived from LLMs. Our results demonstrated that LLMs can consistently map make-and-model IoT device labels to functionality groups (metaclasses) and that classifiers trained on these metaclasses can generalize better to unseen devices and networks than make-and-model trained models in our setting, achieving up to 38% higher accuracy. We also investigated improvement-oriented strategies beyond the multiclass baseline. The strongest one-class results achieved balanced accuracies of 71% for Cameras, 75% for Speakers, and 79% for Sensors, while stacked MLP calibration improved multiclass accuracy across several datasets, reaching 65.02% in the best case. At the same time, these gains were mixed and dataset-dependent. We therefore view metaclass predictions as a promising coarse first-stage signal for IoT device classification.

Future work could explore two key directions. First, LLMs could be augmented with additional metadata, such as textual representations of network packet behavior, to improve the mapping of ambiguous devices. Second, research is needed to handle disagreements between classifiers, better classify devices in rare operational states, and directly evaluate whether metaclass predictions provide measurable value in end-to-end inventorying, triage, or policy workflows.

Acknowledgments

This research is supported by the Commonwealth of Australia as represented by the Defence Science and Technology Group of the Department of Defence.

References

[1] Christian Amsüss et al. 2022. Constrained RESTful Environments (CoRE) Resource Directory. RFC 9176. https://www.rfc-editor.org/info/rfc9176

[2] Leonardo Babun et al. 2020. Z-IoT: Passive Device-class Fingerprinting of ZigBee and Z-Wave IoT Devices. In *Proc. ICC*. Dublin, Ireland. doi:10.1109/ICC40277.2020.9149285

[3] Lei Bai et al. 2018. Automatic Device Classification from Network Traffic Streams of Internet of Things. In *Proc. IEEE LCN*. Chicago, IL, USA.

[4] Avewe Bassene and Bamba Gueye. 2021. A Group-Based IoT Devices Classification Through Network Traffic Analysis Based on Machine Learning Approach. In *Proc. AFRICOMM*. Springer, Ebène City, Mauritius.

[5] Bruhadeshwar Bezawada et al. 2018. Behavioral Fingerprinting of IoT Devices. In *Proc. ACM ASHES*. Toronto, Canada.

[6] Anat Bremler-Barr et al. 2020. IoT or NoT: Identifying IoT Devices in a Short Time Scale. In *Proc. IEEE/IFIP NOMS*. Budapest, Hungary.

[7] Batyr Charyyev and Mehmet Hadi Gunes. 2021. Locality-Sensitive IoT Network Traffic Fingerprinting for Device Identification. *IEEE Internet of Things Journal* 8, 3 (Nov 2021), 1272–1281. doi:10.1109/JIOT.2020.3035087

[8] Ivan Cvitić et al. 2021. Ensemble Machine Learning Approach for Classification of IoT Devices in Smart Home. *International Journal of Machine Learning and Cybernetics* 12, 11 (Jan 2021), 3179–3202.

[9] Sajjad Dadkhah et al. 2022. Towards the Development of a Realistic Multidimensional IoT Profiling Dataset. In *Proc. PST*. Fredericton, Canada.

[10] Jacob Devlin et al. 2018. Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (May 2018).

[11] Bruno Dorsemaine et al. 2015. Internet of Things: A Definition & Taxonomy. In *Proc. NGMAST*. IEEE, Cambridge, UK.

[12] Linna Fan et al. 2022. EvoIoT: An Evolutionary IoT and Non-IoT Classification Model in Open Environments. *Computer Networks* 219 (Dec 2022), 1–12.

[13] Yasir Ali Farrukh et al. 2023. Senet-i: An Approach for Detecting Network Intrusions Through Serialized Network Traffic Images. *Engineering Applications of Artificial Intelligence* 126, 4 (Nov 2023), 1–16.

[14] David Formby et al. 2016. Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems.. In *Network and Distributed System Security Symposium*. San Diego, California.

[15] Huggingface. 2025. *GPT2*. [Accessed 22-03-2025].

[16] Keras. 2025. SparseCategoricalCrossentropy. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy. [Accessed 22-03-2025].

[17] Roman Kolcun et al. 2020. The Case for Retraining of ML Models for IoT Device Identification at the Edge. *arXiv preprint arXiv:2011.08605* (Nov 2020).

[18] Kahraman Kostas et al. 2025. GeMID: Generalizable Models for IoT Device Identification. *Internet of Things* 34 (Nov 2025), 101806.

[19] Matthias Kovatsch et al. 2020. *Web of Things (WoT) Architecture*. W3C Recommendation. W3C. https://www.w3.org/TR/2020/REC-wot-architecture-20200409/

[20] Xiangyu Liu et al. 2022. IoT Device Identification Using Directional Packet Length Sequences and 1D-CNN. *Sensors* 22, 21 (Oct 2022), 1–20.

[21] Samuel Marchal et al. 2019. AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication. *IEEE Journal on Selected Areas in Communications* 37, 6 (Mar 2019), 1402–1412. doi:10.1109/JSAC.2019.2904364

[22] M Hammad Mazhar and Zubair Shafiq. 2020. Characterizing smart home iot traffic in the wild. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 203–215.

[23] Yair Meidan et al. 2017. ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis. In *Proc. SAC*. Marrakech, Morocco.

[24] Markus Miettinen et al. 2017. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In *Proc. IEEE ICDS*. Atlanta, GA, USA.

[25] George A Miller. 1956. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review* 63, 2 (Mar 1956), 81–97.

[26] Gabriel Morales et al. 2024. IoT Device Classification Using Link-Level Features for Traditional Machine Learning and Large Language Models. In *Proc. ICISSP*. INSTICC, SciTePress, Rome, Italy, 297–308. doi:10.5220/0012365700003648

[27] Chigozie Nwankpa et al. 2018. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. *arXiv preprint arXiv:1811.03378* (2018).

[28] oneM2M. 2023. *oneM2M Functional Architecture*. Technical Report TS-0001. oneM2M Partners Type 1 (ARIB, ATIS, CCSA, ETSI, TIA, TTA, TTC). https://www.onem2m.org/technical/published-specifications

[29] Jorge Ortiz et al. 2019. DeviceMien: Network Device Behavior Modeling for Identifying Unknown IoT Devices. In *Proc. IoTDI*. Quebec, Canada.

[30] Aleksandar Pasquini et al. 2025. Descriptor: Deakin IoT Traffic (D-IoT). *IEEE Data Descriptions* 2 (Mar 2025), 1–9. doi:10.1109/IEEEDATA.2025.3549716

[31] Aleksandar Pasquini et al. 2025. Robust and Lightweight Modeling of IoT Network Behaviors From Raw Traffic Packets. *IEEE Transactions on Machine Learning in Communications and Networking* 3 (Dec 2025), 98–116. doi:10.1109/TMLCN.2024.3517613

[32] Antônio J. Pinheiro et al. 2019. Identifying IoT Devices and Events Based on Packet Length from Encrypted Traffic. *Computer Communications* 144 (Aug 2019), 8–17.

[33] Alec Radford et al. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI blog* 1, 8 (Jan 2019), 1–10.

[34] Md Mizanur Rahman et al. 2024. The Influence of Device Type Aggregation on the Classification of Smart Home Devices using Machine Learning Algorithms. In *2024 27th International Conference on Computer and Information Technology (ICCIT)*. IEEE, Cox's Bazar, Bangladesh, 345–350.

[35] Md Mizanur Rahman et al. 2026. FedHome: A Federated Learning Framework for Smart Home Device Classification and Attack Detection by Broadband Service Providers. *Computer Networks* 277 (Mar 2026), 112040.

[36] Jingjing Ren, Daniel J Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In *Proceedings of the Internet Measurement Conference*. 267–279.

[37] Matias R. P. Santos et al. 2018. An Efficient Approach for Device Identification and Traffic Classification in IoT Ecosystems. In *Proc. ISCC*. Natal, Brazil.

[38] Sofia Serrano et al. 2023. *Language Models: A Guide for the Perplexed*.

[39] Arunan Sivanathan et al. 2018. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (Aug 2018), 1745–1759. doi:10.1109/TMC.2018.2866249

[40] Arunan Sivanathan et al. 2020. Detecting Behavioral Change of IoT Devices Using Clustering-based Network Traffic Modeling. *IEEE Internet of Things Journal* 7, 8 (Mar 2020), 7295–7309.

[41] Charlie Snell et al. 2024. Scaling LLM Test-Time Compute Optimally can be more Effective than Scaling Model Parameters. *arXiv preprint arXiv:2408.03314* (Aug 2024).

[42] Mohammed A Tawfeeq and Vian A Ferman. 2021. Gradient Boosting Algorithm for Early Detection of Unknown Internet of Things Devices. In *Online Scientific Conference for Graduate Engineering Students*. Baghdad, Iraq.

[43] Seungyong Yoon et al. 2017. Security Considerations Based on Classification of IoT Device Capabilities. In *Proc. 9th International Confernce on Advanced Service Computing*. Athens, Greece.

Table 16. The combined datasets.

UNSW	CICIoT-2022	Aalto	LSIF	Deakin	Total
Smart Things	Amazon Alexa Echo Dot	Philips Hue Bridge	Gosuna_Outlet_Socket	Amazon Echo Dot (4th Gen)	
Amazon Echo	Amazon Alexa Echo Spot	D-Link Connected Home Hub	Minger_LightStrip	Aeotec Smart Hub	
Netatmo Welcome	Amazon Alexa Echo Studio	WeMo Insight Switch	Smart_LightStrip	Netatmo Smart Indoor Security Camera	
TP-Link Day Night Cloud camera	Google Nest Mini	Ednet Wireless indoor IP camera	itTiot_Camera	TP-Link Tapo Pan/Tilt Wi-Fi Camera	
Samsung SmartCam	Sonos One Speaker	D-Link WiFi Day Camera	Tenvis_Camera	32' Smart Monitor M80B UHD	
Dropcam	AMCREST WiFi Camera	WeMo Switch model	Wemo_SmartPlug	SAMSUNG Pan/Tilt 1080P Wi-Fi Camera	
Insteon Camera	Arlo Base Station	Ednet living starter kit power Gateway	LaCrosse_AlarmClock	EZVIZ Security Camera	
Withings Smart Baby Monitor	Arlo Q Camera	Withings Wireless Scale WS-30	Lumiman_Bulb600	TOPERSUN Smart Plug	
Belkin Wemo switch	Borun/Sichuan-AI Camera	D-Link Door & Window sensor	Ring_Doorbell	Perfk Motion Sensor	
TP-Link Smart plug	DCS8000LHA1 D-Link Mini Camera	Edimax IC-3115W Smart HD WiFi Network Camera	oossxx_SmartPlug	NEST Protect smoke alarm	
iHome Plug	HeimVision Smart WiFi Camera	D-Link HD IP Camera	tp-link_SmartPlug	Netatmo Weather Station	
Belkin Wemo motion sensor	Home Eye Camera	WeMo Link Lighting Bridge model	Smart_Lamp	Withings Body+ (Scales)	
NEST Protect smoke alarm	Luohe Camera Dog	TP-LinkPlugHS110	tp-link_LightBulb	Withings Connect (Blood Pressure)	
Netatmo weather station	Nest Indoor Camera	Smarter iKettle 2.0 water kettle	D-Link_Camera936L	TUYA Smartdoor Bell	
Withings Smart scale	Netatmo Camera	TP-LinkPlugHS100	Wans_Camera	Pix-Star Easy Digital Photo Frame	
Blipcare Blood Pressure meter	SIMCAM 1S (AMPAKTec)	Fitbit Aria WiFi-enabled scale	Lumiman_Bulb900	HP Envy	
Withings Aura smart sleep sensor	Amazon Plug	Smarter SmarterCoffee coffee machine	Lumiman_SmartPlug	Amazon Echo Show 8	
Light Bulbs LiFX Smart Bulb	Atomi Coffee Maker	Homematic pluggable switch HMIP-PS	Gosuna_LightBulb	Ring Video Doorbell	
Triby Speaker	Eufy HomeBase	Cube LAN Gateway for MAX! Home automation sensors	Ocean_Radio	GALAXY Watch5 Pro	
PIX-STAR Photo-frame	Globe Lamp ESP_B1680C	Osram Lightify Gateway	Renpho_SmartPlug		
HP Printer	Gosund ESP_039AAF Socket	D-Link Water sensor	Chime_Doorbell		
Nest Dropcam	Gosund ESP_032979 Plug	D-Link WiFi Motion sensor	Goumia_Coffeemaker		
	Gosund ESP_0C3994 Plug	D-Link Siren			
	Gosund ESP_10098F Socket	Edimax SP-2101W Smart Plug Switch			
	Gosund ESP_1ACEE1 Socket	Edimax SP-1101W Smart Plug Switch			
	Gosund ESP_147FF9 Plug				
	Gosund ESP_10ACD8 Plug				
	HeimVision SmartLife Radio/Lamp				
	Philips Hue Bridge				
	Ring Base Station AC				
	iRobot Roomba				
	Smart Board				
	Teckin Plug				
	Yutron Plug				
	D-Link DCHS-161 Water Sensor				
	LG Smart TV				
	Netatmo Weather Station				
22	37	25	22	19	125

Table 17. Mapping of unique LLM generated labels to a metaclass

Metaclass	ChatGPT-o1	LLaMA 3.3	Claude 3.7	Gemini 2.0
Plugs & Switches	Plugs & Switches Smart Plugs & Switches Plugs/Switches Plugs and Switches Plugs_Switches SmartPlugsAndSwitches	Plugs and Sockets Smart Plugs and Outlets Smart Plugs Smart Plugs and Sockets Smart Plugs and Outlets	Smart Plugs/Sockets/Switches Smart Plugs & Sockets Smart Plugs/Switches Power Management Smart Plugs/Sockets Smart Plugs & Outlets	Smart Plugs/Outlets Smart Plugs/Switches Smart Plugs/Sockets/Switches Smart Plugs/Sockets
Lighting	Lighting Smart Lighting Lights Lights & Bulbs SmartLighting	Lighting Lighting and Plugs Smart Lighting	Smart Lighting Lighting Devices Lighting & Illumination	Smart Lights/Bulbs/Switches Smart Lighting Lighting Smart Lights/Bulbs/Strips Smart Lights/Bulbs
Cameras	Cameras Smart Cameras SmartCameras Cameras & Doorbells Security Doorbells & Security Doorbells	Cameras and Security Security and Alarm Systems Security Cameras Security and Surveillance Home Security Security Cameras	Smart Cameras & Doorbells Smart Cameras Security Devices Security & Monitoring Security & Surveillance	Smart Cameras/Security Cameras/Security Smart Cameras/Video
Voice Assistants & Smart Speakers	Voice Assistants & Smart Speakers Smart Speakers & Voice Assistants Voice Assistants Voice Assistants & Speakers VoiceAssistantsSmartSpeakers Voice_Assistants	Audio and Speakers Speakers and Audio Devices Audio and Entertainment Audio and Visual Speakers Audio Devices		Smart Speakers/Voice Assistants Smart Speakers/Displays/Assistants Home Hubs/Gateways & Speakers
Health & Wellness			Smart Health/Monitoring Devices Health & Wellness Devices Health & Wellness Health & Monitoring Devices Smart Health & Monitoring	
Security & Sensors	Sensors & Alarms Sensors Sensors, Alarms & Doorbells Sensors_Security DoorbellsAndSecuritySensors	Sensors and Monitors Health and Wellness Doorbells and Sensors Sensors and Monitoring Sensors Sensors and Wearables Sensors and Alarms	Smart Home Security & Sensors Environmental Sensors Environmental Monitoring Sensors & Monitoring Devices Smart Sensors/Alarms	Smart Devices/Other Smart Sensors, Smart Sensors/Other Devices Smart Sensors/Detectors Health/Wellness/Sensors
Appliances & Others	Appliances & Others Others Home Appliances & Misc Smart Appliances & Others Misc / Appliances Appliances_Other Health & Fitness Health & Wearables	Other Appliances Home Appliances	Smart Appliances Smart Home Appliances & Hubs Home Appliances Appliances & Automation Smart Kitchen/Appliances Smart Appliances & Other Smart Appliances/Other Miscellaneous Smart Devices,	Smart Appliances/Home Automation Smart Appliances/Other Smart Appliances Home Appliances Smart Appliances/Other Devices Smart Home Appliances/Sensors Other Smart Devices/Computers/Wearables Other/Smart Home Integration Smart Electronics/Other
Hubs & Gateways	Hubs & Gateways Hubs/Gateways Hubs_Gateways HubsGateways Hubs/Gateways/Bridges Hubs / Gateways / Security Sensors	Home Automation Hubs Smart Home and Automation	Smart Home Hubs/Gateways Smart Speakers & Displays Smart Hubs/Controllers Smart Home Hubs & Assistants Smart Home Hubs/Speakers/Assistants Smart Hubs & Assistants Smart Hubs/Gateways/Assistants Smart Sensors & Gateways Home Automation Hubs Smart Hubs/Gateways Smart Speakers/Displays Smart Speakers & Entertainment	Smart Home Hubs/Bridges/Gateways Smart Home Hubs/Bridges Smart Home Hubs/Gateways Home Hubs/Gateways Home Hubs/Gateways & Speakers

Table 18. ChatGPT o1's categorization of the devices.

MetaClass	UNSW	CICIoT-2022	Aalto	LSIF	Deakin	Total
Plugs	Belkin Wemo switch TP-Link Smart plug iHome Plug	Amazon Plug Gosund ESP_039AAF Socket Gosund ESP_032979 Plug Gosund ESP_10098F Socket Gosund ESP_0C3994 Plug Gosund ESP_1ACEE1 Socket Gosund ESP_147FF9 Plug Gosund ESP_10ACD8 Plug Teckin Plug Yutron Plug	WeMo Insight Switch WeMo Switch model TP-LinkPlugHS110 TP-LinkPlugHS100 Homematic pluggable switch HMIP-PS Edimax SP-1101W Smart Plug Switch Edimax SP-2101W Smart Plug Switch	Gosuna_Outlet_Socket Wemo_SmartPlug oossxx_SmartPlug tp-link_SmartPlug Lumiman_SmartPlug Renpho_SmartPlug	TOPERSUN Smart Plug	**27**
Lights	Light Bulbs LiFX Smart Bulb	Globe Lamp ESP_B1680C HeimVision SmartLife Radio/Lamp		Minger_LightStrip Smart_LightStrip Lumiman_Bulb600 Lumiman_Bulb900 tp-link_LightBulb Smart_Lamp Gosuna_LightBulb		**10**
Cameras	Netatmo Welcome TP-Link Day Night Cloud camera Samsung SmartCam Dropcam Insteon Camera Withings Smart Baby Monitor Nest Dropcam	AMCREST WiFi Camera Arlo Q Camera Borun/Sichuan-AI Camera DCS8000LHA1 D-Link Mini Camera HeimVision Smart WiFi Camera Home Eye Camera Luohe Cam Dog Nest Indoor Camera Netatmo Camera SIMCAM 1S (AMPAKTec)	Ednet Wireless indoor IP camera D-Link WiFi Day Camera D-Link HD IP Camera Edimax IC-3115W Smart HD WiFi Network Camera	itTiot_Camera Tenvis_Camera Ring_Doorbell D-Link_Camera936L Wans_Camera Chime Doorbel	Netatmo Smart Indoor Security Camera TP-Link Tapo Pan/Tilt Wi-Fi Camera SAMSUNG Pan/Tilt 1080P Wi-Fi Camera EZVIZ Security Camera TUYA Smartdoor Bell Ring Video Doorbell	**33**
Speakers	Amazon Echo Triby Speaker	Amazon Alexa Echo Studio Amazon Alexa Echo Spot Amazon Alexa Echo Dot Google Nest Mini Sonos One Speaker			Amazon Echo Dot (4th Gen) Amazon Echo Show 8	**9**
Sensors	Belkin Wemo motion sensor Netatmo Weather Station NEST Protect smoke alarm Withings Smart scale Blipcare Blood Pressure meter Withings Aura smart sleep sensor	D-Link Water sensor Netatmo Weather Station	Withings Wireless Scale WS-30 D-Link Door & Window sensor Fitbit Aria WiFi-enabled scale D-Link DCHS-161 Water Sensor D-Link WiFi Motion sensor D-Link Siren		GALAXY Watch5 Pro Netatmo Weather Station NEST Protect smoke alarm Perfk Motion Sensor Withings Body+ (Scales) Withings Connect (Blood Pressure)	**20**
Appliances	PIX-STAR Photo-frame HP Printer	Atomi Coffee Maker iRobot Roomba Smart Board LG Smart TV	Smarter iKettle 2.0 water kettle Smarter SmarterCoffee coffee machine	LaCrosse_AlarmClock Ocean_Radio Goumia_Coffeemaker	32' Smart Monitor M80B UHD Pix-Star Easy Digital Photo Frame HP Envy	**14**
Hubs	Smart Things	Arlo Base Station Eufy HomeBase Philips Hue Bridge Ring Base Station AC	D-Link Connected Home Hub Ednet living starter kit power Gateway WeMo Link Lighting Bridge model Cube LAN Gateway for MAX! Home automation sensors Osram Lightify Gateway Philips Hue Bridge		Aeotec Smart Hub	**12**