Normalizing Audit Logs Using Large Language Models

Anonymous Author(s)

ABSTRACT

We present a novel approach for normalizing audit logs from various *Independent Software Vendor* (ISV)s by generating *Velocity Template Language* (VTL) templates for mapping input events from ISVs to *Open Cybersecurity Schema Framework* (OCSF) format using zero shot learning with *Large Language Model* (LLM)s. In this approach, we use hierarchical classification to classify events from an ISV into appropriate OCSF event categories, event classes and event activities. Then we use the JSON schema for the generated OCSF event classes to generate VTL templates, which map the fields in the input events to the fields in the OCSF format. We use the ISV event name and description for the event classification task and the event json schema and a collection of sample event logs for the VTL template generation task. We evaluate the results of the two tasks using human generated event mappings and VTL templates for various ISVs as ground truth respectively. We also use a different LLM for evaluation of the outputs of the two tasks, by generating confidence scores and qualitative assessment for both tasks using an evaluation prompt. If the confidence score is lower than a preset threshold, the generated qualitative feedback is used to improve the LLM output for the VTL template generation task. This work helps improve the error prone and time consuming audit log normalization process by doubling the event classification accuracy obtained through human annotators, and reducing the VTL template generation process for new ISVs by from 2 days to half a day.

KEYWORDS

large language models, zero shot learning, open cybersecurity schema framework, velocity template language, classification, constrained LLM generation, LLM output evaluation, prompt engineering, log management, log normalization

ACM Reference Format:

Anonymous Author(s). 2024. Normalizing Audit Logs Using Large Language Models. In *Proceedings of Knowledge Infused Learning Workshop (KIL).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

ISVs are organizations that develop and market software products and solutions for commercial or business use. In today's digital landscape, organizations often rely on multiple software systems and applications from different ISVs to support their operations. For example, independent SaaS applications like Slack, Zoom, and Dropbox are incredibly popular with users. Interoperability [7] between these applications ensures that these systems can seamlessly communicate, exchange data, and work together effectively, eliminating data silos and enabling efficient workflows. For enabling interoperability between various ISVs, including a single interface to search, manage, and secure data, a normalization mechanism is needed for structuring raw data from multiple ISVs into a standardized format to facilitate analysis and querying.

ISVs use log management tools [2, 10] or security information and event management solutions to collect, store, and analyze audit logs. Audit logs are chronological records that document the sequence of activities or events that occur within an organization's computer systems, applications, networks, user accounts, or devices. They are essential for monitoring, tracking, and maintaining the security and integrity of an organization's IT infrastructure, by providing insights into user activities, system changes, security events, and potential incidents or breaches. By normalizing audit logs, organizations can ensure data consistency, integrity, and efficiency, while also enabling better data analysis, reporting, and security measures.

In this work, we present a novel approach for normalizing audit logs from various ISVs. Motivated by the success of LLMs in related NLP tasks using zero shot learning, like classification [1, 5, 11] and its reasoning abilities over complex data [3, 4, 8, 12, 14], this paper explores the potential of LLMs for four tasks: i) event classification, ii) template generation for normalization, iii) evaluation of LLM outputs for tasks i) and ii), and iv) using LLM evaluation output to improve the results of tasks i) and ii). In the event classification task, the objective is to classify an input event from an ISV into the appropriate OCSF event class. OCSF [1] is an open-source project that aims to provide a common language and structure to represent and share cybersecurity data and information. In the template generation task, the objective is to convert an input event log from an ISV into the OCSF format for the OCSF event class that the input event was classified into in task i).

We propose a prompt based architecture for instructing LLMs to classify an input event from an ISV into the appropriate OCSF category, class and activity, evaluate the classification output, generate a template to convert the input event into OCSF format for the generated OCSF event class and evaluate the template output. We also propose a mechanism to use the LLM evaluation output as input to the template generation step to improve the LLM output. We use Claude V3 [2] LLMs via AWS Bedrock [3]: Haiku for generation and Sonnet for evaluation, and compare the event classification and template generation performance using human generated event mappings and templates for certain ISVs which are treated as ground truth. Since, event classification into OCSF events is an error prone task, we also ask the human annotators who produced

[1]https://schema.ocsf.io/

[2]https://www.anthropic.com/news/claude-3-family

[3]https://aws.amazon.com/bedrock/

KIL, August 25, 2024, Barcelona, Spain

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

the ground truth event mappings to manually examine the LLM generated event mappings for certain ISVs to identify scenarios where the human annotator is incorrect but the LLM output seems to be a better match.

This paper is structured as follows: Section 2 introduces the event mapping and template datasets. Section 3 describes the experimental setup. Section 4 discusses the LLM prompts, Section 5 discusses the VTL templates generated by the LLM, and Section 6 focuses on future work and improvements that can be made to the current experimental setup to achieve better results.

2 DATASET

This section introduces the datasets used for evaluation.

2.1 Event Classification

10 ISVs were chosen with different number and types of audit log events, for which human annotators classified the ISV events into OCSF event classes. The details of the number of unique events and ground truth OCSF event categories, classes and activities for each ISV can be found in Table 1. We also use Crowdstrike [4] events to determine how often a human annotator thought that the LLM generated output was a better event classification result than the human generated ground truth.

2.2 Template Generation

We generate templates using three combinations of inputs to the LLM: i) only raw audit logs, ii) only input event schema, iii) raw audit logs and input event schema. For i) we use raw audit logs from Figma[5], for ii) input event schema for Crowdstrike, and for iii) we use we use input event schema and raw audit logs from Asana[6]. We use the human generated templates as ground truth for these ISVs for manual comparison against the LLM generated output.

3 EXPERIMENTAL SETUP

This section describes each step in the system architecture described in Figure 1. For each LLM generation step, we set temperature to 0 and top_k value to 1 to ensure maximum reproducibility[9].

3.1 Event Classification

There are 312 OCSF activities that an input event needs to be classified into. For directly classifying an event into one of these activity types, the LLM prompt needs to have names (and descriptions) for all these activities. This results in a very long prompt. Context stuffing, i.e. giving the LLM more context or knowledge as a part of the prompt, has an upper limit on the amount of text that can be provided as context. The reasons for this are manifold. LLM output quality decreases and the risk of hallucination increases as the context size increases [6]. Cost also increases linearly with larger contexts, since processing larger contexts requires more computation, and LLM providers charge per token. There is also a hard upper limit on the number of tokens that can be sent to the model.

To address this issue, the event classification task is broken down into 3 steps [13]. First, we identify the OCSF category for an input event. There are 6 OCSF categories that organize event classes and each of them is aligned with a specific domain or area of focus. Second, we identify the event class within the selected category and lastly, we identify the event activity within the selected event class that best describes the input event. The list of the event categories and the event classes within each category is given in Table 2. The event class chosen in the second step gives the OCSF JSON schema for normalizing the input event. This hierarchical classification approach breaks down the otherwise complex vanilla classification task of classifying an input event directly into 312 output classes to a simpler task having fewer (maximum 18) output classes at any step. This reduces the risk of hallucination and incorrect classification. It also allows us to pass additional information like output class descriptions in the LLM prompt to achieve better classification accuracy, without increasing the total number of input tokens required to get the event classification output.

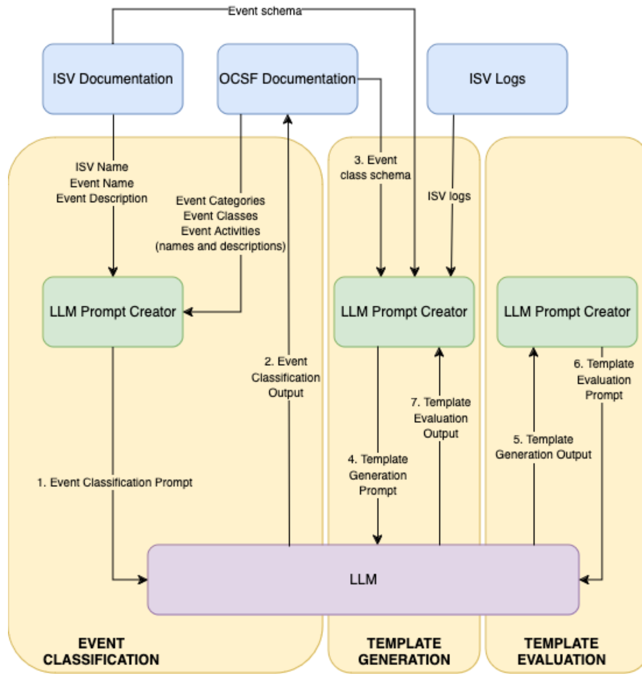

Figure 1: Audit Log Normalization Architecture

Along with the event classification output, we also instruct the LLM to output a verbose reasoning describing how it chose a specific output class for each step of the hierarchical classification process. This verbose reasoning helps a human annotator in a production environment to verify the LLM output and replace it with a human corrected event class if needed.

3.2 Event Classification Evaluation

We evaluate the output for event classification in two ways. For the 10 ISVs for which we have ground truth, we calculate the micro F1 score (ranging from 0.0 to 1.0) for the event category, event class and event activity. Table 3 contains F1 scores for event category, event class and event activity, as well as the total number of input tokens required to get all classification outputs and the

[4]https://www.crowdstrike.com/en-us/
[5]https://www.figma.com/
[6]https://asana.com/

Table 1: Event Classification Dataset

ISV Name	No. of Events	No. of OCSF Categories	No. of OCSF Classes	No. of OCSF Activities
Asana	157	2	6	37
Okta	875	4	11	52
Zoom	324	2	7	41
Cisco	540	3	7	42
Zendesk	31	2	5	16
Salesforce	63	3	7	16
Gitlab	301	3	7	37
Azure	872	3	10	48
1Password	132	2	7	36
Lastpass	114	2	6	37

Table 2: OCSF Event Categories and Classes

OCSF category	No. of OCSF events	List of OCSF Events
System Activity	7	File System Activity, Kernel Extension Activity, Kernel Activity, Memory Activity, Module Activity, Scheduled Job Activity, Process Activity
Findings	5	Security Finding, Vulnerability Finding, Compliance Finding, Detection Finding, Incident Finding
Identity & Access Management	6	Account Change, Authentication, Authorize Session, Entity Management, User Access Management, Group Management
Network Activity	13	Network Activity, HTTP Activity, DNS Activity, DHCP Activity, RDP Activity, SMB Activity, SSH Activity, FTP Activity, Email Activity, Network File Activity, Email File Activity, Email URL Activity, NTP Activity
Discovery	5	Device Inventory Info, Device Config State, User Inventory Info, Operating System Patch State, Device Config State Change
Application Activity	7	Web Resources Activity, Application Lifecycle, API Activity, Web Resource Access Activity, Datastore Activity, File Hosting Activity, Scan Activity

execution time in seconds for the same. It shows the improvement in classification accuracy by using hierarchical classification over vanilla classification. Hierarchical classification gives more than 3X boost in classification accuracy for OCSF event category, class and activity.

To determine if the verbose reasoning generated by the LLM during event classification (which describes how it chose an output event class), is helpful to a human annotator to verify human assigned labels, we use the input events from Crowdstrike. 1186 input events were classified into 31 OCSF event classes by the LLM and ground truth for the same was generated by human annotators without looking at the LLM generated output to prevent bias. The two outputs were then compared by manually examining the verbose reasoning generated by the LLM along with the output classes. Only 82 event classification outputs for Crowdstrike's 1186 events were revised by human annotators after manual inspection, indicating that the LLM was correct 93% of the time.

3.3 Template Generation

We generate VTL templates to convert input events from an ISV into the OCSF format for the OCSF event class for that event. VTL [7] is a Java-based template engine used for web application development. VTL templates are text files that contain a mix of static content (like HTML) and dynamic content placeholders (like variables or expressions) that are processed by the VTL engine to generate the final output. It is often used to map one data format to another format because it provides a flexible and powerful way to transform data structures and generate dynamic content.

We establish 3 types of workflows to generate VTL templates for input events from an ISV: i) using a sample of raw ISV audit logs, ii) using input schema for ISV input events, and iii) using a combination of both i) and ii). We use Figma, Crowdstrike and Asana as target ISVs for i), ii) and iii) respectively. We retrieve the input schema from ISV documentation and use dummy accounts in the ISVs to get raw audit logs. Audit logs have a specific field for the input event name which can be used along with input event description retrieved from ISV documentation, to generate event classification output for a log. This event class is then used to get

[7] https://velocity.apache.org/engine/1.7/user-guide.html

Table 3: Event Classification Results

ISV Name	Classification Type	Event Category F1	Event Class F1	Event Activity F1	No. of Input Tokens	Execution Time (s)
Asana	Vanilla	0.26	0.16	0.12	2799.05	0.75
Okta	Vanilla	0.26	0.17	0.10	2819.50	0.68
Zoom	Vanilla	0.30	0.10	0.06	2785.03	0.70
Cisco	Vanilla	0.10	0.06	0.04	2794.08	0.76
Zendesk	Vanilla	0.25	0.25	0.22	2788.1	0.72
Salesforce	Vanilla	0.25	0.17	0.12	2816.52	0.95
Gitlab	Vanilla	0.24	0.18	0.12	2800.20	0.73
Azure	Vanilla	0.24	0.02	0.01	2798.54	0.76
1Password	Vanilla	0.44	0.36	0.17	2791.84	0.74
Lastpass	Vanilla	0.42	0.34	0.26	2787.625	0.75
Asana	Hierarchical	0.87	0.71	0.40	2831.69	1.59
Okta	Hierarchical	0.76	0.44	0.30	2857.63	1.47
Zoom	Hierarchical	0.75	0.50	0.37	2768.59	1.41
Cisco	Hierarchical	0.85	0.69	0.44	2820.15	1.59
Zendesk	Hierarchical	0.96	0.96	0.74	2817.22	1.64
Salesforce	Hierarchical	0.96	0.63	0.26	2890.25	1.90
Gitlab	Hierarchical	0.87	0.69	0.50	2833.60	1.52
Azure	Hierarchical	0.86	0.62	0.53	2841.51	1.57
1Password	Hierarchical	0.75	0.62	0.39	2784.47	1.87
Lastpass	Hierarchical	0.78	0.56	0.36	2783.96	1.54

the OCSF format corresponding to that class that the input event needs to be converted into. We generate one VTL template for each event class per ISV using either one of the 3 workflows described above. In case of i), the chosen sample of audit logs has no more than one log per input event that has been classified into that OCSF event class.

3.4 Template Evaluation

We use an LLM evaluator prompt for evaluation of the generated VTL templates. The LLM used for template evaluation is Claude V3 Sonnet. The prompt instructs the LLM to output a confidence score between 0 to 1 (where 0 is the lowest score indicating a poor quality template and 1 is the highest score indicating a high quality template). The prompt also contains instructions describing what an ideal template should look like to help the LLM assign correct confidence scores. In addition to this score, we also instruct the LLM to output a qualitative assessment of the template, describing the problems it encountered with the template (if any), to justify the confidence score. This qualitative assessment helps in improving the LLM output as described in the following section, and also helps human annotators to interpret the confidence score for existing templates and using the assessment to manually improve the generated or human created templates if needed.

We also use human generated templates for certain ISVs like Asana to manual evaluation of the LLM generated template.

3.5 Qualitative LLM Feedback

We use separate prompts - which take in the usual inputs for template generation for the 3 different workflows mentioned in Section 3.3, along with the qualitative assessment generated by the LLM during template evaluation mentioned in Section 3.4, as well as the VTL template generated in Section 3.3 - and use the the assessment to improve the template until either the confidence score generated during template evaluation is greater than a preset threshold or the maximum number of retries is reached. We use a preset threshold of 0.9 and set the maximum number of retries to 3.

4 PROMPT ENGINEERING

We make several observations about the Claude V3 class of models while editing the prompts for improving the quality of LLM output based on comparison with ground truth data, manual human assessment and LLM evaluation.

Claude V3 models seem to understand *Extensible Markup Language* (XML) tags well and dividing the complex prompts into various sections using XML tags improves the LLM output. We arrange the sections in the prompt in the following order: i) input data descriptions and input data, ii) output classes or format description, and iii) guidelines and constraints for output generation. We use Pydantic's [8] format instructions for specifying the JSON output format for the LLM for all generation tasks and add an additional instruction at the end of each prompt directing the LLM to only generate the output in the specified JSON format without any XML tags. We observe that adding detailed descriptions of input and output classes for the event classification task improves the classification accuracy, so we add descriptions from ISV and OCSF documentations to the event classification prompt. We also observe that using descriptions of input fields of the ISV's input event for the VTL template generation task enhances the quality of the LLM generated VTL template.

For the event classification task, we instruct the LLM to first identify the main resource type affected by the input event using

[8] https://docs.pydantic.dev/latest/

the input event name and description, then identify the action performed in the input event that alters the identified resource type, and then identify the output class whose name or description indicates or mentions the identified resource type and the action. Such a step by step approach helps in mimicing the workflow of a human annotator and also breaks down the complex task into simpler subtasks for the LLM.

5 RESULTS

Examples of VTL templates generated for Figma, Asana and Crowdstrike can be found in Listings 2, 4 and 6 respectively, along with their LLM generated confidence scores and qualitative assessments in Listings 3, 5 and 7 respectively. The output fields mapped to null values by the LLM in the VTL templates have been removed for readability purposes. The LLM does a good job of mapping fields in the input event to those in the output OCSF format, even where if-else conditions are involved. It also follows the instructions to map fields only if they have the same type. It assigns null values to fields marked as required in the output schema whose mappings it can't determine using the input schema and/or logs. It also includes a list of input fields in the un-mapped section of the output for which no corresponding output fields can be determined. However, the templates suffer from certain problems like output fields being mapped to a constant value inferred from the supplied context, confusion between similar fields like source and destination endpoints, and same input fields being mapped to different output fields.

```
#foreach($inputEvent in $input_events)
{
  "raw_data": $inputEvent,
  "actor": {
    "user": {
      "email_addr": "$inputEvent.actor.email",
      "name": "$inputEvent.actor.name",
      "type": "$inputEvent.actor.type",
      "type_id": #if($inputEvent.actor.type == "user")1#else99#end,
      "uid": "$inputEvent.actor.id"
    }
  },
  "dst_endpoint": {
    "type": #if($inputEvent.entity.editor_type == "figma")browser#elseif($inputEvent.entity.editor_type == "figjam")desktop#else"unknown"#end,
    },
    "type_id": #if($inputEvent.entity.editor_type == "figma")6#elseif($inputEvent.entity.editor_type == "figjam")2#else0#end,
  },
  "file": {
    "hashes": [
      {
        "algorithm_id": 3,
        "value": "$inputEvent.entity.key"
      }
    ],
    "name": "$inputEvent.entity.name",
    "type": "$inputEvent.entity.editor_type",
    "type_id": #if($inputEvent.entity.editor_type == "figma")1#elseif($inputEvent.entity.editor_type == "figjam")2#else0#end,
    "uid": "$inputEvent.entity.key"
  },
  "src_endpoint": {
    "type": #if($inputEvent.context.client_name)browser#elseif($inputEvent.context.ip_address)desktop#else"unknown"#end,
    "ip": "$inputEvent.context.ip_address",
    "type_id": #if($inputEvent.context.client_name)8#elseif($inputEvent.context.ip_address)2#else0#end,
  },
  "type_uid": "$inputEvent.id",
  "time": "$inputEvent.timestamp",
  "timezone_offset": 0,
  "cloud": {
    "provider": "Figma",
  },
  "api": {
    "operation": "$inputEvent.action.type",
    "request": {
      "data": "$inputEvent.action.details",
    },
    "response": {
      "message": "$inputEvent.action.details"
    }
  },
  "message": "$inputEvent.action.type",
  "severity": #if($inputEvent.action.type == "fig_file_view")Informational#elseif($inputEvent.action.type == "fig_file_move" || $inputEvent.action.type == "fig_file_rename" || $inputEvent.action.type == "fig_file_create" || $inputEvent.action.type == "fig_file_save_as" || $inputEvent.action.type == "fig_file_export" || $inputEvent.action.type == "fig_file_unset_password" || $inputEvent.action.type == "fig_file_link_access_change" || $inputEvent.action.type == "fig_file_duplicate")Low#else"Other"#end,
  "severity_id": #if($inputEvent.action.type == "fig_file_view")1#elseif($inputEvent.action.type == "fig_file_move" || $inputEvent.action.type == "fig_file_rename" || $inputEvent.action.type == "fig_file_create" || $inputEvent.action.type == "fig_file_save_as" || $inputEvent.action.type == "fig_file_export" || $inputEvent.action.type == "fig_file_unset_password" || $inputEvent.action.type == "fig_file_link_access_change" || $inputEvent.action.type == "fig_file_duplicate")2#else99#end,
  "status": #if($inputEvent.action.type == "fig_file_view" || $inputEvent.action.type == "fig_file_move" || $inputEvent.action.type == "fig_file_rename" || $inputEvent.action.type == "fig_file_create" || $inputEvent.action.type == "fig_file_save_as" || $inputEvent.action.type == "fig_file_export" || $inputEvent.action.type == "fig_file_unset_password" || $inputEvent.action.type == "fig_file_link_access_change" || $inputEvent.action.type == "fig_file_duplicate")Success#else"Other"#end,
  "status_id": #if($inputEvent.action.type == "fig_file_view" || $inputEvent.action.type == "fig_file_move" || $inputEvent.action.type == "fig_file_rename" || $inputEvent.action.type == "fig_file_create" || $inputEvent.action.type == "fig_file_save_as" || $inputEvent.action.type == "fig_file_export" || $inputEvent.action.type == "fig_file_unset_password" || $inputEvent.action.type == "fig_file_link_access_change" || $inputEvent.action.type == "fig_file_duplicate")1#else99#end,
  "unmapped": {
    "entity_link_access": "$inputEvent.entity.link_access",
```

```
    "entity_proto_link_access": "$inputEvent.entity.proto_link_access",
    "context_team_id": "$inputEvent.context.team_id",
    "context_org_id": "$inputEvent.context.org_id",
    "context_is_figma_support_team_action": "$inputEvent.context.is_figma_support_team_action",
    "context_client_name": "$inputEvent.context.client_name"
  }
}
#end
```

Listing 1: LLM generated VTL template for OCSF event class "File Hosting Activity" for Figma using raw audit logs

```
Score : 0.8
Assessment : The VTL template maps most of the required fields correctly, but there are a few issues:

1. The dst_endpoint.location fields are not mapped, which should ideally be populated based on the IP address if possible.
2. The src_endpoint.type mapping could be improved. It currently maps to 'browser' if context.client_name is present, 'desktop' if context.ip_address is present, and 'unknown' otherwise. A more accurate mapping would be to use the context.client_name value directly if available, or map to 'desktop' if context.ip_address is present and context.client_name is not available.
3. The severity and severity_id mappings could be more granular based on the specific action.type values.
4. The status and status_id mappings are reasonable, but could potentially be more specific based on the action.type values.
5. The unmapped object contains fields that are not mapped to the output schema, which is acceptable, but it would be better to map them if possible.

Overall, the template covers most of the required fields and makes reasonable assumptions for mapping, but there is room for improvement in terms of accuracy and completeness.
```

Listing 2: LLM generated confidence score and qualitative assessment for Figma's VTL template for OCSF event class "File Hosting Activity"

```
#foreach($inputEvent in $input_events)
{
  "raw_data": $inputEvent,
  "actor": {
    "user": {
      "email_addr": "$inputEvent.actor.email",
      "name": "$inputEvent.actor.name",
      "type": "$inputEvent.actor.actor_type",
      "type_id": #if($inputEvent.actor.actor_type == "user")1#else99#end,
      "uid": "$inputEvent.actor.gid"
    }
  },
  "auth_protocol": #if($inputEvent.details.method.contains("PASSWORD"))
    "PASSWORD"
  #else
    "Other"
  #end,
  "auth_protocol_id": #if($inputEvent.details.method.contains("PASSWORD"))
    7
  #else
    99
  #end,
  "dst_endpoint": {
    "type": "$inputEvent.context.context_type",
    "hostname": "$inputEvent.context.user_agent",
    "ip": "$inputEvent.context.client_ip_address",
    "type_id": #if($inputEvent.context.context_type == "web")8#else99#end,
  },
  "http_request": {
    "http_headers": [],
    "uid": "$inputEvent.gid",
    "user_agent": "$inputEvent.context.user_agent",
  },
  "is_mfa": false,
  "is_remote": true,
  "logon_type_id": #if($inputEvent.event_type == "user_login_succeeded")
    10
  #elseif($inputEvent.event_type == "user_login_failed")
    99
  #elseif($inputEvent.event_type == "user_logged_out")
    13
  #else
    0
  #end,
  "service": {
    "name": "Asana",
  },
  "user": {
    "email_addr": "$inputEvent.resource.email",
    "name": "$inputEvent.resource.name",
    "type": "$inputEvent.resource.resource_type",
    "type_id": #if($inputEvent.resource.resource_type == "user")1#else99#end,
    "uid": "$inputEvent.resource.gid"
  },
  "device": {
    "type": "$inputEvent.context.context_type",
    "type_id": #if($inputEvent.context.context_type == "web")8#else99#end,
    "hostname": "$inputEvent.context.user_agent",
    "ip": "$inputEvent.context.client_ip_address"
  },
  "type_uid": #if($inputEvent.event_type == "user_login_succeeded")
    100
  #elseif($inputEvent.event_type == "user_login_failed")
    101
  #elseif($inputEvent.event_type == "user_logged_out")
    102
  #else
    0
  #end,
  "time": "$inputEvent.created_at",
  "message": #if($inputEvent.event_type == "user_login_succeeded")
    "User $inputEvent.resource.name logged in successfully"
  #elseif($inputEvent.event_type == "user_login_failed")
    "User $inputEvent.actor.name failed to log in"
  #elseif($inputEvent.event_type == "user_logged_out")
    "User $inputEvent.actor.name logged out"
  #else
    null
  #end,
```

```
  "severity": #if($inputEvent.event_type == "user_login_succeeded")
    "Informational"
  #elseif($inputEvent.event_type == "user_login_failed")
    "Medium"
  #elseif($inputEvent.event_type == "user_logged_out")
    "Informational"
  #else
    "Unknown"
  #end,
  "severity_id": #if($inputEvent.event_type == "user_login_succeeded")
    1
  #elseif($inputEvent.event_type == "user_login_failed")
    3
  #elseif($inputEvent.event_type == "user_logged_out")
    1
  #else
    0
  #end,
  "status": #if($inputEvent.event_type == "user_login_succeeded")
    "Success"
  #elseif($inputEvent.event_type == "user_login_failed")
    "Failure"
  #elseif($inputEvent.event_type == "user_logged_out")
    "Success"
  #else
    "Unknown"
  #end,
  "status_id": #if($inputEvent.event_type == "user_login_succeeded")
    1
  #elseif($inputEvent.event_type == "user_login_failed")
    2
  #elseif($inputEvent.event_type == "user_logged_out")
    1
  #else
    0
  #end
}
#end
```

Listing 3: LLM generated VTL template for OCSF event class "Authentication" for Asana using raw audit logs and input event schema"

```
Score : 0.8
Assessment : The VTL template maps most of the input fields to the corresponding output fields correctly. However, there are a few issues:

1. The `auth_protocol` and `auth_protocol_id` fields are not mapped accurately. The template assumes that if the `details.method` contains 'PASSWORD', it maps to the PAP authentication protocol, which may not always be correct.

2. The `dst_endpoint.type_id` field is not mapped correctly for all context types. It only handles the 'web' context type and maps everything else to 99 (Other).

3. The `logon_type_id` field is not mapped accurately for all event types. It only handles 'user_login_succeeded', 'user_login_failed', and 'user_logged_out' event types.

4. The `type_uid` field is not calculated correctly according to the formula `class_uid * 100 + activity_id`. The template assumes hardcoded values based on the event type.

5. The `timezone_offset` field is not mapped.

6. The `cloud` object fields are not mapped.

7. The `api` object fields are not mapped.

8. The `severity` and `severity_id` fields are not mapped accurately for all event types. The template only handles a few specific event types.

9. The `status` and `status_id` fields are not mapped accurately for all event types. The template only handles a few specific event types.

10. Some fields like `dst_endpoint.port`, `dst_endpoint.svc_name`, `dst_endpoint.container` fields, `dst_endpoint.location` fields, `http_request.http_headers`, `http_request.url` fields, `device.location` fields, `device.region`, `device.container` fields, and others are not mapped or set to null, which may not be accurate.

Overall, the template covers a good portion of the mapping, but there is room for improvement in handling edge cases, accurate mapping of certain fields, and ensuring all required fields are populated correctly.
```

Listing 4: LLM generated confidence score and qualitative assessment for Asana's VTL template for OCSF event class "Authentication"

```
#foreach($inputEvent in $inputEvents)
{
  "device": {
    "location": {
      "city": $inputEvent.K8SResourceNamespace,
      "continent": "unknown",
      "coordinates": [#if($inputEvent.K8SAdmissionReviewTimestamp)$inputEvent.K8SAdmissionReviewTimestamp, $inputEvent.K8SAdmissionReviewTimestamp#else[0.0, 0.0]#end],
      "country": "unknown"
    },
    "name": $inputEvent.K8SResourceName,
    "region": $inputEvent.K8SClusterName,
    "type": #if($inputEvent.K8SResourceKind == "Pod")
      "container"
    #elseif($inputEvent.K8SResourceKind == "Deployment")
      "server"
    #elseif($inputEvent.K8SResourceKind == "ReplicaSet")
      "server"
    #elseif($inputEvent.K8SResourceKind == "DaemonSet")
      "server"
    #elseif($inputEvent.K8SResourceKind == "StatefulSet")
      "server"
    #else
      "unknown"
    #end,
    "type_id": #if($inputEvent.K8SResourceKind == "Pod")
      6
    #elseif($inputEvent.K8SResourceKind == "Deployment")
      1
    #elseif($inputEvent.K8SResourceKind == "ReplicaSet")
```

```
        1
    #elseif($inputEvent.K8SResourceKind == "DaemonSet")
        1
    #elseif($inputEvent.K8SResourceKind == "StatefulSet")
        1
    #else
        0
    #end,
    "uid": $inputEvent.K8SResourceUid,
    "container": {
      "name": $inputEvent.K8SResourceName,
      "image": {
        "name": $inputEvent.K8SImage,
        "uid": $inputEvent.K8SImageId
      },
      "uid": $inputEvent.K8SContainerId,
      "size": #if($inputEvent.Size)$inputEvent.Size#else"unknown"#end,
      "hash": {
        "algorithm_id": 3,
        "value": $inputEvent.SHA256HashData
      }
    },
    "namespace_pid": #if($inputEvent.K8SPodUid) $inputEvent.K8SPodUid#else null#end,
    "hostname": $inputEvent.K8SResourceName,
    "ip": $inputEvent.K8SPodIP4
  },
  "actor": {
    "user": {
      "name": $inputEvent.K8SServiceAccountName,
      "type": "Service Account",
      "type_id": 3,
      "uid": $inputEvent.K8SServiceAccountName
    }
  },
  "app": {
    "name": $inputEvent.K8SImage,
    "uid": $inputEvent.K8SImageId,
    "vendor_name": #if($inputEvent.AppVendor)$inputEvent.AppVendor#else"unknown"#end,
    "version": #if($inputEvent.AppVersion)$inputEvent.AppVersion#else"unknown"#end
  },
  "type_uid": $inputEvent.K8SAdmissionReviewAction,
  "time": $inputEvent.K8SAdmissionReviewTimestamp,
  "timezone_offset": 0,
  "cloud": {
    "provider": #if($inputEvent.CloudPlatform == 1)
      "AWS"
    #elseif($inputEvent.CloudPlatform == 2)
      "Azure"
    #elseif($inputEvent.CloudPlatform == 3)
      "GCP"
    #else
      "Kubernetes"
    #end,
    "region": $inputEvent.K8SClusterName
  },
  "api": {
    "operation": $inputEvent.K8SAdmissionReviewOperation,
    "request": {
      "data": $inputEvent.K8SResourceObject,
      "uid": $inputEvent.K8SAdmissionReviewUid
    },
    "response": {
      "code": #if($inputEvent.K8SAdmissionReviewAction == "0x1")
        200
      #elseif($inputEvent.K8SAdmissionReviewAction == "0x2")
        400
      #else
        null
      #end,
      "error": #if($inputEvent.K8SAdmissionReviewAction == "0x2")
        "Failure"
      #else
        null
      #end,
      "error_message": $inputEvent.K8SAdmissionReviewMsg,
      "message": $inputEvent.K8SAdmissionReviewMsg
    }
  },
  "message": $inputEvent.K8SAdmissionReviewMsg,
  "severity": #if($inputEvent.K8SDetectionSeverity == "Low")
    "Low"
  #elseif($inputEvent.K8SDetectionSeverity == "Medium")
    "Medium"
  #elseif($inputEvent.K8SDetectionSeverity == "High")
    "High"
  #elseif($inputEvent.K8SDetectionSeverity == "Critical")
    "Critical"
  #elseif($inputEvent.K8SDetectionSeverity == "Informational")
    "Informational"
  #else
    "Other"
  #end,
  "severity_id": #if($inputEvent.K8SDetectionSeverity == "Low")
    2
  #elseif($inputEvent.K8SDetectionSeverity == "Medium")
    3
  #elseif($inputEvent.K8SDetectionSeverity == "High")
    4
  #elseif($inputEvent.K8SDetectionSeverity == "Critical")
    5
  #elseif($inputEvent.K8SDetectionSeverity == "Informational")
    1
  #else
    99
  #end,
  "status": #if($inputEvent.K8SAdmissionReviewAction == "0x1")
    "Success"
  #elseif($inputEvent.K8SAdmissionReviewAction == "0x2")
    "Failure"
  #elseif($inputEvent.K8SAdmissionReviewImageAssessmentPolicyID)
    "Success"
  #else
    "Other"
  #end,
  "status_id": #if($inputEvent.K8SAdmissionReviewAction == "0x1")
    1
  #elseif($inputEvent.K8SAdmissionReviewAction == "0x2")
    2
  #elseif($inputEvent.K8SAdmissionReviewImageAssessmentPolicyID)
    1
  #else
    99
  #end,
```

```
  "raw_data": $inputEvent
}
#end
```

Listing 5: LLM generated VTL template for OCSF event class "Application Lifecycle" for Crowdstrike using input event schema

```
Score: 0.9
Assessment: The VTL template is well-designed and
     accurately maps most of the relevant input fields to
      the corresponding output fields defined in the JSON
      schema. It handles various cases and conditions to
     map the input data correctly. However, there are a
     few minor issues:

1. The 'continent' field is hardcoded to 'unknown'
     instead of being derived from the input data.
2. The 'country' field is hardcoded to 'unknown' instead
     of being derived from the input data.
3. The 'vendor_name' and 'version' fields are hardcoded
     to 'unknown' if the corresponding input fields are
     missing, instead of being left null.
4. The 'instance_uid', 'interface_name', and '
     interface_uid' fields are hardcoded to null instead
     of being derived from the input data (if available).
5. The 'email_addr' field is hardcoded to null instead of
      being derived from the input data (if available).
6. The 'timezone_offset' field is hardcoded to 0 instead
     of being derived from the input data (if available).

Overall, the template effectively maps most of the input
     fields to the output schema, and the issues
     mentioned above are relatively minor. With some
     additional improvements, the template can be made
     even more accurate and comprehensive.
```

Listing 6: LLM generated confidence score and qualitative assessment for Crowdstrike's VTL template for OCSF event class "Application Lifecycle"

6 FUTURE WORK

The event classification performance can be improved by adding more domain specific information to the prompt, like description of the ISV, resource types supported by the ISV, resource types affected by the input event according to ISV documentation, etc. Qualitative assessment generated as a part of the event classification evaluation prompt can also be used as feedback to re-generate the event classification output if the generated confidence score is lesser than a preset threshold, in the same way as described for the template generation task in Section 3.3. We observe that the Claude Sonnet model used for evaluation generally gives high confidence scores for most templates, ranging from 0.7 to 0.9, even when it identifies a substantial number of problems with the generated VTL template. Hence, prompt engineering can be done to generate a more interpretive confidence score that aligns with the generated qualitative assessment. Other state of the art LLMs can also be explored for both the event classification and template generation tasks.

7 CONCLUSION

In this paper, we investigate the ability of Claude V3 class of LLMs to normalize audit logs from ISVs by first classifying the input events in these logs to OCSF event classes, and then using the schema for these event classes to generate VTL templates to convert the input events into the format specified by the schema. We perform both these generation tasks using zero shot learning. We achieve a 2X boost in classification accuracy as compared to human annotators and a speed up of 4X in generation of VTL templates. This automated method for generating templates for normalizing audit logs helps reduce human effort and human error and paves the way for better data analysis and reporting across different kinds of applications.

REFERENCES

[1] Youngjin Chae and Thomas Davidson. 2023. Large Language Models for Text Classification: From Zero-Shot Learning to Fine-Tuning. https://doi.org/10.31235/osf.io/sthwk

[2] A. Chuvakin, K. Schmidt, and C. Phillips. 2012. *Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management.* 1–434 pages.

[3] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2023. Chain-of-Verification Reduces Hallucination in Large Language Models. arXiv:2309.11495 [cs.CL]

[4] Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards Reasoning in Large Language Models: A Survey. arXiv:2212.10403 [cs.CL]

[5] Pierre Lepagnol, Thomas Gerald, Sahar Ghannay, Christophe Servan, and Sophie Rosset. 2024. Small Language Models are Good Too: An Empirical Study of Zero-Shot Classification. arXiv:2404.11122 [cs.AI]

[6] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024. Long-context LLMs Struggle with Long In-context Learning. arXiv:2404.02060 [cs.CL]

[7] Shijun Liu, Liwen Wang, Xiangxu Meng, and Lei Wu. 2012. *Dynamic Interoperability Between Multi-Tenant SaaS Applications.* 217–226. https://doi.org/10.1007/978-1-4471-2819-9_19

[8] Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei Huang, and Huajun Chen. 2023. Reasoning with Language Model Prompting: A Survey. arXiv:2212.09597 [cs.CL]

[9] Matthew Renze and Erhan Guven. 2024. The Effect of Sampling Temperature on Problem Solving in Large Language Models. arXiv:2402.05201 [cs.CL]

[10] Narongsak Sukma, Wasin Srisawat, Prush Sa-nga ngam, and Adisorn Leelasantitham. 2019. An Analysis of Log Management Practices to reduce IT Operational Costs Using Big Data Analytics. In *2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON).* 1–5. https://doi.org/10.1109/TIMES-iCON47539.2019.9024400

[11] Zhiqiang Wang, Yiran Pang, and Yanbin Lin. 2023. Large Language Models Are Zero-Shot Text Classifiers. arXiv:2312.01044 [cs.CL]

[12] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601 [cs.CL]

[13] Quan Yuan, Mehran Kazemi, Xin Xu, Isaac Noble, Vaiva Imbrasaite, and Deepak Ramachandran. 2023. TaskLAMA: Probing the Complex Task Understanding of Language Models.

[14] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. arXiv:2205.10625 [cs.AI]