Identifying Operators of IP Services At Scale with OperatorSage

Paper #537, 11 pages body, 19 pages total

Abstract

Identifying the operators of Internet-facing services is essential for understanding and securing the Internet. Existing approaches often assume IP operatorship aligns with hosting, or rely on manual analysis which limits scalability. We present OperatorSage, a system that automates operator attribution using Large Language Models (LLMs) to interpret high-volume IP service data such as DNS records, TLS certificates, and WHOIS entries. OperatorSage is 46 times faster and more accurate than a human. Within 40 hours, OperatorSage attributes operators for over 150,000 IPv4 addresses, enabling the largest study to date of individual IPv4 service operators. OperatorSage is open-sourced under the Apache 2.0 license to support scalable, automated analysis in the research community.

1 Introduction

Identifying the entities that operate Internet-facing services is critical to both understanding and protecting the Internet. Coordinated vulnerability disclosure [5, 12, 19], Internet centralization studies [25], legal action against malicious activity [4], all rely on answering "Who is using this IP?"

Traditionally, attribution focuses on identifying the operator of the underlying infrastructure—such as the WHOIS contact [14, 22, 23] or the Autonomous System (AS) announcing the prefix [26]. In some cases, researchers may fingerprint the hardware vendor or hosting provider for attribution, especially when the issue involves a vulnerability in the infrastructure itself [2]. These methods are reasonable when IP ownership, service operation, and hardware deployment align.

However, the widespread adoption of cloud platforms, virtual hosting, and IP address leasing has increasingly decoupled infrastructure ownership from service operation [17]. An IP address may belong to a block owned by Amazon or Google, yet the actual application is deployed and managed by a third-party tenant. Consequently, we now require methods that go beyond infrastructure ownership to accurately identify service operators. For example, coordinated vulnerability disclosure requires identifying the true operator of a service so that researchers can responsibly report flaws [22]. Understanding IP leasing markets hinges on tracking who is currently operating services on leased address space [11]. In such scenarios, the misalignment between infrastructure signals and actual operators may partly explain persistently low rates of vulnerability remediation [14].

To bridge this gap, researchers can resort to manual analysis to extract operator identities from contextual clues. For instance, prior work has analyzed access control lists [12] and TLS certificates [13] to infer service ownership in cloud-hosted and shared ISP environments. However, these methods are labor-intensive and inherently non-scalable: our own findings show that manually contextualizing application-layer data to identify service operators takes, on average, 1.6 minutes per host—making manual interpretation impractical for datasets even modestly larger than 1,000 assets. Without automated techniques to attribute service operation, researchers are left with vast quantities of Internet measurement data, but limited means to interpret it meaningfully.

We introduce OperatorSage, a system that automates the attribution of service operators from IP service data. At its core, OperatorSage uses a Large Language Model (LLM) to interpret high-volume data—such as DNS records, TLS certificates, and WHOIS entries—that all together would be impractical for a human to analyze at scale. To improve accuracy, OperatorSage trains the LLM with an Internet-measurement-specific prompt that incorporates structured input, few-shot examples, and guardrails. To enhance scalability and reduce cost, OperatorSage prunes application-layer data before it reaches the LLM and caches previously seen data with its corresponding operator to avoid redundant processing.

OperatorSage is 46 times faster and 16% more accurate than a human. We run OperatorSage for 40 hours (at a cost of roughly $600), analyzing the operators of over 150,000 IPv4 addresses. This enables the largest study to date of individual IPv4 operators across several industries at higher risk of cyber attacks, including healthcare, government, and banking.

OperatorSage is designed to evolve alongside the rapid advancements in language models. By relying entirely on off-the-shelf LLMs, it inherits improvements in accuracy and capability without requiring architectural changes. While this work focuses on operator attribution for IPv4 services, the core design principles of OperatorSage can be extended to other domains with large volumes of service metadata, such as cloud storage. We open source OperatorSage under the Apache 2.0 license and hope that it helps reduce the burden on researchers tasked with manually analyzing vast, noisy datasets.

```
IP: 206.214.224.XXX
Reverse DNS:
starlink.lax.starlinkisp.net
Routing: 206.214.224.0/24 via
SPACEX-STARLINK (AS14593)
WHOIS: SpaceX Services, Inc.
HTTP 443/TCP:
   TLS Cert: Subject CN=*.1.oca.nflxvideo.net
   Cert. Issuer: Google Trust Services
```

Figure 1: Determining the Service Operator—Reverse DNS, routing data, and WHOIS point to Starlink (SpaceX) as the infrastructure provider of the IP. However, the TLS certificate is issued to `*.1.oca.nflxvideo.net`, indicating the service is operated by Netflix and merely hosted on Starlink infrastructure. We confirm with Netflix they operate the service.

```
IP: 62.141.48.XXX
Reverse DNS: keyhelp.maits.de
Forward DNS: mail.wenne.de, www.maits.de,
mail.mobilcut.com, mail.maits.de,
www.wenne.de, www.reha-sz.de,
sandbox.maits.de, mobilcut.com, ... (16
total)
Routing: 62.141.48.0/20 via KEYWEB-AS, DE
(AS31103)
TLS Certificate: Subject CN=
keyhelp.maits.de
Issuer C=US, O=Let's Encrypt, CN=R11
Names: keyhelp.maits.de
```

Figure 2: Indiscernible Operator Example—While the TLS certificate and reverse DNS indicate keyhelp.maits.de, the forward DNS lists an array of domains from seemingly unrelated sources. Combined with routing via KEYWEB-AS (AS31103) in Germany, these conflicting identifiers make it difficult to attribute a single responsible operator for the Internet-visible services.

2 Defining an IP-service Operator

We define the IP-service operator to be the individual or entity that has administrative control over the deployment, configuration, and ongoing management of an IP-based application service running on a network-accessible host. This party assumes responsibility for the behavior, security posture, and interaction of the application with the network and its users. Notably, the entity that manages the IP-based application services is often not the owner of the IP address, nor the BGP announcer. Simply put, we want to answer "Who is using this IP?"

Determining the true operator of a network service requires focusing on who controls the behavior and configuration of the application itself. For example, in the case of IP address 206.214.224.XXX in Figure 1, evidence points to Netflix as the service operator. Examining the services running on the host, HTTP over TCP on Port 443, contains references to `nflxvideo.net` in the TLS certificate. The appearance of Netflix-specific identifier in the TLS certificate suggests that Netflix is actively operating the service. Indeed, Netflix operates its own global CDN which deploys servers—called Open Connect Appliances (OCAs)—in ISPs' networks, like Starlink. Since our definition of the IP-service operator centers on the entity responsible for the ongoing management, configuration, and deployment of the network *application*, Netflix fits this role. We confirm with Netflix it is indeed the operator.

In contrast, other surface-level indicators could easily lead to incorrect conclusions about operator identity. The reverse DNS entry ("starlink.lax.starlinkisp.net"), the routing information through AS14593, and the WHOIS entry point to SpaceX's Starlink, the Internet Service Provider. However, Starlink's role is limited to allocating the IP address and providing Internet connectivity. They are not responsible for configuring the service or managing the application running on the device. Similarly, the HTTPS service reveals a certificate issuer Google Trust Services. While Google Trust Services is a Certificate Authority, they do not control how the device is ultimately set up or what services are exposed to the Internet. Thus, neither Starlink nor Google Trust Services can be considered the operator under our definition.

This example illustrates why careful examination of the service's operational behavior—rather than relying on ISP names, WHOIS, or DNS entries—is essential to accurately extract the true operator of an Internet-facing application. Furthermore, nearly 19K IPs present TLS certificates whose Subject Common Names (CN) match the domain pattern "∗.oca.nflxvideo.net." Given the large volume of matching certificates, manually extracting the operator repeatedly would be inefficient— and likely unnecessary—suggesting that a caching mechanism could greatly streamline this process.

In other cases, hints about the service operator may not be clearly visible in the service data. Certain network configurations—such as NAT (Network Address Translation) and port forwarding—can further complicate identification by allowing multiple independent entities to share a single public IP address. When this happens, the available service metadata may reflect a confusing or conflicting mixture of seemingly unrelated domains, certificates, or banners, making it impossible to reliably discern a single responsible operator.

For example, IP address 62.141.48.XXX contains descriptive information, but a definitive operator cannot be extracted. Figure 2 shows a subset of the results from an application-layer scan of 62.141.48.XXX conducted on March 19th, 2025. In this instance, the forward DNS records list a diverse array of domains—including entries from `wenne.de`, `maits.de`, `mobilcut.com`, and `keyhelp.maits.de`—while both the reverse DNS and TLS certificate consistently reference `keyhelp.maits.de`. Additionally, the IP is routed via KEYWEB-AS (AS31103) in Germany. This mix of conflicting indicators suggests that the IP address may be part of a shared hosting environment or multi-tenant infrastructure managed by different organizations, rather than being attributable to a single, discernible operator.

3 Identifiable Operators in the Wild

In this section, we present evidence that identifiable Internet service operators appear frequently in measurement data. Motivating the need for OperatorSage, we measure that it takes on average 1.6 minutes for a human to identify the operator of a service, and therefore would take over 100 consecutive days to identify the operators of only 100K (0.0023% of) IPv4 addresses. To inform the design of our system, we analyze what data fields are most often used to extract an operator from application layer data.

Methodology. We construct a set of 250 random IPv4 hosts with services for humans to manually identify the operator of. To do so, we use a Censys [7] snapshot of the full IPv4 address space from January 15th, 2025, and iteratively sample random addresses, discarding any that lack at least one responsive service. We assign two researchers to identify the operator of each IP. We instruct each researcher not to use AI, but allow them to search the Internet to verify a potential operator's identity (e.g., confirming a potential operator is not an obscure router manufacturer). If there is disagreement among two researchers, we employ a third researcher to resolve the disagreement. To quantify the level of agreement between researchers, we calculate Cohen's Kappa [9], a standard measure from qualitative research that assesses inter-rater reliability and agreement.

Results. Roughly 24% of the randomly generated IPs with services had an identifiable operator. On average, it takes 1.6 minutes for a human to identify the operator (or lack thereof) of an IP address using application layer data about the IP. It takes over 10 hours in aggregate for seven researchers to analyze 250 IPs, motivating the need for OperatorSage.

Overall, researchers exhibit substantial agreement (Cohen's Kappa > 0.79) when an operator is discernible. Thus, our annotation yields reliable labels, providing a solid foundation for downstream automated attribution. Disagreements most often occurred when one researcher judged the operator to be indiscernible while the other identified one—accounting for 13.3% of all cases. These typically involved services hosted on shared, multi-tenant infrastructures such as CDNs or large telecom networks, where clear ownership signals are often absent. Thus, when identifying operators, we consider success to be if OperatorSage performs at least with at least an 75% accuracy. Again, the goal of OperatorSage is to be at least as good as a human.

For the extracted operators, the majority (99%) of operator information is revealed in one of the following application-layer or networking attributes: DNS records[1] (56%), TLS certificates (36%), WHOIS record (20%), and HTTP data (3%). These results indicate that historical methodologies that simply rely on WHOIS to identify operators [14, 22, 23] are likely missing many opportunities to identify operators of services. Driven by the significant time cost of manual operator extraction, the complexity of the task, and the diverse set of fields that reliably signal operator identity, we design OperatorSage.

4 System Requirements

Given an Internet service, OperatorSage's goal is to identify the operator. The input of the system will be a list of IP addresses up to the entire IPv4 address space. The output of the system is a mapping of each IP address and its associated operator. Notably, the system's goal is to simply be at least as good at identifying operators as today's "state of the art" methodology: a human [13, 15].

To meet the objective, OperatorSage must satisfy the following constraints:

Fast. There are approximately 230.4 million IPv4 hosts on the Internet that offer services. OperatorSage must not only perform faster than a human, but must do so within a reasonable wall-clock time.

Accessible. OperatorSage must be accessible to the broader community of Internet measurement and vulnerability disclosing researchers. Most researchers do not have access to extensive proprietary training datasets, specialized infrastructure, or large datacenters filled with GPUs. Therefore, rather than expecting researchers to fine-tune or host their own large language models, OperatorSage leverages a callable API-based LLM.

Cost Efficient. OperatorSage must remain affordable for researchers. Callable API-based LLMs, if used inefficiently or indiscriminately across large datasets, can quickly incur

[1] Censys uses both DNS queries and proprietary methods to associate domains to their corresponding IP addresses.

costs of thousands of dollars. To maintain cost efficiency, OperatorSage must carefully limit the use of LLM APIs, invoking them only on a need-to-run basis to minimize expenses without sacrificing effectiveness. Additionally, OperatorSage must minimize both input and output token usage—by crafting concise prompts and requesting streamlined outputs—to further reduce API call costs without sacrificing necessary functionality.

Accurate. OperatorSage must be reliable and trustworthy. Large language models (LLMs) are known to occasionally hallucinate—producing incorrect or fabricated outputs—and to exhibit non-determinism, where the same query may yield slightly different answers across runs. Given these risks, OperatorSage must be carefully designed to maximize accuracy at every step, including through prompt engineering, result verification, and by limiting LLM usage to tasks where correctness can be systematically checked or controlled.

5 System Architecture

To identify IP service operators, OperatorSage uses a large language model to interpret application and network data. However, scaling large language models (LLMs) to quickly, cost efficiently, and accurately identify service operators across millions of IP addresses means OperatorSage cannot naively apply the large language model to every IP address. Rather, OperatorSage runs in a 3-stage pipeline. After looking up the application layer and associated network data of an IP address, OperatorSage (1) prunes application layer data to reduce the cost of running an LLM and increase overall LLM accuracy, (2) runs the LLM with a specific prompt that uses structured input, examples, and structured output to increase scalability and accuracy, (3) caches relevant application layer data with its identified operator (or lack thereof) to avoid re-running the pipeline on identical data, thereby decreasing costs and increasing scalability. Figure 3 outlines our current system design.

5.1 Prune Phase

After collecting application-layer and network attributes (e.g., WHOIS) for each IP address, OperatorSage executes a prune phase to reduce the volume of data passed to the LLM, thereby lowering both computational cost and improving accuracy. This phase is motivated by the fact that not all application-layer information is meaningful in extracting an operator. For example, cryptographic keys, key exchange algorithms and host fingerprints are not expected to hold semantically identifiable information about the operator of a service.

The prune phase implements a set of automated heuristics to reduce the dataset. The heuristics integrate a mix between regular expressions (RegEx) and alphanumeric ratio to detect and remove human indiscernible/non-meaningful/high-entropy information from the dataset. We show that RegEx filters are used to filter for timestamps, while alphanumeric ratio thresholds are used to help flag and exclude cryptographic fingerprints (e.g., SHA-256 hashes), which typically contain a high proportion of hexadecimal characters. We target the creation of pruning heuristics towards the five attributes that revealed the most operator information (Section 3): HTTP, DNS, WHOIS, TLS, and ASN (Autonomous System Number).

DNS Names. DNS pruning proceeds in three stages, with further technical details, exact reg-ex patterns, and examples provided in Appendix A.5.2.

The first stage performs normalization and tokenization. OperatorSage discards empty records and entries that exactly match IPv4 patterns, then cleans and segments DNS names by collapsing repeated delimiters and splitting on common characters like dots and hyphens. OperatorSage removes fragments that match IP-like sequences and frequent structural keywords (e.g., “static,” “deploy,” “dynamic”, see Appendix A.5.2 for more information) when they appear between delimiters. This helps isolate potentially identifying components while avoiding the over-removal of legitimate domain terms.

In the second stage, OperatorSage iteratively strips high-frequency tokens across the dataset. Tokens that are short, numeric, or overly generic are identified through frequency counts and removed from all entries. OperatorSage reassembles and re-filters names at each iteration, dropping those that become empty or structurally meaningless (e.g., just a top-level domain or residual numbers). This approach avoids hard-coded exclusion lists and adapts to different infrastructure naming conventions.

Finally, the third stage applies pattern-based filtering to remove low-entropy or hash-like names. OperatorSage identifies entries dominated by alphanumeric strings with limited semantic value and remove them from the dataset. After this multi-phase pruning, DNS names that retain structured, human-readable tokens—such as university or enterprise subdomains—are passed to the LLM for operator inference.

TLS Certificates. We use the Common CA Database [24] to identify and filter out widely trusted Certificate Authorities (CAs) from TLS common names, as these issuers do not reflect the actual operator of the IP address.

ASN Names. We cross-reference AS numbers with ASdb to identify and filter ASN names owned by ISPs, which are typically intermediaries, not end operators.

HTTP. We retain only responses with successful status codes (<300), discarding redirects and errors which typically lack operator-specific context.

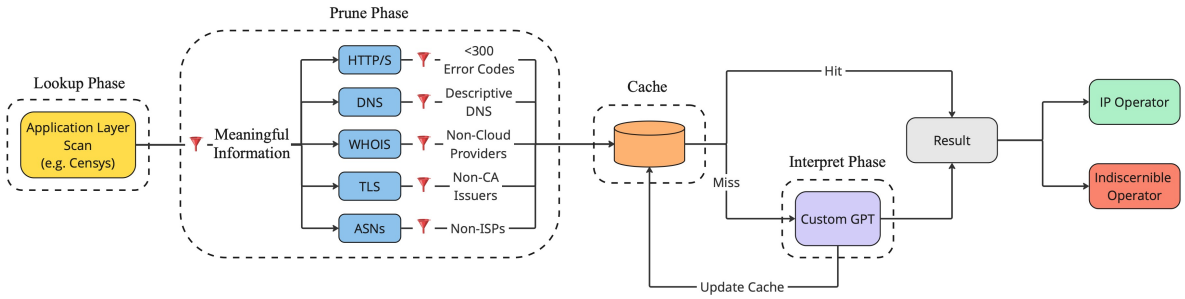


Figure 3: OperatorSage Pipeline—OperatorSage begins with conducting an application layer scan on every input IP. After the lookup, OperatorSage prunes the output of the scan for meaningful information informed by Section 3, mirroring how a human would, for example, look for error codes below 300 in HTTP(S) or would seek out non-Certificate Authorities. Subsequently, meaningful information is sent to a cache to check if an operator has been extracted for this set information before and thus does not need to be routed to an LLM (Section 5.3). If there is a cache miss, the meaningful information is sent to an LLM with an instructive prompt (Section 5.2). The LLM outputs whether it could extract an operator or not, subsequently filling in the cache.

WHOIS. We apply RegEx filters to detect and remove entries tied to common cloud hosts or ISPs, while retaining records that may reflect direct organizational ownership. The exact RegEx we use for WHOIS filtering is given in Appendix A.5.1.

High Entropy Pruning. We use alphanumeric ratio thresholds to detect and exclude non-human-readable data such as cryptographic fingerprints (e.g., SHA-256 hashes), which typically consist of dense hexadecimal character sequences. These forms of information—hashes, fingerprints, and other machine-generated identifiers—tend to exhibit high alphanumeric ratios due to their structured character composition. In contrast, human-discernible fields contain a more varied character set, including letters, punctuation, and whitespace, resulting in lower alphanumeric ratios.

In Appendix A.5.3, we plot the distribution of alphanumeric ratios for all hash-like values in our test set alongside human-interpretable fields. Since human-readable fields are clustered between a 0 to 0.35 alphanumeric ratio, but hashes are clustered around 0.9, we use 0.9 as a threshold to distinguish meaningful, human-interpretable information from machine-generated artifacts (i.e., we filter out fields that show alphanumeric ratios higher than 0.9). We also evaluated Shannon entropy as a discriminator, but found that human-readable fields often display high entropy as well, limiting its effectiveness. Alphanumeric ratios, by contrast, provide a more reliable signal for filtering non-human-readable content.

Garbage Collection. A final garbage collection pass eliminates any empty fields produced by the previous pruning steps. This step not only improves processing efficiency but also reduces the number of input tokens sent to the LLM.

5.2 Interpret Phase

The Interpret Phase of OperatorSage uses an LLM to infer the operator of an IP based on application-layer and network metadata. To maximize reasoning accuracy, we follow prompt design best practices informed by recent literature on prompting strategies for LLMs [18], including: (1) providing a clear and scoped task definition, (2) formatting structured input, (3) offering specific constraints, and (4) using few-shot examples. The full prompt is shown in Appendix A.3.

Task Specification. We begin the prompt with a precise description of the task and a scoped definition of an "IP Service Operator"—the entity responsible for configuring or managing the Internet-visible services on an IP address. Crucially, we distinguish this from infrastructure providers (e.g., cloud hosts or AS owners) and certificate authorities, which are often mistakenly treated as service operators. The prompt explicitly states: *"The IP Service Operator is the entity responsible for Internet-visible services on a given IP [...] not the ISP, CDN, Hosting Provider, or TLS Certificate Issuer."*

Input Structuring and Reasoning Heuristics. To support consistent reasoning, the prompt imposes a structured analytical process: identify and discard likely non-operators, validate any remaining entities, and explain the reasoning used in the final decision. We write: *"First identify what is most likely not an operator. Then use the remaining information to identify if an operator exists. Really think about the decisions and conclusions you are making and self-assess*

whether your conclusions actually make sense." This encourages chain-of-thought reasoning and discourages premature or shallow inference.

Guardrails and Fallback Behavior. The prompt explicitly discourages hallucination and overconfidence by instructing the model to default to "Indiscernible" when evidence is weak, ambiguous, or purely infrastructural: *"Default to 'Indiscernible' [...] when only ISP/hosting/CDN information is available with no indication of the actual customer."* Additional constraints reinforce groundedness and reproducibility, such as: *"ONLY use information from the attached JSON file and do not utilize previous context from another IP."*

Few-Shot Prompting. Few-Shot Prompting is the process of feeding in examples to the model along with the instruction that provide example inputs and their expected output. We apply few-shot prompting using two labeled examples. One example includes sufficient metadata (e.g., domain names, certificate subject fields) to attribute the operator (e.g., "City of Rocklin, CA"), while the other reflects an "Indiscernible" case with only infrastructure signatures. These examples are not reused in evaluation and are chosen to span the key decision boundary. In Section 6.2, we evaluate how performance varies with the number of few-shot examples.

5.3 Cache

Operators frequently manage multiple IP addresses, often with overlapping content that can reveal the operator's identity. Indeed, as seen in Section 3's, looking up the extracted Netflix domain from the TLS certificate in Censys results in nearly 19K IP addresses with the same Subject Common Name. Motivated by this example, OperatorSage uses a caching mechanism to improve scalability and reduce operational costs. The cache maps previously observed data to the corresponding operators that were identified earlier, allowing the system to efficiently recognize recurring patterns without redundant computation.

The pruned IP address data is queried against a pre-configured cache to lookup IP operator mappings for similar IP addresses. The cache contains mappings from each individual feature field—DNS, WHOIS, TLS, and ASN—to its associated IP operator. Moving to the next phase is contingent of the cache containing similar IP address to operator mappings while potentially reducing the financial burden of invoking the interpret phase. The cache and interpret phase run in a positive feedback loop wherein cache hits continually strengthen future predictions and reduce reliance on the LLM. A cache lookup works by simply matching one feature (e.g., DNS record).

There are two potential challenges to address when designing the cache: (1) dealing with conflicting cache hits (i.e., when a host contains two features that map to different operators) and (2) ensuring egregiously erroneous operator mappings do not propagate through the cache.

Conflict Avoidance. Luckily, conflicting cache hits appear exceedingly rare, thereby reducing the need for conflict resolution in the cache. In Section 6.3, we show that the vast majority (98%) of the time hosts contain no more than one feature that produces a cache hit, and in the rare case they do, both features map to the same operator. Our cache matches features in the following order—WHOIS, ASN, TLS, and HTTP—not because order is critical, but to provide a consistent and deterministic implementation.

Avoidance of Propagation Error. To prevent egregious operator mapping errors from spreading through the cache, we proactively filter out overly common features that may point to hyperscale infrastructure providers (e.g., Amazon, Comcast) rather than true IP operators. Our approach relies on the observation that large-scale providers (e.g., cloud platforms, ISPs, and certificate authorities) exhibit disproportionately large footprints across the IPv4 space. Without this safeguard, mappings from generic or infrastructure-heavy WHOIS or ASN entries could mistakenly propagate through the cache and dominate subsequent classifications. By aggregating the number of IPs associated with each feature, we identify and exclude high-frequency entities from being cached as operators. This ensures that the cache does not inherit misleading mappings simply because a particular value (e.g., WHOIS entry for a CDN or TLS cert from a large CA) is common.

5.4 Implementation

The system is designed as a modular pipeline that integrates external data sources, language models, and a scalable caching layer. Implemented primarily in Python, the system is built for efficiency, scalability, and future extensibility as a Python package. OperatorSage code is all open sourced under the Apache 2.0 license and is available at [anonymized for submission].

Lookup Phase. To retrieve application-layer data associated with an IP address, we utilize the Censys platform and its official Python SDK. Specifically, the `api.lookup` feature is employed to query IP addresses and extract application-layer scan data.

Prune Phase. After retrieval, the raw application-layer data is pruned to retain only relevant fields (Section 5.1). Pruning logic is written in Python and pruned data is stored locally in JSON format for further processing.

Cache. The caching layer is implemented using Google BigQuery [8], a cloud-hosted data warehouse service. The database schema is designed to include fields for all relevant

IP operator indicators, enabling fast lookups and convenient prepopulation using entire Censys snapshots. Several SQL scripts are used to configure, query, and maintain the cache.

Interpret Phase. The interpret phase is implemented using the OpenAI `o3-mini` model API, which analyzes the pruned application-layer data to generate operator attribution insights. As of May 2025, `o3-mini` is the model with the most advanced reasoning capabilities that allows for files of data to be uploaded. In Section 6.2, we empirically justify the selection of this model by comparing it to other models. We invoke the OpenAI Assistants API via the Python `beta_threads` endpoint to instantiate a custom assistant configured by our few-shot instructions. To manage throughput and isolate each analysis, we write the pruned JSON payload to a temporary file, spawn a background thread that reads the file, streams it to the assistant, and writes the assistant's `JSON`-formatted response to a second temporary file. This file is then parsed by our pipeline to extract operator attributes. We evaluate alternative models Section 6.

6 Evaluation

In this section, we evaluate OperatorSage's ability to accurately, cost-efficiently, and quickly identify the operators of a service. We first focus on evaluating individual stages of the pipeline—thereby justifying our design decisions—and finish by evaluating the entire pipeline against our set of manually extracted ground truth. OperatorSage is 46 times faster and overall more accurate at identifying operators than humans.

Methodology. We evaluate the cost, runtime, and accuracy of individual design decisions using two datasets: a small validation set (N=50) sampled using the methodology described in Section 3, and a larger test set (N=250) introduced in the same section. We use the validation set to evaluate all design decisions, and the test set to evaluate the final pipeline (Section 6.4). Due to the non-deterministic nature of LLMs, we run at least three trials of every experiment and report average values of metrics, unless otherwise stated.

Given the inherent ambiguity in some operator attribution cases, we employ multiple metrics to assess both accuracy and appropriate abstention. Across our evaluation we define the true positive rate as the accuracy of extracting an operator when one is present (which we will denote as Operator %) and the true negative rate as the accuracy of concluding there is no operator to be extracted, denoted as Indiscernible %. We report a *Conservative Accuracy*, which reflects our design preference to abstain from classification rather than risk incorrect operator attribution. This metric measures the proportion of correct predictions among all outputs made by the system, not penalizing for false negatives. It is defined as:

$$\text{Conservative Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP}}$$

where TP is the number of true positives, TN is the number of true negatives, and FP is the number of false positives. This formulation ensures that OperatorSage is not penalized for abstaining (i.e., producing no output) when the operator is discernible. Recognizing that conservative accuracy prioritizes caution over coverage, we present it alongside traditional accuracy metrics.

6.1 Pruning

Pruning dramatically reduces LLM input size and processing cost without compromising accuracy. We evaluate the pruning stage on a January 15 snapshot of the full IPv4 scan from Censys. Figure 4 shows that after pruning, roughly 80% of IPs have at least an 80% cost reduction.

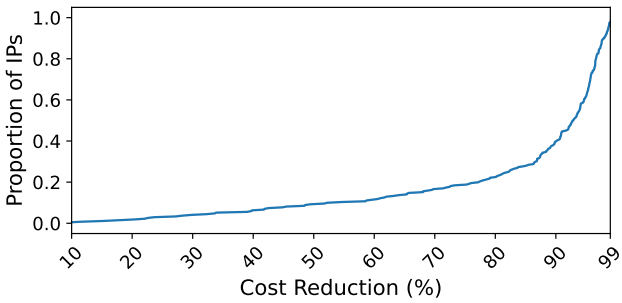


Figure 4: Cost Reduction Post Pruning—Nearly 80% of IPs saw greater than an 80% input-token cost reduction post pruning.

IP Attribute	Filter Out	% IP Attributes Filtered Out
WHOIS	Cloud, ISP	71.1% (169.3 M / 238.0 M)
ASN	Cloud, ISP	75.3% (179.4 M / 238.0 M)
TLS	Cert Auth	49.3% (74.9 M / 151.8 M)
DNS	Cloud, IP Addr	40.2% (50.2 M / 124.9 M)
HTTP	< 300 Error	48.8% (241.2 M / 493.8 M)

Table 1: **Prune Stage Filtering**— WHOIS metadata is pruned out the majority of the time. The denominators represent the number of IP attribute occurrences across all of IPv4 (i.e., every IP address only has one WHOIS record, but may have more than one HTTP service).

Table 1 summarizes how much data is discarded across five structured fields. WHOIS and ASN fields contribute the most to reductions, with 71% and 75% of entries removed, respectively. These two alone filter out more than 169M and

179M records. TLS and HTTP pruning eliminate 49% and 48% of their respective records, removing another 300M+ combined. Even DNS filtering removes 40%, filtering out over 50M entries. Passing the pruned dataset to our GPT-based classification pipeline does not yield notable gains or reduction in accuracy, however is valuable for cost reduction.

6.2 Interpret Phase (LLM)

OpenAI's o3-mini LLM model using 5 IPs per query and 2-shot prompting produces the best balance between accuracy, time, and cost.

Model Selection. To select the optimal model for the interpret phase, we compare the accuracy, computational cost, and processing time of various language models. Specifically, we evaluate three of the latest-released models: Anthropic Claude [1] (API invoked), OpenAI's GPT-4o (a custom, non-API invoked GPT), and OpenAI's o3-mini (API invoked). We use the default parameters for settings such as the temperature and top_p, which control the randomness of the output and the threshold for considering tokens respectively. Table 2 shows that OpenAI's o3-mini achieves the highest overall accuracy, conservative accuracy, and operator-specific accuracy. Notably, Claude experiences significant rate limiting, thereby operating 20 times slower than OpenAI's models. All models cost roughly the same.

Parallelizing. We also test the impact of parallelizing the number of IPs per LLM query with the o3-mini model. Table 3 shows that batching 5 IPs per query at a time provides provides better speed and reduces input tokens while not sacrificing accuracy.

Few-Shot Prompting. We evaluate how the number of in-context examples during prompting affects accuracy and cost. Using the o3-mini model, we test three configurations: a single example demonstrating a true positive operator identification (City of Rocklin, described in Section 5.2); two examples—one operator identification and one indiscernible; and three examples with two operator identifications (the additional example using a different IP attribute) and one indiscernible. As shown in Table 4, the 2-shot configuration yields the best balance of accuracy and cost.

6.3 Cache

We evaluate both the likelihood of cache conflicts and the overall coverage provided by the cache. Conflicting feature-to-operator mappings are rare. In both our validation and test sets, 98% of hosts contain at most one feature that results in a cache hit. In the rare cases where multiple features do match, they almost always map to the same operator.

To assess cache coverage, we first run all OperatorSage stages—except for the cache—for 40 hours. In this time, OperatorSage evaluates operators for nearly 90K IP addresses, from which it identifies 13.7K unique operators. Second, we populate the cache with feature mappings from OperatorSage's first run. Third, we apply the resulting cache to a full IPv4 snapshot from Censys (roughly 200 million IPs with public services) to identify additional hosts with matching features. It takes roughly 1.5 hours to complete the matchings. The cache resolves identifiable operators for 9K more IPs, increasing the number of IPs OperatorSage identifies operators for by 66%. Notably, the overall number of unique operators does not increase, as the newly matched IPs share features with previously attributed IP services. However, the cache increases the number of IPs labeled as indiscernible (by 70K), as it surfaces more hosts whose attributes originally failed to indicate any operator. Thus, the cache enables OperatorSage to scale its analysis to 150K IPs within 40 hours.

6.4 Full Pipeline

We compare OperatorSage's overall performance to the commonly used methodology to extract operator identities—human analysis—in Table 5. Unsurprisingly, OperatorSage is 46 times faster at identifying an operator compared to a human: OperatorSage spends less than 2 seconds per IP, compared to a human spending roughly 81 seconds.

OperatorSage achieves 87.0% overall accuracy, significantly outperforming the 74.9% accuracy of human annotators. In contrast, humans lead in conservative accuracy (98.3% vs. 93.2%). Conservative accuracy rewards a cautious labeling strategy: by abstaining when uncertain, human experts almost eliminate misclassifications at the expense of coverage.

This trade-off is most pronounced in the Operator class: OperatorSage correctly attributes 70.0% of discernible operators versus just 42.3% for humans. On the other hand, humans score a marginally higher 97.8% on the Indiscernible class compared to OperatorSage's 92.7%. The conservative accuracy metric thus captures the value of human wariness—mitigating false attributions in ambiguous cases—while also highlighting its cost in lost coverage. OperatorSage, by contrast, balances these objectives: it confidently labels more operators (boosting overall coverage) yet still leverages our abstention threshold to preserve high conservative accuracy.

In the next section, we provide a sneak peak into what we can learn about the operators that OperatorSage identifies.

7 Analysis of Operators

In this section, we show that OperatorSage excels at identifying operators who rent infrastructure from large cloud providers. Our results are derived from running OperatorSage on 150,000 IP addresses, which resulted in 13,777 unique,

Model / Pipeline	Time per IP (s)	Accuracy (%)	Conservative Acc. (%)	Operators (%)	Indiscernible (%)
Claude	23.85	61.1	62.2	36.7	97.2
GPT-4o	1.10	82.6	87.5	52.9	93.3
o3-mini	1.45	87.5	93.2	69.5	92.7

Table 2: **Model Selection**—OpenAI's o3-mini produces the best balance between accuracy and time.

#IPs	Time (s)	Acc. (%)	Tokens
1	4.02	88	12,998
5	1.65	92	4,280
10	1.41	90	2,487
20	1.24	88	1,143

Table 3: **Parallel Querying**—Batching 5 IPs per query at a time provides provides better speed and reducing input tokens while not sacrificing accuracy.

operator-identifiable addresses, as described in Section 6.3. To avoid bias toward clusters of identical services, we restrict our analysis to IPs whose operators were identified directly by the LLM, excluding those resolved via cache.[2]

Operator Attribution Within Cloud Networks. The autonomous system with the most number of identifiable operators is Amazon (ASN 16509), which is also responsible for the most number of IPv4 addresses with public services. We identify 1,695 unique operators inside Amazon, as well as 1,081 operators in Akamai (ASN 16625), 562 in Amazon's secondary network (ASN 14618), and 322 in Digital Ocean (ASN 14061). Notably, not a single IP address in this set has a WHOIS record that would identify the operator. We manually analyze all WHOIS records across all fields to confirm that not even the `descr` field annotates customer information.

Cloud networks hold a diverse set of operators. For example, OperatorSage identifies a U.S politician's campaign management infrastructure in Amazon, by using the HTTP header of the campaign website. OperatorSage identifies a host likely operated by JPMorgan Chase, through the extraction and interpretation ofthe `jpmc` string from a forward DNS record. Overall, operator identifiable information is found across 3,185 autonomous systems.

Identifiable Operators Span Many Industries. Roughly 6.5% (909) of operators belong to a critical infrastructure sector, including legal, banking, government, and healthcare.

We organized the 13,777 operator strings into eleven high-level categories using keyword matching[3]. Table 6 shows a breakdown of the categories. Notably, operators from critical infrastructure sectors are particularly sensitive to software vulnerabilities (e.g., CVEs). Timely identification of operators in these sectors is essential for responsible disclosure and effective mitigation, as unpatched systems in critical infrastructure can lead to widespread harm.

Case Study of Healthcare Sector. Healthcare providers rely on public clouds. Of the 125 operators under the healthcare category, we find almost half (55) use five cloud providers including Amazon, Akamai, Google, and Microsoft to host their services. We manually look through these operators and find 8 hospitals in the U.S, where 4 contained the hospital identifier in the IP WHOIS record, while the other 4 only in the TLS certificate. These hospitals and healthcare providers include remote patient monitoring platforms, online pharmacies, and health records portals, which likely handle sensitive patient data and are subject to strict regulatory requirements (e.g., HIPAA in the U.S.). Given the healthcare sector's exposure to ransomware and data breaches, it is important to timely notify its operators of any security vulnerabilities and anomalous traffic. OperatorSage contributes to this goal by swiftly identifying the responsible healthcare organizations behind cloud-hosted IP addresses.

Operator Use of Domains. Of the 13,777 identified operator, 48.1% (6,626) were extracted from domain names. Roughly 1.2% (1,676) contained non-English characters. We further analyzed the 255 top-level domains (TLDs) for the 6,626 domain names and found 115 country code top-level domains (ccTLDs). The ccTLDs that corresponds to the most operators were .cn (168), .de (129), .jp(126), .ru (94), and .br(80).

8 Related Work

A range of prior work has aimed to attribute Internet-facing services to the entities responsible for operating them. Broadly, these efforts fall into three categories: (1) manual analysis of application-layer metadata, (2) machine learning–based

[2]Because the cache only maps previously seen IPs to known operators, it does not introduce new uniquely identifiable operators.

[3]To classify operator strings into industry sectors, we first define high-level categories that include critical infrastructure such as healthcare, banking, and government. We curate a list of keywords that likely to appear in the operator/entity names of each category (e.g., "hospital", "clinic", "health" for healthcare sector).

	Acc (%)	Cons. Acc (%)	Operator (%)	Indisc. (%)	Time (s/IP)	Total Cost ($)
0-shot	86	92.9	66.0	94.0	–	–
1-shot	86	97.7	80.0	86.7	1.493	$0.081
2-shot	90	95.7	75.0	92.9	1.550	$0.121
3-shot	92	100.0	100.0	90.7	1.647	$0.132

Table 4: **Few-shot prompting performance**—Increasing prompting increases performance, generally at the expense of cost and time.

Model / Pipeline	Cost per IP ($)	Time per IP (s)	Accuracy (%)	Conservative Acc. (%)	Operators (%)	Indiscernible (%)
OperatorSage	0.0063	1.73	87.0	93.2	70.0	92.7
Human	-	81	74.9	98.3	42.3	97.8

Table 5: **OperatorSage (Full Pipeline) Performance**— Evaluation of cost, runtime, and prediction accuracy. Conservative accuracy excludes false negatives, reflecting our preference to abstain rather than misclassify. "Operators" and "Indiscernible" indicate class-specific accuracy for discernible and indiscernible operator predictions, respectively.

Category	Count
Technology	683
Education	396
Legal	376
Bank	214
Media	194
Government	193
Cloud/Hosting	144
Healthcare	125
Telecom/ISP	60
Non-Profit	31
Other	11361

Table 6: **Identifiable Operators Span Many Industries**—Roughly 900 operators fall into critical infrastructure sectors, including legal, banking, government, and healthcare.

classification of infrastructure, and (3) more recent attempts to apply large language models (LLMs) for scalable attribution. While all three lines of work offer valuable insights, they differ significantly in terms of scalability, granularity, and attribution goals.

Traditionally, service attribution has relied on manual interpretation of metadata. *Hitchhiking* [13] identifies exposed Starlink user terminals by inspecting TLS certificate details, while Liu et al. [15] explores mail exchange (MX) record mappings to associate email infrastructure with operating organizations. El Yadmani et al. [12] attempt to attribute S3 buckets to real-world entities using access patterns and metadata but conclude that responsible disclosure at scale is infeasible. Similarly, Sanchez-Rola et al. [20] study cookie tracking behaviors to map web services to backend operators. In all of these efforts, attribution hinges on researchers manually interpreting ambiguous or noisy signals to link infrastructure to its operator.

To overcome the scalability limitations of manual analysis, other studies have applied machine learning techniques to classify Internet infrastructure. These efforts automate classification, but generally limit their scope to broad taxonomies (e.g., ISP vs. university), rather than resolving specific operator identities. Dimitropoulos et al. [10] developed a taxonomy for Autonomous Systems (ASes) based on BGP behavior, which Cai et al. [6] later extended to group ASes by organizational ownership. The ASdb platform [26] replaces these heuristic approaches with text-based classifiers trained on public registry data. Additional work by Baumann and Fabian [3] applied n-gram analysis to classify ASes by industry, while Sebastián et al. [21] leveraged NLP techniques to extract operator information from TLS certificates and website content.

More recently, IPdb [16], for example, uses an LLM to categorize IP addresses into sectors using features like RDNS, WHOIS, and TLS metadata. While useful for high-level classification, such approaches stop short of identifying the actual entity running the service. In contrast, our work targets fine-grained operator attribution, emulating human reasoning to extract the names of organizations or individuals managing specific deployments—even in the presence of hosting intermediaries, address leasing, or CDN layers.

9 Conclusion

We introduced OperatorSage, a system that automates the answering of "Who is using this IP?" OperatorSage uses off-the-shelf LLMs to ensure ease-of-deployment, scalability, and flexibility. OperatorSage is 46 times faster than a human, allowing it to attribute operators for over 150,000 IPv4 addresses in about 40 hours. While OperatorSage accelerates operator identification by over an order of magnitude, its current design is optimized for targeted IP sets rather than the entirety of IPv4. Scaling to full coverage remains an open challenge and an active area of ongoing work. Nevertheless, OperatorSage is an important step towards scalable, automated operator attribution for the Internet measurement community.

References

[1] Anthropic. 2023. Claude AI. https://www.anthropic.com/index/introducing-claude.

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, et al. 2017. Understanding the Mirai botnet. In *USENIX Security Symposium*.

[3] A. Baumann and B. Fabian. 2014. Who Runs the Internet? Classifying Autonomous Systems into Industries. In *Proceedings of the 2014 International Conference on Web Information Systems and Technologies*. SciTePress, 79–90.

[4] Aaron J. Burstein. 2008. Conducting Cybersecurity Research Legally and Ethically. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 08)*. USENIX Association, San Francisco, CA. https://www.usenix.org/legacy/event/leet08/tech/full_papers/burstein/burstein_html/

[5] Jack Cable, Drew Gregory, Liz Izhikevich, and Zakir Durumeric. 2021. Stratosphere: Finding vulnerable cloud storage buckets. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*. 399–411.

[6] Xue Cai, John Heidemann, Balachander Krishnamurthy, and Walter Willinger. 2010. Towards an AS-to-organization map. In *Proceedings of the ACM Internet Measurement Conference (IMC)*. ACM, 199–205.

[7] Hudson Clark, Jeff Cody, Elliot Cubit, Zakir Durumeric, Matt Ellison, Liz Izhikevich, and Ariana Mirian. 2025. Censys: A Map of Internet Hosts and Services. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[8] Google Cloud. 2024. *Google BigQuery*. https://cloud.google.com/bigquery.

[9] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[10] Xenofontas Dimitropoulos, Dmitri Krioukov, George Riley, and kc claffy. 2006. Revealing the Autonomous System taxonomy: The machine learning approach. In *Passive and Active Network Measurement Workshop (PAM)* (Adelaide, Australia).

[11] Ben Du, Romain Fontugne, Cecilia Testart, Alex C. Snoeren, and kc claffy. 2024. Sublet Your Subnet: Inferring IP Leasing in the Wild. In *Proceedings of the 2024 ACM Internet Measurement Conference (IMC '24)*. Association for Computing Machinery, Madrid, Spain, 328–336. https://doi.org/10.1145/3646547.3689010

[12] Soufian El Yadmani, Olga Gadyatskaya, and Yury Zhauniarovich. 2025. The File That Contained the Keys Has Been Removed: An Empirical Analysis of Secret Leaks in Cloud Buckets and Responsible Disclosure Outcomes. In *Proceedings of the 2025 Conference on Cloud Security*. https://project-theseus.nl/publication/2025/the-file-that-contained/the-file-that-contained.pdf

[13] Liz Izhikevich, Manda Tran, Katherine Izhikevich, Gautam Akiwate, and Zakir Durumeric. 2024. Democratizing LEO Satellite Network Measurement. *Proc. of the 8th ACM POMACS* (2024).

[14] Frank Li, Gilbert Wondracek, Marco Balduzzi, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. 2016. You've Got Vulnerability: Exploring Effective Vulnerability Disclosure. In *25th USENIX Security Symposium*. USENIX Association.

[15] Enze Liu, Gautam Akiwate, Mattijs Jonker, Ariana Mirian, Stefan Savage, and Geoffrey M. Voelker. 2021. Who's Got Your Mail? Characterizing Mail Service Provider Usage. In *Proceedings of the 2021 ACM Internet Measurement Conference*. 1–15. https://doi.org/10.1145/3487552.3487820

[16] Author Name and Co author Name. 2023. IP Sector Categorization Using Machine Learning Techniques. In *Proceedings of the Conference on Network and Service Management (CNSM)*. https://openreview.net/pdf?id=wzeZ2kp7jS

[17] Arman Noroozian, Jan Koenders, Eelco Van Veldhuizen, Carlos H Ganan, Sumayah Alrwais, Damon McCoy, and Michel Van Eeten. 2019. Platforms in everything: analyzing {Ground-Truth} data on the anatomy and economics of {Bullet-Proof} hosting. In *28th USENIX Security Symposium (USENIX Security 19)*.

[18] OpenAI. 2025. Reasoning Best Practices. https://platform.openai.com/docs/guides/reasoning-best-practices Accessed: 2025-03-19.

[19] Eric Pauley, Paul Barford, and Patrick McDaniel. 2023. The CVE Wayback Machine: Measuring Coordinated Disclosure from Exploits against Two Years of Zero-Days. In *Proceedings of the 2023 ACM on Internet Measurement Conference*.

[20] Iskander Sanchez-Rola, Matteo Dell'Amico, Davide Balzarotti, Pierre-Antoine Vervier, and Leyla Bilge. 2021. Journey to the Center of the Cookie Ecosystem: Unraveling Actors' Roles and Relationships. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE.

[21] Silvia Sebastián, Raluca-Georgia Diugan, Juan Caballero, Iskander Sanchez-Rola, and Leyla Bilge. 2023. Domain and Website Attribution beyond WHOIS. In *Proceedings of the 39th Annual Computer Security Applications Conference (ACSAC '23)*. Association for Computing Machinery, New York, NY, USA, 124–137. https://doi.org/10.1145/3627106.3627190

[22] Ben Stock, Giancarlo Pellegrino, Christian Rossow, Martin Johns, and Michael Backes. 2016. Hey, you have a problem: On the feasibility of {Large-Scale} web vulnerability notification. In *25th USENIX Security Symposium (USENIX Security 16)*. 1015–1032.

[23] Florian Streibelt, Martina Lindorfer, Seda Gürses, Carlos Hernández Gañán, and Tobias Fiebig. 2023. Back-to-the-Future Whois: An IP Address Attribution Service for Working with Historic Datasets. In *Passive and Active Measurement: 24th International Conference, PAM 2023, Virtual Event, March 21–23, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 13882)*. Springer, 209–226. https://doi.org/10.1007/978-3-031-28486-1_10

[24] The Linux Foundation. 2024. *Common CA Database (CCADB)*. Accessed: 2025-05-15.

[25] Yiluo Wei, Dennis Trautwein, Yiannis Psaras, Ignacio Castro, Will Scott, Aravindh Raman, and Gareth Tyson. 2024. The Eternal Tussle: Exploring the Role of Centralization in IPFS. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 441–454. https://www.usenix.org/conference/nsdi24/presentation/wei

[26] Maya Ziv, Liz Izhikevich, Kimberly Ruth, Katherine Izhikevich, and Zakir Durumeric. 2021. ASdb: A System for Classifying Owners of Autonomous Systems. In *Proc. of the 21st ACM IMC*.

A Appendix

A.1 Ethics

Our work does not raise any ethical issues. We do not perform any new Internet scans and use existing data available to the public.

A.2 Zero-Shot Instruction

This is the exact instruction we gave the LLM for the Zero-Shot Instruction experiments:

```
Core Purpose:
You are an AI assistant specialized in determining the IP
Service Operator for an IP address given a JSON file from
Censys containing information about the various ports and
services from that IP. The IP Service Operator is the
entity responsible for Internet-visible services on a
given IP of which they can directly control and configure.
In most cases, IPs are owned by the ISPs, CDNs, Hosting
Providers, etc. and the IP Service Operator would be the
CUSTOMER of that entity that owns the IP.
Specifically Distinguishing a IP Service Operator:
The following list below are NOT the IP Service Operator
unless proven otherwise:
  • Autonomous System (AS) associated with the IP
    address
  • Any Certificate Issuers Entities
  • Cloud providers or Hosting infrastructure providers
    or Infrastructure Leaser
  • Any third-party service providers or
    telecommunication organizations that do not have
    direct control over the Internet services running on
    an IP
Critical Thinking Requirements:
  • Explain your chain of thought
  • First Identify what is most likely not an operator
  • Then use the remaining information to identify if an
    operator exists
  • Really think about the decisions and conclusions you
    are making and self-assess whether your conclusions
    actually make sense
Important Notes:
  • IP Service Operators tend to be present through
    representations of domains in the metadata. In the
    case that a domain is identified that may
    potentially be the operator. Make sure to validate
    and confirm that the operator is valid.
  • Default to "Indiscernible" for the IP Service
    Operator when only ISP/hosting/CDN information is
    available with no indication of the actual customer
    (IP Service Operator), the data only shows technical
    details without organizational identifiers, and
    there's no clear evidence of who operates the IP
    services. Additionally, be cautious with networking
    equipment: Router/modem services usually indicate
    infrastructure or hardware (not the IP Service
    Operator).
  • A human should be able to understand the IP Service
    Operator output so they can identify the associated
    entity. If the IP Service Operator is not readable
    or can't be easily verified by a human such as an
    unknown id or unknown domain, then default to
    "Indiscernible".
  • If the IP Service Operator is an acronym that is not
    commonly known, leave it in that form without
    unravelling it.
Important constraints:
  • ONLY use information from the attached JSON file and
    do not utilize previous context from another IP
  • Rely on the JSON data and your pre-existing
    knowledge about internet services and do not attempt
    to hallucinate or come up with a reasoning without
    strongly supported data from the JSON or your
    knowledge. Make sure to verify and validate any
    classification
Sample Examples: (NOTE: I will provide you only a snippet
of the JSON for a given example where it contains
information about some of the Operators. These examples
are not necessarily representative of all the IP
addresses and DO NOT bias by only focusing on the
metadata that is provided below to find/classify the
operator).
Final Output: Output only the IP address, IP Service
Operator in the JSON format specified.
```

The Important Notes section is specifically focused on analyzing and validating domains from the input. This is because we noticed in previous iterations of testing the LLM, it would frequently discard and make general conclusions about domains parsed from the input: this would lead the LLM to output an incorrect classification as compared to the ground truth.

A.3 Few-Shot Instruction

This is the exact instruction we gave the LLM for the Few-Shot Instruction experiments:

```
Core Purpose:
You are an AI assistant specialized in determining the IP
Service Operator for an IP address given a JSON file from
Censys containing information about the various ports and
services from that IP. The IP Service Operator is the
entity responsible for Internet-visible services on a
given IP of which they can directly control and configure.
In most cases, IPs are owned by the ISPs, CDNs, Hosting
Providers, etc. and the IP Service Operator would be the
CUSTOMER of that entity that owns the IP.
Specifically Distinguishing a IP Service Operator:
The following list below are NOT the IP Service Operator
unless proven otherwise:
  • Autonomous System (AS) associated with the IP
    address
  • Any Certificate Issuers Entities
  • Cloud providers or Hosting infrastructure providers
    or Infrastructure Leaser
  • Any third-party service providers or
    telecommunication organizations that do not have
    direct control over the Internet services running on
    an IP
Critical Thinking Requirements:
  • Explain your chain of thought
  • First Identify what is most likely not an operator
  • Then use the remaining information to identify if an
    operator exists
  • Really think about the decisions and conclusions you
    are making and self-assess whether your conclusions
    actually make sense
Important Notes:
  • IP Service Operators tend to be present through
    representations of domains in the metadata. In the
    case that a domain is identified that may
    potentially be the operator. Make sure to validate
    and confirm that the operator is valid.
  • Default to "Indiscernible" for the IP Service
    Operator when only ISP/hosting/CDN information is
    available with no indication of the actual customer
    (IP Service Operator), the data only shows technical
    details without organizational identifiers, and
    there's no clear evidence of who operates the IP
    services. Additionally, be cautious with networking
    equipment: Router/modem services usually indicate
    infrastructure or hardware (not the IP Service
    Operator).
  • A human should be able to understand the IP Service
    Operator output so they can identify the associated
    entity. If the IP Service Operator is not readable
```

or can't be easily verified by a human such as an unknown id or unknown domain, then default to "Indiscernible".
- If the IP Service Operator is an acronym that is not commonly known, leave it in that form without unravelling it.

Important constraints:
- ONLY use information from the attached JSON file and do not utilize previous context from another IP
- Rely on the JSON data and your pre-existing knowledge about internet services and do not attempt to hallucinate or come up with a reasoning without strongly supported data from the JSON or your knowledge. Make sure to verify and validate any classification

Sample Examples: (NOTE: I will provide you only a snippet of the JSON for a given example where it contains information about some of the Operators. These examples are not necessarily representative of all the IP addresses and DO NOT bias by only focusing on the metadata that is provided below to find/classify the operator).

Example 1:

Portion of a Filtered JSON:


```
{
  "ip": "207.167.125.51",
  "services": [
    {
      "_decoded": "banner_grab",
      "_encoding": {
        "banner_hex": "DISPLAY_HEX"
      },
      "banner": "\u0000\u0005\u0000\u0001\u0000\u0000",
      "extended_service_name": "TFTP",
      "labels": [
        "file-sharing"
      ],
      "port": 69,
      "service_name": "TFTP",
      "source_ip": "167.94.138.37",
      "transport_protocol": "UDP"
    },
    {
      "_decoded": "banner_grab",
      "_encoding": {
        "certificate": "DISPLAY_HEX"
      },
      "extended_service_name": "UNKNOWN",
      "jarm": {
        "_encoding": {
          "fingerprint": "DISPLAY_HEX",
          "cipher_and_version_fingerprint": "DISPLAY_HEX",
          "tls_extensions_sha256": "DISPLAY_HEX"
        }
      },
      "port": 3998,
      "service_name": "UNKNOWN",
      "software": [
        {
          "part": "o",
          "vendor": "redhat",
          "product": "enterprise_linux",
          "source": "OSI_TRANSPORT_LAYER"
        }
      ],
      "source_ip": "199.45.155.88",
      "tls": {
        "version_selected": "TLSv1_2",
        "certificates": {
          "_encoding": {
            "leaf_fp_sha_256": "DISPLAY_HEX",
            "chain_fps_sha_256": "DISPLAY_HEX"
          },
          "leaf_data": {
            "names": [
              "255.255.255.255"
            ],
            "pubkey_bit_size": 2048,
            "pubkey_algorithm": "RSA",
            "issuer": {
              "organization": [
                "Mitel"
              ],
              "province": [
                "ON"
              ]
            },
            "subject": {
              "common_name": [
                "SAC connection"
              ],
              "organization": [
                "Mitel"
              ],
              "province": [
                "ON"
              ]
            },
            "public_key": {
              "key_algorithm": "RSA",
              "rsa": {
                "exponent": "AAEAAQ==",
                "length": 256
              }
            },
            "signature": {
              "signature_algorithm": "SHA256-RSA"
            }
          },
          "chain": [
            {},
            {}
          ]
        },
        "_encoding": {
          "ja3s": "DISPLAY_HEX"
        },
        "ja4s": "t120100_009c_bc98f8e001b5",
        "versions": [
          {
            "tls_version": "TLSv1_2",
            "_encoding": {
              "ja3s": "DISPLAY_HEX"
            },
            "ja4s": "t120100_009c_bc98f8e001b5"
          },
          {
            "tls_version": "TLSv1_1",
            "_encoding": {
              "ja3s": "DISPLAY_HEX"
            },
            "ja4s": "t110100_002f_bc98f8e001b5"
          },
          {
            "tls_version": "TLSv1_0",
            "_encoding": {
              "ja3s": "DISPLAY_HEX"
            },
            "ja4s": "t100100_002f_bc98f8e001b5"
          }
        ]
      },
      "transport_fingerprint": {
        "id": 103,
        "os": "Redhat"
      },
      "transport_protocol": "TCP"
    },
    {
      "_decoded": "banner_grab",
      "_encoding": {
        "certificate": "DISPLAY_HEX"
```

```
},
"extended_service_name": "UNKNOWN",
"jarm": {
  "_encoding": {
    "fingerprint": "DISPLAY_HEX",
    "cipher_and_version_fingerprint": "DISPLAY_HEX",
    "tls_extensions_sha256": "DISPLAY_HEX"
  }
},
"port": 5061,
"service_name": "UNKNOWN",
"software": [
  {
    "part": "o",
    "vendor": "redhat",
    "product": "enterprise_linux",
    "source": "OSI_TRANSPORT_LAYER"
  }
],
"source_ip": "206.168.34.88",
"tls": {
  "version_selected": "TLSv1_2",
  "certificates": {
    "_encoding": {
      "leaf_fp_sha_256": "DISPLAY_HEX",
      "chain_fps_sha_256": "DISPLAY_HEX"
    },
    "leaf_data": {
      "names": [
        "207.167.125.51"
      ],
      "pubkey_bit_size": 2048,
      "pubkey_algorithm": "RSA",
      "issuer": {
        "organization": [
          "Mitel"
        ],
        "province": [
          "ON"
        ]
      },
      "subject": {
        "common_name": [
          "SIP connection",
          "207.167.125.51"
        ],
        "organization": [
          "Mitel"
        ],
        "province": [
          "ON"
        ]
      },
      "public_key": {
        "key_algorithm": "RSA",
        "rsa": {
          "exponent": "AAEAAQ==",
          "length": 256
        }
      },
      "signature": {
        "signature_algorithm": "SHA256-RSA"
      }
    },
    "chain": [
      {},
      {}
    ]
  },
  "_encoding": {
    "ja3s": "DISPLAY_HEX"
  },
  "ja4s": "t120100_009c_bc98f8e001b5",
  "versions": [
    {
      "tls_version": "TLSv1_2",
      "_encoding": {
        "ja3s": "DISPLAY_HEX"
      },
      "ja4s": "t120100_009c_bc98f8e001b5"
    },
    {
      "tls_version": "TLSv1_1",
      "_encoding": {
        "ja3s": "DISPLAY_HEX"
      },
      "ja4s": "t110100_002f_bc98f8e001b5"
    },
    {
      "tls_version": "TLSv1_0",
      "_encoding": {
        "ja3s": "DISPLAY_HEX"
      },
      "ja4s": "t100100_002f_bc98f8e001b5"
    }
  ]
},
"transport_fingerprint": {
  "id": 103,
  "os": "Redhat"
},
"transport_protocol": "TCP"
},
{
"_decoded": "banner_grab",
"_encoding": {
  "certificate": "DISPLAY_HEX"
},
"extended_service_name": "UNKNOWN",
"jarm": {
  "_encoding": {
    "fingerprint": "DISPLAY_HEX",
    "cipher_and_version_fingerprint": "DISPLAY_HEX",
    "tls_extensions_sha256": "DISPLAY_HEX"
  }
},
"port": 6801,
"service_name": "UNKNOWN",
"software": [
  {
    "part": "o",
    "vendor": "redhat",
    "product": "enterprise_linux",
    "source": "OSI_TRANSPORT_LAYER"
  }
],
"source_ip": "199.45.155.111",
"tls": {
  "version_selected": "TLSv1_2",
  "certificates": {
    "_encoding": {
      "leaf_fp_sha_256": "DISPLAY_HEX",
      "chain_fps_sha_256": "DISPLAY_HEX"
    },
    "leaf_data": {
      "names": [
        "255.255.255.255"
      ],
      "pubkey_bit_size": 2048,
      "pubkey_algorithm": "RSA",
      "issuer": {
        "organization": [
          "Mitel"
        ],
        "province": [
          "ON"
        ]
      },
      "subject": {
        "common_name": [
          "MINET connection"
        ],
        "organization": [
          "Mitel"
```

```
            ],
            "province": [
              "ON"
            ]
          },
          "public_key": {
            "key_algorithm": "RSA",
            "rsa": {
              "exponent": "AAEAAQ==",
              "length": 256
            }
          },
          "signature": {
            "signature_algorithm": "SHA256-RSA"
          }
        },
        "chain": [
          {},
          {}
        ]
      },
      "_encoding": {
        "ja3s": "DISPLAY_HEX"
      },
      "ja4s": "t120100_009c_bc98f8e001b5",
      "versions": [
        {
          "tls_version": "TLSv1_2",
          "_encoding": {
            "ja3s": "DISPLAY_HEX"
          },
          "ja4s": "t120100_009c_bc98f8e001b5"
        },
        {
          "tls_version": "TLSv1_1",
          "_encoding": {
            "ja3s": "DISPLAY_HEX"
          },
          "ja4s": "t110100_002f_bc98f8e001b5"
        },
        {
          "tls_version": "TLSv1_0",
          "_encoding": {
            "ja3s": "DISPLAY_HEX"
          },
          "ja4s": "t100100_002f_bc98f8e001b5"
        }
      ]
    },
    "transport_fingerprint": {
      "id": 103,
      "os": "Redhat"
    },
    "transport_protocol": "TCP"
  }
],
"location": {
  "country": "United States",
  "country_code": "US",
  "city": "Rocklin",
  "province": "California",
  "coordinates": {
    "latitude": 38.79073,
    "longitude": -121.23578
  }
},
"autonomous_system": {
  "asn": 400257,
  "bgp_prefix": "207.167.125.0/24",
  "country_code": "US"
},
"whois": {
  "network": {
    "cidrs": [
      "207.167.125.0/24"
    ],
    "allocation_type": "ALLOCATION"
  },
  "organization": {
    "handle": "CR-994",
    "city": "Rocklin",
    "country": "US",
    "abuse_contacts": [
      {
        "name": "Hostmaster"
      }
    ],
    "admin_contacts": [
      {
        "name": "Hostmaster"
      }
    ],
    "tech_contacts": [
      {
        "name": "Hostmaster"
      }
    ]
  }
},
"operating_system": {
  "part": "o",
  "vendor": "redhat",
  "product": "enterprise_linux",
  "source": "OSI_TRANSPORT_LAYER"
},
"dns": {
  "names": [
    "sip.rocklin.ca.us"
  ],
  "records": [
    "mbg.rocklin.ca.us",
    "sip.rocklin.ca.us"
  ],
  "reverse_dns": {
    "names": [
      "sip.rocklin.ca.us"
    ]
  }
},
"labels": [
  "file-sharing"
]
}
```


Operators for Example 1:
IP Service Service Operator: City of Rocklin, CA

Example 2:

Partial Filtered JSON Input:


```
{
 "code": 200,
 "status": "OK",
 "result": {
   "ip": "24.234.24.53",
   "services": [
     {
       "tls": {
         "certificates": {
           "leaf_data": {
             "issuer": {
               "common_name": [
                 "fortinet-subca2001"
               ],
               "country": [
                 "US"
               ],
               "email_address": [
                 "support@fortinet.com"
               ],
               "locality": [
                 "Sunnyvale"
               ],
```

```
              "organization": [
                "Fortinet"
              ],
              "organizational_unit": [
                "Certificate Authority"
              ],
              "province": [
                "California"
              ]
            },
            "issuer_dn": "C=US, ST=California, L=Sunnyvale,
            O=Fortinet, OU=Certificate
            Authority, CN=fortinet-subca2001,
            emailAddress=support@fortinet.com",
            "subject": {
              "common_name": [
                "FGT81FTK23011756"
              ],
              "country": [
                "US"
              ],
              "email_address": [
                "support@fortinet.com"
              ],
              "locality": [
                "Sunnyvale"
              ],
              "organization": [
                "Fortinet"
              ],
              "organizational_unit": [
                "FortiGate"
              ],
              "province": [
                "California"
              ]
            },
            "subject_dn": "C=US, ST=California,
            L=Sunnyvale, O=Fortinet, OU=FortiGate,
            CN=FGT81FTK23011756, emailAddress=support@fortinet.com",
          }
        }
      }
    }
  ]
 }
}

Operators for Example 2:
Service Operator: Indiscernible

Final Output: Output only the IP address, IP Service
Operator in the JSON format specified.
```


A.4 LLM Response Configuration

This is the response configuration that defines the structure of LLM's output

```
RESPONSE_TEXT_CONFIG = {
  "format": {
    "type": "json_schema",
    "name": "ip_operators",
    "schema": {
      "type": "object",
      "properties": {
        "final_answer": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "ip_address": {
                "type": "string",
                "description": "The IP address represented by this element"
              },
              "ip_service_operator": {
                "type": "string",
                "description": "Comma separated list of IP-service operators or \"Indiscernible Operator\""
              }
            },
            "required": ["ip_address", "ip_service_operator"],
            "additionalProperties": False
          }
        }
      },
      "required": ["final_answer"],
      "additionalProperties": False
    },
    "strict": True
  }
}
```

A.5 Pruning

In this section we provide the exact implementation details used in pruning.

A.5.1 WHOIS Pruning The exact RegEx OperatorSage uses for WHOIS filtering is:
`(net|tele|isp|internet|provider|communications|network|broadband|wireless|cable|telephone|te`

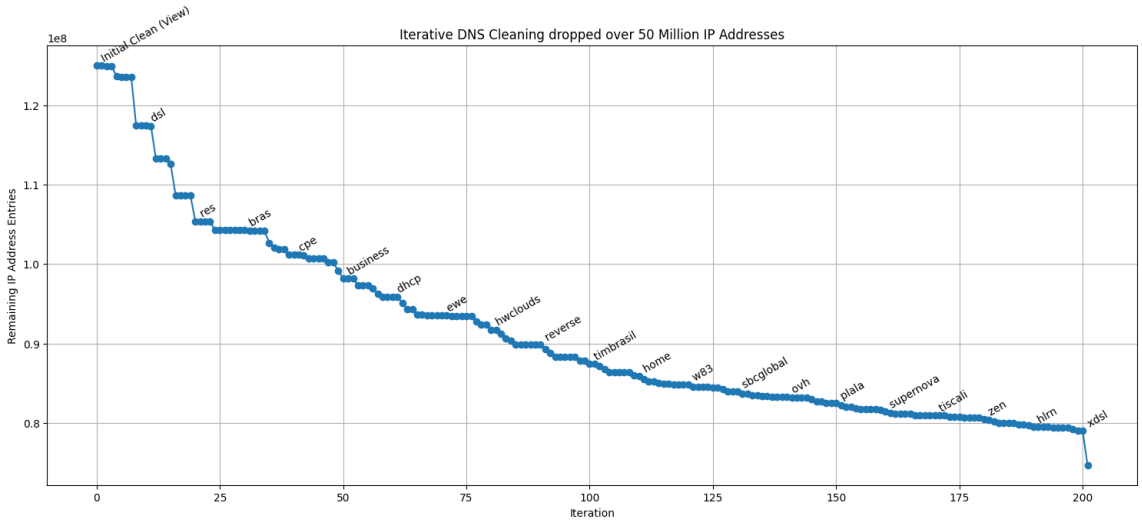


Figure 5: IPv4 Iterative DNS Processing: Our results of aggregating the most common keywords from our normalized DNS entries iteratively removed over 50 Million Rows, which could save billions of input tokens when prompting the LLM (innately reducing cost).

A.5.2 DNS Pruning We implement DNS pruning that emulates a human's reasoning, and systematically remove noise at scale.

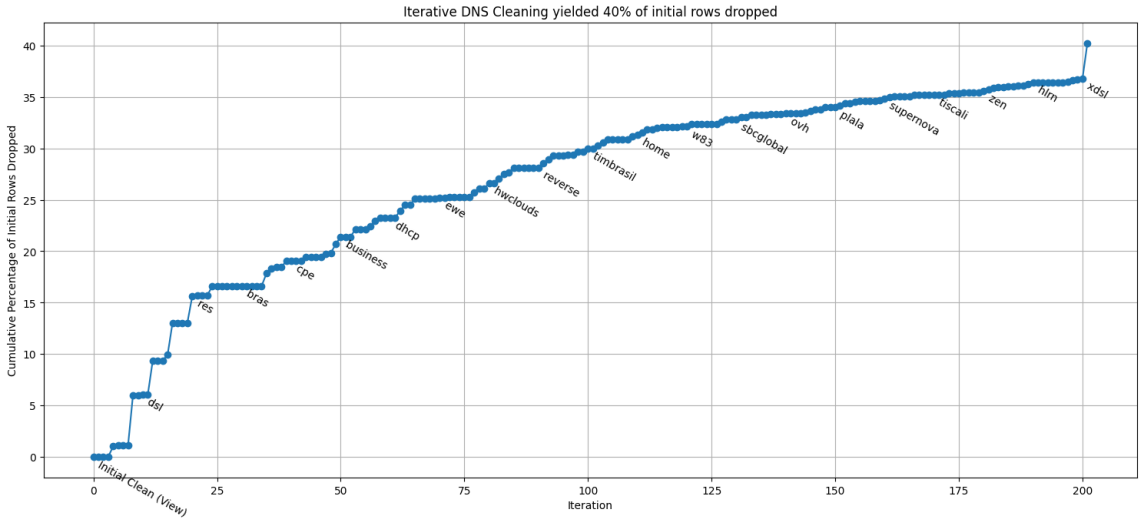


Figure 6: Cumulative Pruning during DNS: Our results of aggregating the most common keywords from our normalized DNS entries processed and dropped 40% of the initial non-null sample set, amounting to roughly 20% of all IPv4 Addresses

Stage 1: Preliminary Cleaning and Tokenization. We first drop any record that exactly matches an IPv4 pattern (e.g., matching `^d{1,3}(?:[-.]d{1,3}){3}$`) or is empty. Next, we normalize each name by collapsing repeated dots and hyphens and stripping leading/trailing delimiters. We then remove both the dotted and hyphenated forms of the host IP (and any other IP-like sequences) to prevent numeric fragments from polluting our tokens. We refine our filtering by splitting DNS names at common delimiters (e.g., '.' and '-') and removing the most frequent strings found between them. This process excludes IP sequences, compute/CDN keywords (e.g., AWS, Azure), and dynamic DNS terms (e.g., dynamic, static). By only checking between delimiters, we avoid mistakenly removing relevant domain parts in other contexts (e.g., "ghost" from "host").

`a23-215-205-65.deploy.static.`
`akamaitechnologies.com`

remove IP, ↓ normalize & split

[`deploy`, `static`, `akamaitechnologies`, `com`].

Stage 2: Iterative Common-Token Removal (Phase 1). We run up to N iterations of the following loop (for our study we incremented from 50 to 200 to find a suitable number of iterations).

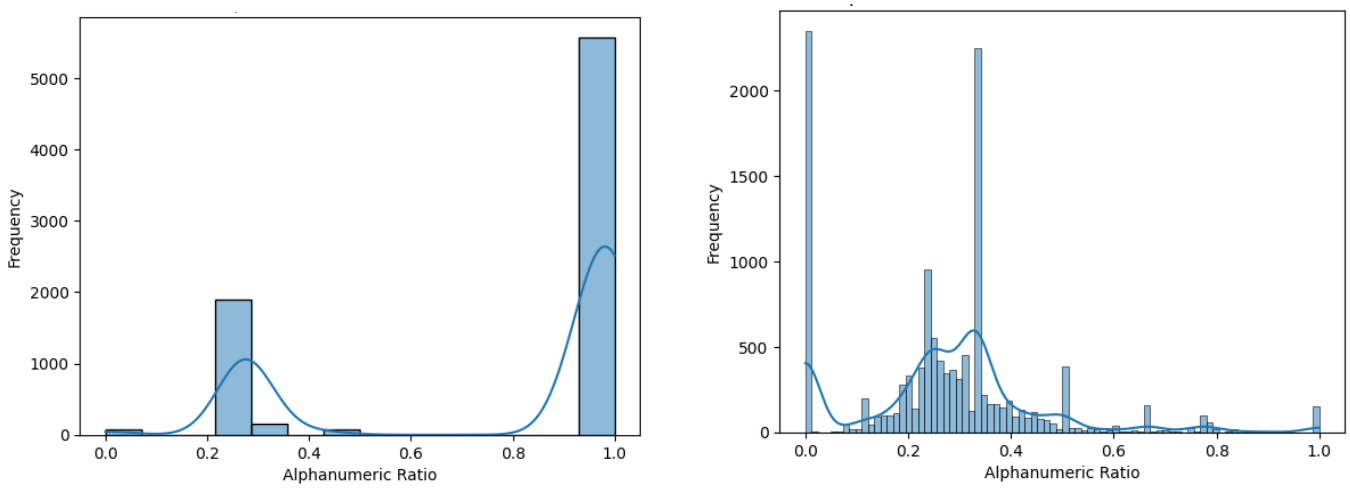
1) Identify the most frequent token across all records, excluding Top-Level Domains, numeric strings, and tokens shorter than three characters.
2) Remove exact matches of this token from every record's token list.
3) Reassemble each record and drop those that become empty, still match an IP pattern, or consist solely of TLDs and short fragments.

By removing only exact, data-driven common tokens (e.g., "static," "deploy"), we avoid hard-coding exclusion lists and adapt to the actual distribution of infrastructure markers.

Stage 3: Final Refinement with Alphanumeric Filtering (Phase 2). For the survivors of Phase 1, we re-tokenize and strip any tokens removed earlier, repeat IP-sequence stripping and normalization, then apply an alphanumeric filter (e.g., assessing patterns of hexadecimal characters and length) to eliminate hash-like names. This final pass ensures that remaining records, such as `suprasetter.desktop.utk.edu` (tokens `[suprasetter, desktop, utk]`), preserve semantically meaningful operator cues while generic entries like `186.202.65.34.bc.googleusercontent.com` are discarded.

If neither delimiter exists, we ultimately pass the full domain to the LLM.

Applying this pipeline to an entire IPv4 Snapshot (where there were 124, 970, 972 IP Addresses with a valid DNS service), we pruned 50, 257, 506(40.21%)(Figure 5), entries (~40 %(Figure 6)) that contained only generic or infrastructure-related tokens, retaining the remaining 74, 713, 466 DNS records for further LLM-based operator attribution.



(a) Hashes

(b) Human-Interpretable fields

Figure 7: Alphanumeric Ratio Compositions—Hashes exhibit significantly higher alphanumeric character ratios compared to human-discernible fields.

A.5.3 High Entropy Pruning Since human-readable fields are clustered between a 0 to 0.35 alphanumeric ratio, but hashes are clustered around 0.9, we use 0.9 as a threshold to distinguish meaningful, human-interpretable information from machine-generated artifacts (i.e., we filter out fields that show alphanumeric ratios higher than 0.9).