Trust But Verify: Evaluating the Accuracy of LLMs in Normalizing Threat Data Feeds

Author: Nick Peterson, nickp2121@gmail.com
Advisor: *Lenny Zeltser*

Accepted: June 14th, 2025

Abstract

This paper examines whether Large Language Models (LLMs) can be reliably applied to the normalization of Indicators of Compromise (IOCs) into Structured Threat Information Expression (STIX) format. Using benchmark datasets of 200 IOCs across three types (MD5 hashes, URLs, and IPv4 addresses), the performance of Google's Gemini 2.0 Flash and OpenAI's ChatGPT-4o will be evaluated. While both models achieved 100% validity in generating syntactically correct STIX outputs, their fidelity in accurately preserving IOC values varied significantly. Gemini outperformed ChatGPT overall, though both models struggled with hash values, exhibiting frequent omissions and erroneous pattern translations. The inconsistencies in these errors pose a major obstacle to the reliable use of LLMs in operational security and data engineering pipelines.

1. Introduction

Following their widespread introduction to the public in 2022 with the release of ChatGPT 3.5, Large Language Models (LLMs) have been successfully leveraged in various use cases (Cheng et al., 2025). In information security specifically, LLMs have been applied for tasks such as summarization of data related to incident response cases (Rosique et al., 2024), aggregation and synthesis of threat intelligence reporting (Recorded Future, 2024), and evaluation of large data sets to identify anomalous activity (GreyNoise, 2023).

These models, however, are not deterministic like a traditional computer program (Ouyang et al., 2023). While they can approximate determinism, uniform inputs will not necessarily create the same output (Song et al., 2024). Secondarily, LLMs have known issues with hallucinating data (inventing values that either do not exist or were not present in the input) and omitting data points (Asaduzzaman et al., 2025). Given these constraints, could an LLM be successfully utilized in a data engineering capacity, reformatting bulk sets of threat data from disparate sources for security applications?

1.1. Structured Threat Information Expression (STIX)

A common challenge in cybersecurity programs is ingesting threat data feeds, specifically Indicators of Compromise (IOCs), to defend an enterprise environment (Pawlinski & Kompanek, 2016). This is typically accomplished through a Threat Intelligence Platform (TIP), which acts as a repository for threat information, including IOCs (Faiella et al., 2019). These IOCs can come from multiple sources and contain various indicator types and contexts, and while typically available in JSON or CSV, the specific formatting of any given feed is very infrequently standardized (Ramsdale et al., 2020).

One solution to this issue has been the advent of the Structured Threat Information Expression Version 2.1 (STIX) data standard (Barnum, 2012). STIX is a language and specification for communicating both threats and observable information, comprising a set of schemas for object types, which are represented through JSON. IOCs

in STIX are represented as indicator objects consisting of a pattern (e.g., an IPv4 address) and the surrounding context, such as the last date seen, the date it is valid until, or a description of the activity associated with the pattern.

STIX usage, however, is far from ubiquitous (Mkuzangwe & Khan, 2020), and normalizing IOC feeds for use within a TIP built to process STIX indicator objects requires connectors or integrations, programs designed to convert data from each specific source into STIX (Sauerwein et al., 2017).

The ability of LLMs to take unstructured, variable input data and provide a structured output could offer an opportunity to alter this workflow. By taking input from multiple data sources and performing this conversion using an LLM, we could potentially replace multiple integrations or connectors with one that queries an LLM via its API, supplying the indicator feed data as part of its prompt. This use case could only be supported if the model performing the reformatting operation had an extremely high degree of accuracy. To this end, two popular LLM models will be evaluated to provide a snapshot of their current accuracy in translating content and properly formatting when normalizing structured feeds of IOCs into STIX.

2. Research Method

To measure accuracy rates in situations similar to real-world use cases, our experiment took multiple lists, each consisting of a different indicator type, and supplied them as part of the prompt in an API call to each LLM. This prompt also included system instructions, directing the model to convert items in the file into a list of STIX indicator objects with a given number of fields and required context. Each input list was run through this operation 20 times, with output results saved to a directory. Following the completion of all queries, all files in each directory were then evaluated for adherence to STIX formatting standards, completeness of records, and presence of any unexpected values.

2.1.Model Selection

Before exploring accuracy rates, the appropriate models for this task had to be identified. Several criteria are measured by benchmarking services such as LiveBench (an independent service that poses questions to AI models and grades their responses), including reasoning, mathematics, and coding. Given that the proposed task involved data reformatting, rather than advanced reasoning, the requirements for this testing differed from common benchmarks and focused on the following:

- Ability to handle structured inputs and outputs
- Ability to accept a large amount of instruction / large context window (in the form of an indicator schema and prompt, including feed data)
- Speed in conducting the operation
- Cost relative to available alternatives

Models were also expected to be production-ready and stable releases. While newer models with advanced reasoning were available, such as Gemini 2.5 or ChatGPT 4.5, those were eschewed in favor of more tested models without the "experimental" designation.

Owing to its comparatively large context window (the amount of information that can be sent and received in a given prompt) of 1 million tokens, its optimization for tasks involving structured data, and its speed of response, Google's Gemini 2.0 Flash model was selected for the initial round of experiments. While Google offers a non-'Flash' Gemini 2.0 model, given the use case of converting large streams of indicators for use in enterprise systems, the cheaper and shorter response time Flash version was selected to more accurately align with the requirements of a production system in an enterprise environment.

While lacking a similarly large context window, OpenAI's 4o model's ubiquity as an LLM (Guinness, 2025) and reliable JSON handling (Liu et al., 2024) made it a natural selection as a point of comparison. Both models supported JSON responses through features titled "Structured Outputs" (same title for both models). This allows users of the Google or OpenAI generative AI Python libraries to supply a JSON schema via the Pydantic Python library. This instructs the model to provide all output in that schema in a

pseudo-deterministic fashion, as opposed to the more error-prone method of providing an example output in the prompt.

2.2. Indicator Source Selection

To ensure the data normalization use case is applicable to a wide range of inputs, three threat data feeds were selected to represent different IOC types and levels of contextual detail. The IOC types selected were URLs, IPv4 Addresses, and MD5 hashes. These indicators were drawn from the following sources:

- AbuseIPDB Blacklist (Abuse IPDB, n.d.)
- UrlHaus Recent Additions (URLHaus, n.d.)
- MalwareBazaar TrickBot Samples (MalwareBazaar, n.d.)

Each source was queried via REST API, and JSON responses were provided. Responses were limited to 200 results to provide a realistic sample size large enough to draw inferences from and to limit the tokens input per request. The response from the MalwareBazaar query contained SHA-256 and SHA-1 hashes in addition to an MD5 hash. To simplify the LLM's task and avoid the unintentional generation of additional indicator objects, this response was processed to remove these fields. All responses were stored locally as JSON files.

2.3. Scripting The LLM API Call

Scripts were written in Python to make the API calls to each LLM to perform the data reformatting operations (see attached: gpt_stix.py and gemini_stix.py). These scripts performed the following actions:

1. Define a JSON schema for the LLM to output data in. To simplify this experiment, this schema represented only the required fields for an STIX indicator object and an optional description field to add context.
2. Batch objects from the input file. Despite the sample size of 200 indicators, both models required batching to avoid incomplete or aborted responses. Testing revealed that the optimal batch size varied according to both the

indicator type and model used, resulting in batches that ranged from 25 to 50 objects in length.

3. Perform the data conversion request. Provide system instructions, the input file as a prompt, and the schema as an output configuration option.
4. Append additional fields required to create a valid STIX indicator object. To reduce the task complexity and minimize opportunities for hallucinated values, fields like timestamps and unique identifiers (UUIDs) were intentionally left blank through instructions in both the prompt and the schema itself. These fields were generated and added in post-processing.
5. Create a STIX bundle. Wrap indicator lists returned from all batched requests into a bundle with its own UUID.
6. Save to the output file in a given directory.
7. Repeat this operation 20 times.

2.3.1. Prompt Engineering

Each request to the LLM API consisted of a prompt split into two parts: the batched JSON contents of the input file and a series of system instructions. The system instructions (with minor variation due to differing Structured Output requirements in each model) were:

```
You are a system that converts indicators of compromis
e (IOCs) from raw threat intelligence into STIX 2.1 In
dicator objects.

Each input item may represent a URL, domain, IP addres
s (IPv4 or IPv6), file hash (MD5, SHA1, SHA256), or ot
her observable.

For each item:

· Use the appropriate STIX pattern (e.g., `[url:value =
  '...']`, `[domain-
  name:value = '...']`, `[file:hashes.SHA256 = '...']`,
  `[ipv4-addr:value = '...']`)
```

```
· Include a basic name and a short description if availa
  ble

· Leave `id`, `created`, `modified`, and `valid_from` fi
  elds blank

Output a **JSON array** of STIX Indicator objects — do
  not wrap in a bundle, dictionary, or markdown
```

These instructions followed the prompting techniques outlined in Gemini's Prompt Design Strategies document (Google, 2025), as well as those outlined in OpenAI's Prompt Engineering guide (OpenAI, 2025)— namely defining the model's role as a 'system', clearly explaining the use case, providing examples of valid responses, and providing explicit requirements for desired output.

2.4. Measuring Accuracy of Content and Formatting

The output files generated by these reformatting operations were then evaluated by two Python scripts. The first script (see attached validate_format.py) would scan each file in the directory and confirm the validity of its formatting. This task was facilitated by the Stix2Validator Python library. The results of this evaluation, consisting of a True/False statement on the valid formatting of each file in the target directory, were then saved locally. Once these results were saved, an accuracy rate was determined by dividing the number of valid STIX responses by the total number of sample submissions.

To determine a high level of statistical confidence in our accuracy measurement, the sample size must be determined (Woolson, 1986). The minimum number of sample submissions (n) is determined by the desired confidence level (Z), the margin of error (E), and a guess at the accuracy rate (p) using Cochran's Sample Size Formula (Cochran, 1954):

$$n = (Z^2 * p * (1-p)) \ / \ E^2$$

Early testing, solely leveraging prompting to reformat input and lacking the structured output component, yielded very few errors in translation of indicator values.

Given these results, the expected accuracy rate was conservatively set to 95%. To achieve a 95% confidence level in these accuracy rates with a 5% margin of error, Cochran's formula yielded a minimum sample size of 73.

Testing showed that the maximum batch size for submitting indicators from these sources is 40. Submitting more than this occasionally caused truncation, leading to incomplete batches and affecting sample post-processing. As each batch of 40 represented a unique request, performing this operation 20 times for each set of 200 indicators resulted in 100 unique requests. As discussed in the findings, the batch size was subsequently further reduced to 25 on hash values due to the sample file taking up additional tokens per indicator. This reduction put the total number of requests for hash value operations at 160, double the minimum sample size. In both cases, the total number of requests exceeded the minimum sample size and provided a small buffer to account for failed requests or other errors.

The second script gauged the accuracy of value translation and was specific to each indicator type (see Appendix). Each of the indicator-specific scripts performed the same set of actions:

- Read the input file (e.g., the original JSON response of indicators) and create a list of indicator values.
- Open each file in a directory and search its contents for each of the values listed in the input file.
- Identify missed values, unexpected values (not in the input file), and repeated values.
- Provide an accuracy percentage for each file
- Save results to a local file

Accuracy rates for the translation of value were then determined by averaging the percentages of accurately translated indicator values among the 20 sample submissions for each indicator type.

3. Findings and Discussion

The findings illustrated that both models excel at formatting their output properly, given the appropriate instruction, and can be successful at indicator translation tasks for certain indicator types. Gemini 2.0 Flash outperformed ChatGPT 4o at data normalization tasks, achieving greater accuracy in correctly translating indicator values while omitting fewer indicators and encountering no mistranslation errors.

While not explicitly measured as part of this experiment, Gemini was also demonstrably faster than ChatGPT at producing output as well. This result was interesting but could be due to several reasons, such as the purposeful allocation (or constraint) of resources to a given model to favor performance. Regardless, requests to reformat groups of 200 MD5 hashes took roughly 1.5x longer for ChatGPT to perform compared to Gemini, a time difference that quickly added up over the course of several dozen requests.

3.1. Validation of Formatting

Initial testing, using only a prompt without leveraging structured outputs, had shown that the primary cause for failures in formatting accuracy was the occasional omission of an entire required field, such as “pattern_type”. These were batch-level errors, representing individual requests to the LLM’s API being met with responses that ignored certain fields for all indicators in that batch.

While the Structured Outputs feature was chosen to ameliorate these issues and attempt to simplify this operation, the number of possible output fields was stripped down to those essential to allow a receiving system to correctly interpret a feed. Each model was supplied the following pared-down STIX indicator schema via Pydantic:

```
class Indicator(BaseModel):
    type: Literal["indicator"] = Field(..., description="
            Must be the literal 'indicator'")
    spec_version: Literal["2.0", "2.1"] = Field(..., desc
            ription="STIX specification version")
```

```
    id: str = Field(..., description="STIX ID, leave blan
k in output")
    created: str = Field(..., description="ISO 8601 times
            tamp, leave blank")
    modified: str = Field(..., description="ISO 8601 time
            stamp, leave blank")
    pattern: str = Field(..., description="Detection patt
ern (e.g., '[ipv4-addr:value = '1.1.1.1']')")
    pattern_type: str = Field(..., description="Pattern t
ype (e.g., 'stix', 'yara')")
    valid_from: str = Field(..., description="Start time

  from which the indicator is valid (ISO 8601)")
    description: Optional[str] = Field(None, description=
            "Contextual description of the indicator")
```

Both Gemini 2.0 Flash and ChatGPT 4o leveraging Structured Outputs were successful in producing valid STIX content from all input files. The models adhered perfectly to the supplied schema, including all required fields and correctly using the STIX formatting prefix for recording all “indicator patterns” (the values assigned to indicator objects, e.g., ipv4-addr:value = 1.1.1.1) despite only explicitly listing the IPv4 pattern prefix in the prompt.

The optional “description” field contents varied with both models, as only vague instructions on what to provide were given. In some cases, indicators were linked to a type of malware, e.g., “Mozi Malware Download URL," while in others, they provided significantly more context, e.g., “URL associated with Mozi malware download targeting 32-bit ELF MIPS. Reporter: geenensp.” These differences were observed in batches, with each API request corresponding to a decision on how much or how little context to provide in the description field. While not explicitly evaluated for accuracy as part of this experiment due to the abstract nature of the instructions for this field, sampled description fields appeared to provide relevant and accurate data.

More relevant to this experiment was that, in all instances, the supplied values adhered to STIX formatting requirements. Batch size remained a constraint, as samples at or near a given model's token limit would frequently be truncated, resulting in an error in formatting due to improperly terminated parentheses or brackets. Testing revealed 40 to be a manageable batch size for IPv4 indicators and URL indicators on each model. Due to the increased amount of context available in the MalwareBazaar hash sample, and therefore increased token usage in the prompt, this batch was reduced to 25 indicators for hash samples on both models. Using these batch sizes, each model performed the task with no errors in output formatting (See Figure 1).

Model	IPv4	URL	Hash
Gemini 2.0 Flash	100.0	100.0	100.0
ChatGPT 4o	100.0	100.0	100.0

Figure 1: Percentage of Valid STIX Responses by Indicator Type

3.2. Accuracy of Translated Values

While both models performed well in translating values of IPv4 indicators, Gemini 2.0 Flash vastly outperformed ChatGPT 4o in faithfully reproducing URL and Hash indicator values. Figure 2 shows the overall accuracy rate for each indicator type; however, this percentage is only part of the picture. Within the individual responses, there existed a large amount of variation in accuracy, which we will explore by model.

Model	IPv4	URL	Hash
Gemini 2.0 Flash	100.0	100.0	99.975
ChatGPT 4o	100.0	98.7	99.075

Figure 2: Percentage of Accurate Indicator Values by Indicator Type

3.2.1. Gemini 2.0 Flash

As illustrated in Figure 3, Gemini faithfully translated all IP and URL indicator values into STIX objects; however, the results with hashes were mixed. As the amount of context, and therefore input size in tokens, was larger in the hash input file than that of the other indicator types, the batch size of 40 resulted in one batch being improperly processed and resulting in a translation of only 160/200 indicators. As this error was in the post-processing phase, not in the translation of values phase, this response was discarded rather than being weighed toward accuracy. Of the remaining 19 responses, 8 had one omitted value, resulting in an aggregate accuracy rate of 99.789%.

As the large batch size had resulted in a processing error and was a potential cause of omitted values, the batch was reduced to 25 objects for a second round of testing. This managed to eliminate any issues from improper processing, though omitted values remained. Of the twenty response samples, ten omitted at least one value. This resulted in an accuracy rate of 99.7%.

One hypothesis for the omissions is that certain values shared a commonality that caused them to be omitted; however, no such commonality was identified. A second hypothesis is that these correspond to the edge or last values passed in a batched request, which are more likely to be "forgotten" (citation); however, those would presumably be uniform throughout each request and represent the same values, which these did not.

To try to limit these occurrences, a third round of testing was performed in which the following instruction was added to the system message of the prompt, under the instructions "for each item":

> "`Generate exactly one STIX indicator. Process all items without omission, skipping none.`"

This alteration to the prompt significantly reduced omissions and resulted in a 99.975% accuracy rate, representing one response out of 20 omitting a single indicator.

Response Number	IPv4	URL	Hash	Hash (Batch Size 25)	Hash (Prompt Altered)
stix_output_001	100.0	100.0	100.0	100.0	100.0
stix_output_002	100.0	100.0	99.5	99.5	100.0
stix_output_003	100.0	100.0	99.5	99.5	100.0
stix_output_004	100.0	100.0	99.5	99.5	100.0
stix_output_005	100.0	100.0	100.0	99.5	100.0
stix_output_006	100.0	100.0	100.0	100.0	100.0
stix_output_007	100.0	100.0	99.5	100.0	100.0
stix_output_008	100.0	100.0	100.0	99.5	100.0
stix_output_009	100.0	100.0	80.0	99.5	100.0
stix_output_010	100.0	100.0	100.0	100.0	100.0
stix_output_011	100.0	100.0	100.0	100.0	100.0
stix_output_012	100.0	100.0	99.5	100.0	100.0
stix_output_013	100.0	100.0	99.5	100.0	99.5
stix_output_014	100.0	100.0	100.0	100.0	100.0
stix_output_015	100.0	100.0	100.0	99.5	100.0
stix_output_016	100.0	100.0	100.0	99.5	100.0
stix_output_017	100.0	100.0	99.5	100.0	100.0
stix_output_018	100.0	100.0	99.5	99.5	100.0
stix_output_019	100.0	100.0	100.0	100.0	100.0
stix_output_020	100.0	100.0	100.0	98.5	100.0
Avg.	100.0	100.0	99.7894736842105	99.7	100.0

Response Number	IPv4	URL	Hash	Hash (Batch Size 25)	Hash (Prompt Altered)

Figure 4: Gemini Accuracy Rates Per Output File

3.2.2. ChatGPT 4o

Accuracy rates within the responses provided by ChatGPT also varied by indicator type. Like Gemini, ChatGPT was able to successfully translate all values from the IP input list. URLs fared worse, with accuracy rates of as low as 87.5%. Hashes similarly fared poorly, with an accuracy rate bottoming out at 93.5% in one response sample (See: Figure 5).

Response Number	IPv4	URL	Hash
stix_output_001	100.0	100.0	93.5
stix_output_002	100.0	100.0	99.5
stix_output_003	100.0	100.0	99.5
stix_output_004	100.0	100.0	100.0
stix_output_005	100.0	100.0	100.0
stix_output_006	100.0	100.0	100.0
stix_output_007	100.0	100.0	98.0
stix_output_008	100.0	100.0	100.0
stix_output_009	100.0	100.0	100.0
stix_output_010	100.0	87.5	98.0
stix_output_011	100.0	100.0	100.0
stix_output_012	100.0	100.0	98.0
stix_output_013	100.0	100.0	100.0
stix_output_014	100.0	100.0	97.0

Response Number	IPv4	URL	Hash
stix_output_015	100.0	99.0	100.0
stix_output_016	100.0	100.0	100.0
stix_output_017	100.0	100.0	99.0
stix_output_018	100.0	100.0	100.0
stix_output_019	100.0	87.5	99.0
stix_output_020	100.0	100.0	100.0
Avg.	100.0	98.7	99.075

Figure 5: ChatGPT Accuracy Rates Per Output File

While the hash accuracy rate was solely the result of omitted values, the translation errors encountered in the URL sample were of another sort. In all 20 response files, ChatGPT successfully returned 200 indicator objects. Upon inspection of the resulting outputs, it was revealed that the input URLs were mistakenly parsed as a different indicator type in three files. In two files, these errors corresponded to an entire batch parsing URLs such as “http://42.239.95.74:40774/Mozi.m” (defanged) as IP addresses, extracting the IP value and storing it as “[ipv4-addr:value = ’42.239.95.74’]". This resulted in 25 missing values out of 200 in each file, representing the two files with an 87.5% accuracy rate.

In a third file, two URLs were mistakenly translated as domain objects, taking values such as “hxxps://buqoc[.]icu/“ (defanged) and outputting a domain object for buqoc[.]icu. This error type resulted in 2 missing values out of 200.

3.3. Discussion

These results led to a few initial observations. Clearly, while these models are known to be non-deterministic, both ChatGPT and Gemini can be coaxed to closely approximate determinism. This can be interpreted both positively and negatively. On the positive side, this level of accuracy is clearly useful in many use cases, given the

previously mentioned successful applications of LLMs in cybersecurity. It may even be better than some alternative methods of accomplishing a given task (e.g., assigning a reformatting operation to a person to do manually would almost certainly result in a higher error rate).

There exists, however, the prospect of this 'illusion' of determinism leading to the integration of LLMs in environments where this rate of accuracy is insufficient. This could result in silent errors, both in omissions and hallucinated or mistranslated data, where an LLM is employed in an enterprise environment to conduct some banal reformatting task (e.g., curating either an allow or a block list) with unknown downstream effects.

With the observed error rates, could scripts like the ones leveraged for this experiment be put into use on production systems to manage the intake of disparate indicator feeds? Potentially, Gemini's accuracy rates for all indicator types and the ability of both models to produce uniformly structured outputs in STIX are particularly encouraging. Any current implementation would likely have to be very limited in scope, perhaps focusing on one indicator type (IPv4, for example) and leveraging sources with limited context as inputs. It is also important to distinguish that while the statistical confidence of these rates is 95% with a 5% margin of error, in the instances in which an indicator type received 100% accuracy (such as with IPv4 indicators), this is likely a function of an insufficiently sensitive or expansive experiment, rather than a truly perfect accuracy rate. Even with a tightly scoped use case, there are still hurdles to overcome.

Due to the uncertainty of the results produced, any system to perform this operation would have to be implemented in tandem with robust error correction. At a minimum, any system would have to parse the input file for values, validate the results, and make the request again if there are errors in the outcome. This would at least ensure the translation of all desired values.

Secondly, as observed in the types of errors encountered by ChatGPT translating URL values, there exists the distinct possibility that additional objects would be generated by the LLM inadvertently and passed to the receiving system. Threat feeds can often provide a large amount of context per IOC, and that can include additional IOCs, as

addressed in our pre-processing of JSON responses from MalwareBazaar to remove additional hashes.

Errors of this type could potentially pose several problems, the first of which relates to resource utilization. Inadvertent indicator creation could artificially increase the number of tokens used by the model in its output. If an input contains several potentially valid indicators of compromise, the output could grow drastically by creating an object with all associated contexts for each one. Imagine one malware record producing separate indicator objects for MD5, SHA1, SHA256, etc. hash values. Should this number exceed a certain threshold, it would likely result in both truncated results and an increase in the rate of omissions of the intended indicators.

Another problem presented by this type of error is the inadvertent ingestion of benign observables as indicators of compromise. Threat feeds can contain benign domains, for example, if the information is derived from a cybersecurity blog or news article. As evidenced in the URL sample data, URLs frequently contain bare IP addresses. These can pose issues when this information is operationalized in an automated way. If an automated workflow is designed to pick up indicators from a high-fidelity feed of known-bad infrastructure and add them to a blocklist that is sent out to other security appliances (e.g., firewalls), a URL being misinterpreted as an IP could have a serious operational impact on an organization. IP addresses, particularly large cloud providers, can and frequently do host millions of domains or resources. If, instead of one resource being blocked, an organization ends up inadvertently blocking a major cloud provider, the consequences could be serious.

Another question generated by this research is whether or not a system like this would be practical. At present, the answer appears to be no. In this experiment, we leveraged fairly conservative volumes of 200 indicators. Processing speeds for these indicators ranged from 127 seconds with Gemini 2.0 Flash to 189 seconds with ChatGPT 4o. These are relatively arbitrary values that could likely be scaled up or down at the whim of the service provider; however, they represent an order of magnitude more time than required by a traditional Python program designed to handle a specific input. Even factoring in large step changes in speed in the future, bringing these numbers down to tens of seconds would still be unconscionably slow compared to the alternative.

Indicator feeds also frequently contain vastly more information than the samples used in this experiment, and may need to be pulled daily. Performing this operation on tens of thousands of indicator objects in batches would constitute hundreds of requests, each taking minutes to complete. When taken in concert with the necessary error correction and secondary requests, it would not be a pragmatic use of time or resources to do these large-scale operations with an LLM.

Despite that, as response speeds increase along with accuracy rates, it is possible that a use case like this could be supported in the future. Are there additional similar use cases that could be supported by this functionality? Absolutely. For the ad hoc processing of indicators into different formats, LLMs at this accuracy rate are extremely useful and are undoubtedly used by analysts worldwide for small-scale data normalization tasks. Given their difference from traditional computing, the proper balance of accuracy, speed, and cost must be considered when weighing the use of LLMs for a given use case.

4. Recommendations and Implications

Both the findings and the experimentation leading up to them yielded several takeaways, including some nuance in approaching prompting and teasing out the appropriate structured response from both models. Secondly, the findings and errors identified within the output pose some interesting opportunities to explore additional accuracy questions and tease out why certain error types exist.

4.1. Recommendations for Prompting and Structured Output

Despite using structured outputs, each model requires a bit of prompting to get it to perform the task you have in mind. While the OpenAI and Google prompt engineering guides are helpful, they are also light on detail. Trial and error, including the identification of common error types for your specific data set and querying the model itself on how best to mitigate those errors, is an invaluable tool.

Structured Output with Gemini was intuitive and perhaps better suited to the exact task assigned to it. Once the schema was supplied, lists of indicator objects in that format were immediately produced, as intended. ChatGPT required a little more prompting.

Initial attempts following OpenAI's documentation returned only one item in a given schema (e.g., the first indicator encountered). To get it to support the use case we had in mind, a wrapper schema was added to encapsulate the indicator objects. This wrapper was then explicitly called out in the response_format field in the call to the OpenAI API:

```
Class IndicatorListWrapper (BaseModel):
    items: list[Indicator]
```

The following line was also added to the prompt to ensure a properly formatted response:

```
Wrap the array of indicators inside a JSON object with key items.
```

These tweaks succeeded in getting ChatGPT to return the data in the correct format.

4.2. Implications for Future Research

There is no shortage of advancements in LLMs, with new models appearing multiple times a year (Sindhu et al., 2024). As LLM models develop, it will be interesting to run this or a similar experiment against newer models to see if there is an increase or decrease in errors of omission or misidentification of data types. It would also be worthwhile to research the speed of the operation. While that may be a function of design choices or resourcing by each service provider, the speed at which these operations are performed still presents a large practical hurdle to implementing an LLM for the use case posited in this paper.

The results with IPv4 indicators were uniformly impressive; however, given the aberrations seen in other data types, it is safe to assume that their true accuracy rate is not 100% but that a larger sample size would be required to determine it. Performing a similar experiment on a much larger dataset could be an interesting additional line of research, either by performing this operation an additional X number of times on the

same sets of 200 indicators or by expanding the dataset to encompass the tens of thousands of IP addresses frequently seen in IP threat feeds.

URL indicator types were similarly impressive, but their occasional errors in the ChatGPT results are interesting. It is easy to understand why a URL consisting of a bare IP might be misinterpreted as an IP address object; however, it is less clear why two short URLs were misinterpreted as domains and what factors might mitigate those occurrences. While it's likely possible to tune the prompt to reduce instances of misidentification in the URL input file, given the use case in this specific experiment (handling diverse sets of data with a variety of contents), continual attempts to tune a request to cater to one data source would defeat the stated purpose of using an LLM. It is easy to imagine an ever-increasing, byzantine system prompt designed to handle inputs from a growing number of data sources, taking up more room in the LLM's context window.

Finally, the biggest question generated by this experiment is: Why is there a clear downgrade in accuracy for hash values? It would be worthwhile to do additional testing on a variety of samples involving file hashes, attempting to identify why they are more frequently omitted from responses than other indicator types and what workarounds may exist to eliminate omissions. As LLMs function in large part by predicting the next token (Downes et al., 2024), could the omission be due to the amount of entropy in hashes (Mitzenmacher & Vadhan, 2008), making them more "difficult" for the models to process? Is it simply a larger token count in the request causing omission errors, and if so, why did these errors continue to present when the batch size was reduced?

The sample files used in this experiment represent a fairly curated version of real-world data, particularly regarding the pre-processing of the data from MalwareBazaar. It is highly likely that feeds encountered in real-world usage would present significantly more complex challenges for LLMs, including multiple valid indicators and much more accompanying context. Identifying the underlying cause for the errors encountered or identifying a model not similarly troubled by these error types would be worthwhile research.

5. Conclusion

This experiment demonstrated that both models had a remarkable ability to produce valid STIX outputs given an input schema, particularly for systems defined by their non-deterministic output. While Gemini 2.0 Flash managed to outperform ChatGPT 4o in faithfully translating values across data formats and did so significantly faster, both encountered data omission errors that cast doubt on the ability of these systems to be used in a capacity that requires a high degree of accuracy. At present, the specific use case of bulk indicator reformatting might not be practical to implement for most organizations due to a number of factors, but these results remain encouraging for both information security use cases, and any data normalization tasks that leverage LLMs and the types of errors encountered represent areas of potential future research.

References

(2025). Abuse IPDB. https://abuseipdb.com

Asaduzzaman, M., Giorgi, I., & Masala, G. (2025, April 30). Filtering hallucinations and omissions in large language models through a cognitive architecture. IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/10977857

Barnum, S. (2012). Standardizing cyber threat intelligence information with thestructured threat information expression (stix). *Mitre Corporation*, *11*, 1–22

Cheng, J., Ghate, K., Hua, W., Wang, W., Shen, H., & Fang, F. (2025, March 24). *REALM: A dataset of real-world LLM use cases*. arXiv.org. https://doi.org/10.48550/arXiv.2503.18792

Downes, S. M., Forber, P., & Grzankowski, A. (2024). LLMs are not just next-token predictors. *arXiv preprint arXiv:2408.04666*.

Faiella, M., Granadillo, G. G., Medeiros, I., Azevedo, R., & Zarzosa, S. G. (2019, July). Enriching Threat Intelligence Platforms Capabilities. In *ICETE (2)* (pp. 37-48).

Google. (2025). *Prompt design strategies | Gemini API | Google AI for developers*. Google AI for Developers. https://ai.google.dev/gemini-api/docs/prompting-strategies

GreyNoise. (2023, October 2). *Introducing sift: Automated threat hunting*. https://www.greynoise.io/blog/introducing-sift-automated-threat-hunting

Guinness, H. (2025, May). *The best large language models (LLMs) in 2025*. Zapier: Automate AI Workflows, Agents, and Apps. https://zapier.com/blog/best-llm/#

Liu, Y., Li, D., Wang, K., Xiong, Z., Shi, F., Wang, J., ... & Hang, B. (2024). Are LLMs good at structured outputs? A benchmark for evaluating structured output capabilities in LLMs. *Information Processing & Management*, *61*(5),103809.

(2025). MalwareBazaar. https://bazaar.abuse.ch

Mitzenmacher, M., & Vadhan, S. P. (2008, January). Why simple hash functions work: exploiting the entropy in a data stream. In *SODA* (Vol. 8, pp. 746-755).

Mkuzangwe, N. N., & Khan, Z. C. (2020). Cyber-threat information-sharing standards: A review of evaluation literature. *The African Journal of Information and Communication*, *25*, 1-12.

OpenAI. (2025). *Text generation and prompting*. https://platform.openai.com/docs/guides/text?api-mode=chat

Ouyang, S., Zhang, J., Harman, M., & Wang, M. (2023, August). An empirical study of the non-determinism of ChatGPT in code generation. ADS. https://ui.adsabs.harvard.edu/abs/2023arXiv230802828O/abstract

Pawlinski, P., & Kompanek, A. (2016, February 24). *Evaluating threat intelligence feeds*. FIRST - Improving Security Together. https://www.first.org/resources/papers/munich2016/kompanek-pawlinski-evaluating-threat-ntelligence-feeds.pdf

Ramsdale, A., Shiaeles, S., & Kolokotronis, N. (2020). A comparative analysis of cyber-threat intelligence sources, formats and languages. *Electronics*, *9*(5), 824.

Recorded Future. (2024, February 13). *Recorded future launches enterprise AI for intelligence*. https://www.recordedfuture.com/press-releases/recorded-future-launches-enterprise-ai-for-intelligence

Rosique, L., Keller, J., Kramer, D., Bowen, A., & Cho, A. (2024, April 26). *Accelerating incident response using generative AI*. Google Online Security Blog. https://security.googleblog.com/2024/04/accelerating-incident-response-using.html

Sauerwein, C., Sillaber, C., Mussmann, A., & Breu, R. (2017). Threat intelligence sharing platforms: An exploratory study of software vendors and research perspectives.

Sindhu, B., Prathamesh, R. P., Sameera, M. B., & KumaraSwamy, S. (2024, May). The evolution of large language model: Models, applications and challenges. In *2024 International Conference on Current Trends in Advanced Computing (ICCTAC)* (pp. 1-8). IEEE.

Song, Y., Wang, G., Li, S., & Lin, B. (2024, July 15). The good, the bad, and the greedy: Evaluation of LLMs should not ignore non-determinism. arXiv.org. https://arxiv.org/abs/2407.10457

(2025). URLHaus. https://urlhaus-api.abuse.ch/

Woolson, R. F., Bean, J. A., & Rojas, P. B. (1986). Sample size for case-control studies using Cochran's statistic. *Biometrics*, 927-932.

Nick Peterson, nickpt90017@gmail.com

Appendix

All scripts used in this research project are available at:

https://github.com/nick-pete/trust_but_verify_proj/