**Documentation**

**Table of contents**

[**Introduction** 2](#_Toc183594234)

[**Development of the heuristic extractor instance** 3](#_Toc183594235)

[**Design iteration 1** - First Complexity level (One textual description per event) 3](#_Toc183594236)

[**Design iteration 1.1** - Application of the first complexity level on the recruiting process 4](#_Toc183594237)

[**Design iteration 1.2** - Application of the first complexity level on the logistics process 8](#_Toc183594238)

[**Design iteration 1.3** - Application of the first complexity level on the P2P-process 12](#_Toc183594239)

[**Level closing evaluation for complexity level 1** - Application of the first complexity level on the Order-management-process 16](#_Toc183594240)

[**Design iteration 2** - Second Complexity level (Textual descriptions for grouped, but intersection-free events) 17](#_Toc183594241)

[**Design iteration 2.1** - Application of the second complexity level on the recruiting process 17](#_Toc183594242)

[**Design iteration 2.2** - Application of the second complexity level on the logistics process 20](#_Toc183594243)

[**Design iteration 2.3** - Application of the second complexity level on the P2P process 22](#_Toc183594244)

[**Level closing evaluation for complexity level 2** - Application of the second complexity level on the Order-management-process 23](#_Toc183594245)

[**Design iteration 3** - Third Complexity level (Textual descriptions for grouped and intersecting event sets) 24](#_Toc183594246)

[**Design iteration 3.1** - Application of the third complexity level on the recruiting process 24](#_Toc183594247)

[**Design iteration 3.2** - Application of the third complexity level on the logistics process 25](#_Toc183594248)

[**Design iteration 3.3** - Application of the third complexity level on the P2P process 26](#_Toc183594249)

[**Level closing evaluation for complexity level 3** - Application of the third complexity level on the Order-management-process 28](#_Toc183594250)

[**Development of the generative extractor instance** 28](#_Toc183594251)

[**Development of the combined extractor instances** 31](#_Toc183594252)

[**Results** 31](#_Toc183594253)

[**Detailed Results** 31](#_Toc183594254)

[Recruitment log 31](#_Toc183594255)

[Logistics log 33](#_Toc183594256)

[P2P log 34](#_Toc183594257)

[Order-Management log 36](#_Toc183594258)

[Production log 38](#_Toc183594259)

[Age of empires log 40](#_Toc183594260)

[**Consolidated results** 42](#_Toc183594261)

# **Introduction**

In this documentation, we will describe the development of an extractor artefact that extracts object-centric event logs in OCEL 2.0 format from textual descriptions. It consists of a collector and refiner subcomponent. The goal of the collector is to extract extensive information from textual descriptions, while the refiner integrates, cleans, and refines the extracted information from multiple descriptions. The overall functionality is illustrated in figure 1.

Ein Bild, das Text, Screenshot, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

Figure 1: Functionality of the extractor artefact

The artefact is instantiated in four variants: a heuristic HEU-HEU extractor (heuristic collector and heuristic refiner), a generative GEN-GEN extractor (generative collector and generative refiner), and two hybrid extractors, namely GEN-HEU extractor (generative collector and heuristic refiner) and a HEU-GEN extractor (heuristic collector and generative refiner). The four variants are illustrated in figure 2.

Ein Bild, das Text, Screenshot, Schrift, Quadrat enthält.

Automatisch generierte Beschreibung

Figure : Four extractor variants

Development wise, we start with the development of the heuristic HEU-HEU extractor that comprises the heuristic collector and refiner subcomponents. Afterwards, we continue with the implementation of the generative GEN-GEN extractor, consisting of the generative subcomponent variants. Lastly, we set up the hybrid extractors, combining the previously developed subcomponents. After the development is completed, we conduct an artificial evaluation of all extractor variants against different datasets to compare them against each other.

# **Development of the heuristic HEU-HEU extractor**

The heuristic HEU-HEU extractor, consisting of a heuristic collector and a heuristic refiner, is developed within a development framework that enables continuous evaluation. The structure of the development framework can be analyzed in figure 3.

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

Figure 3: Development framework

The development framework follows a reversed engineering approach and consists of three main components – a *Generator*, *Extractor*, and *Comparison* instance. Firstly, the *Generator* instance receives an original log in OCEL 2.0 format and transforms it into a corresponding textual description leveraging LLM capabilities. Afterwards, the *extractor* instance – here it corresponds to the HEU-HEU extractor - analyzes the textual descriptions and tries to re-create the original log. Therefore, it uses a *collector* and *refiner* subcomponent that respectively firstly collects all relevant information from the textual descriptions and then cleans and refines the resulting preliminary OCEL snippets using NLP capabilities. Lastly, the original log and the extracted log are compared in the *comparison* instance that results in different quantifiable measures that indicate the similarity of the two event logs on different levels. The different levels thereby compare for example the extraction quality of different object types, object instances, object instance-type mappings, event types and event instances.

This development framework is applied on three different event logs - a recruiting event log, a logistics event log and a P2P event - over three complexity levels, resulting in nine design iterations. Furthermore, at the end of each complexity level, a level closing evaluation is conducted on unused event log – the order management log. The overall development logic can be analyzed in figure 4.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Figure 4: Development logic with nine design iterations (DI) and three level closing validations (CV)

In the first complexity level, the generator instance creates one textual description per event while minimizing noise. In the second complexity level, textual descriptions are grouped for multiple events, while enforcing an intersection-free grouping. In the third complexity level, this intersection-free requirement is abolished and descriptions that describe the same event from different perspectives are allowed. This increase in complexity should ensure that the textual descriptions become more and more realistic with every iteration without making the initial setup too complex.

Within one complexity level, the development framework is applied on all three training event logs – the recruiting event log, logistics event log and P2P event log with each exhibiting different characteristics – which results in three design iterations per complexity level. Therefore, the event logs are separated into a training subset consisting of the first 100 events, a validation subset consisting of the next 100 events, and a test subset containing 1000 unused events. The adaption of the HEU-HEU extractor will always be conducted while using the training subset. After the ‘training’ is finalized, the validation will be conducted using the validation subsets. If the comparison instance yields satisfactory results (F1-Score > 50%) over all quality categories on the validation datasets, a change to the next higher complexity level is conducted. If the results are not satisfactory, we return to the training subset within the same design iteration and try to further improve the HEU-HEU extractor. The level closing validation is always conducted on the validation subset of the order management log. The test subsets as well as the development framework itself will be used in the final evaluation of all extractor variants again.

## **Design iteration 1** - First Complexity level (One textual description per event)

During the first design iteration – the first complexity level – the complete development framework is built up using the python programing language. The resulting python project is structured as follows:

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Figure 5: Folder structure of the python project

Within this python project, 0\_Train\_Validation\_Test\_split\_creator\_instance creates a train-validation-test-split of the event logs stored within the Data-folder. 1\_Generator\_instance, 2\_Extractor\_instance, and 3\_Comparison\_instance then correspond to the *Generator*, *Extractor* and *Comparison* instance defined in figure 3. Besides that, the Documentation-folder contains this documentation, while the Results-folder will store the results.

For the development of the HEU-HEU extractor, we use three training event logs and one level closing validation log in the OCEL2.0 format that are publicly available under the OCEL-standard website (<https://ocel-standard.org/event-logs/>). These event logs correspond to a recruiting log (<https://zenodo.org/records/8433706>),a logistics log (<https://zenodo.org/records/8428084>), a P2P log (<https://zenodo.org/records/8412920> ) and an order management log (<https://zenodo.org/records/8428112> ).

The recruiting process is a simplified legacy event log that was originally built for the OCEL1.0 format but was then transferred to the OCEL2.0 log format. This event log was selected as it is a simple, easy-to-understand, simulated event log and therefore, represents and easy starting point for the development of our pipeline. The event log contains 1505 object instances that correspond to 6 object types as well as 6980 event instances corresponding to 16 event types. However, no additional attributes are caught for object instances and types and the object qualifier in event-to-object-relationships is not defined. Furthermore, no object-to-object relationships are tracked at all.

On the other hand, the more complex logistics process was directly built for the OCEL2.0 standard and therefore contains more information than the recruiting process. In total, the logistics process contains 13910 object instances scattered over 7 object types as well as 35413 event instances that correspond to 14 event types. In contrast to the recruiting process, the logistics process catches additional attributes for object instances and types as well as object-to-object-relationships. However, it doesn’t catch event attributes.

The P2P log, in contrast, is the most complex log, catching all OCEL components, including event attributes. It contains 14671 events scattered across 10 event types and 9543 objects corresponding to 7 object types.

Lastly, the order management log is used for the level closing validations. Similarly to the logistics log, it catches object attributes and object-to-object relationships but misses event attributes. It contains 21008 events corresponding to 11 event types and 10841 objects according to 6 object types.

Using the 0\_Train\_Validation\_Test\_split\_creator\_instance, we create training, validation and testing subsets of the four event logs. Thereby, the training and validation size per event log comprises 100 events with its corresponding event types, object types, and object instances. The training dataset will be used to develop the pipeline and to analyze different cases. The validation set will be used at the end of each design iteration to measure the quality of this design iteration and to decide if to proceed to the next design iteration. A switch to a more complex design iteration is only conducted, if the validation dataset reaches a F1-Score over 50% overall categories. 1000 unused events are then assigned to the testing subsets. These testing subsets are not used during the development of the HEU-HEU extractor. Instead, they will be used at the very end for the evaluation of all extractor variants.

To make the development of our HEU-HEU extractor and its development framework easier, we start with the training subset of the simplified recruiting process within the first complexity level stage.

### **Design iteration 1.1** - Application of the first complexity level on the recruiting process

In the following subchapter, we will first report on the status of the development pipeline at the end of this design iteration, before we will report on the results that we achieved using this pipeline.

#### Status of the development pipeline at the end of design iteration 1.1

Within this design iteration the initial development framework was created using the recruiting training dataset. Thereby, multiple iterations have been conducted with the training dataset to get an overview over all cases and variants that can appear. During these iterations, the framework has been further developed and adapted whenever necessary until we reached a satisfactory status.

##### Status of the generator instance

We started our development with the generator instance (1\_Generator\_instance) that should be used to create one textual description per event. Within the generator instance, a separator-function primarily divides the training log into further subsets. In this complexity level, every subset corresponds to an event log comprising exactly one event. These separated sub event logs are then provided one after each other to a report-creator-function that uses the OpenAI-API with its assistance-functionalities over an Azure endpoint to create one textual description per event. Specifically, the GPT3.5-turbo model is thereby used in the background. At this stage, the following system prompt (‘instructions’) and user prompt (‘summary\_request’) are used for the generation of the textual descriptions:

instructions = """You are a process mining expert. Use your knowledge base to summarize the provided OCEL2.0-logs in natural language. The report should describe all activities in the event log as well as all objects that were involved in the corresponding activity. Please make it sound as natural and human-like as possible by writing everything in one text without using bullet points. Ensure that you summarize all activities mentioned in the OCEL2.0-log properly,  
but don't come up with any new activities. Integrate all timestamps, object types and object labels/IDs in an intuitive way into the text. Use only standard utf-8 characters. Don't mention the events, objects or types specifically, but rather integrate them implicitly into the text. Make the description as concise and as short as possible describing one event with one or maximum two sentences and leave out unnecessary descriptions that are not specified within the log. Write everything in an objective tone where you simply describe what happened. Don't refer to the uploaded log, but rather describe it in a natural way without leaving any details out. Don't put any events into parentheses.   
It is very important that the generation is lossless, and all IDs are included.

Here are some examples on how your answers could look like:

'The manager named XY opened a vacancy for the position of Manager on May 20, 2019 at 12:26:57 UTC, with the ID Vacancy[550001] - Manager.'

'On May 20, 2019 at 15:45:06 UTC, the applicant XY submitted an application Application[770002]'

'An applicant called XY submitted an application with the ID "Application[770006]" for the vacancy with the ID "Vacancy[550001] - Manager" on May 20, 2019 at 17:14:31 UTC.'

"""  
  
summary\_request = """Please create based on the OCEL2.0-logs in your knowledge base textual descriptions of the event logs. Your answer should consist only of the textual description without mentioning the OCEL-log or its objects and events it is based on. The descriptions should be as short as possible (maximum one or two sentences per event). Nevertheless, be very exact with all details provided in the log and integrate all timestamps, object types, and object labels/IDs in an intuitive way into the text. Don't refer specifically to the uploaded log, but rather describe it in a natural way without leaving any details out. It is very important that the generation is lossless, and all IDs are included."""

##### Status of the extractor instance

After all textual descriptions have been successfully created and saved to a specific folder, the extractor instance (2\_Extractor\_instance) takes over. The extractor instance, corresponding to the HEU-HEU extractor, consists of two subcomponents: The heuristic collector and the heuristic refiner. The task of the collector is to collect all necessary information per textual description and to bring it into a preliminary order. In the case of the recruiting process this means that the heuristic collector extracts one timestamp, one activity, multiple candidate object instances and multiple candidate object types per textual description. Besides that, it tries to create an initial mapping of object instances to object types and refines the extracted candidate object types based on the object type mapping. If no mapping from an object instance to a type was found, then a dummy ‘Object\_type\_not\_identified’ object type is assigned. For all these collection mapping and refinement steps, the heuristic collector uses a number of different NLP techniques, including but not limited to named-entity-recognition, tokenization, part-of-speech-tagging, and lemmatization. The goal was thereby to create a heuristic collector that is universally applicable and is not specified on the recruiting process by pre-defining any event types or objects. Instead, it for example extracts verbs as activities and DATE- and TIME-entities as timestamps (this is a very simplified description on how the collection and mapping function works that only serves explanation purposes and does not 100% correspond to the actual functions). Using the information collected on one textual description, the information is structured into the OCEL2.0 format and exported into a new event log. In this manner, for every textual description corresponding to one event a new event log is created that also comprises exactly one event. These newly created event logs are then handed over to the heuristic refiner-subcomponent.

The heurisitic refiner-subcomponent then concatenates the extracted event logs to one big, concatenated event log that corresponds in size to the original training subset. Then several refinement steps are applied on the whole event log leveraging NLP- as well as majority-based techniques. The main difference between the collector and the refiner is therefore that the collector is based on only one textual description, while the refiner can use the knowledge of all extracted event logs leveraging group knowledge advantages. The refinement steps are separated into object refinements, event refinements, and object-event-refinements.

On an object level, it is firstly ensured that all object types that are assigned to an object instance are also present in the object-types-dictionary. Then, the names of the object types are cleaned by singularizing and capitalizing their names, while removing extraneous elements (only letters allowed). Afterwards, prohibited pre-defined object types (in this case, ‘ID’ and ‘Name’) are removed from the object types. As a next step, the object IDs of the object instances are cleaned by capitalizing all words in the object id. Afterwards, if a substring of the ID of an object instance corresponds to an object type, this object type is automatically assigned to the instance. Then, object instances with the same ID but different assigned object types are merged. Thereby, the most common object type is accepted as the real object type (except the object type corresponds to ‘Object\_type\_not\_identified’). Then, the object instances and types are tested for substrings of each other. If one object instance or type is a substring of another object instance or type, the substring is removed while handling a correct object typing mapping for the object instance case. Lastly, for object instances of the same object type the formatting is assimilated by identifying the most common pattern per object type. During all these steps, duplicates are removed regularly, and all changes are always propagated to the corresponding affected events.

On the event level, further refinement steps are applied. These comprise as well cleaning the event names (allowing for only letters; removing PERSON-entities, if they are also an object instance; removing repeated words), removing duplicates, merging event types, if they are substrings of each other, and merging event types that have a semantic similarity to each other over 80%. Lastly, on the event-object-level object types are tried to be inferred for object instances with ‘object\_type\_not\_identified’ based on the event type they are assigned to. All these refinement steps are applied twice to the concatenated event log, resulting in the final refined event log. This final event log is then handed over to the comparison instance.

##### Status of the comparison instance

The comparison instance then reads the original training log and the newly extracted and refined event log and compares them to each other. The comparison of the old and the new event log is conducted on different levels. Within these complexity levels, we differentiate between parent levels and child levels. The parent levels contain the categories object type, object instance, event type, and event instance. Within this level, we test if the corresponding entities exist correctly in the new log. Therefore, we compare the names/IDs of the object types, object instances, and event types in the original and extracted log with each other and try to find correct matchings. For the event instance level, we define an event instance as a combination of a timestamp and an event type (as the event ID normally doesn’t carry valuable information – in the recruitment process for example, the event ID is simply an increasing number) and try to find matchings based on this definition as well.

Based on the parent levels, we defined further child levels. The child levels currently comprise the matching of object instances to object types and the relationship analysis of event instances to object instances. Child levels are always calculated in an absolute and relative form. The absolute form of the object-to-type matching thereby tests for example over the complete log, if correct object-type-mappings exist, while the relative form tests only for the correctly identified and matched object instances, if they also comprise the correct object type. The same logic of absolute and relative form applies to the event-to-object-mapping-level as well. Lastly, we also define an overall log level that averages all results of the other levels excluding relative child levels.

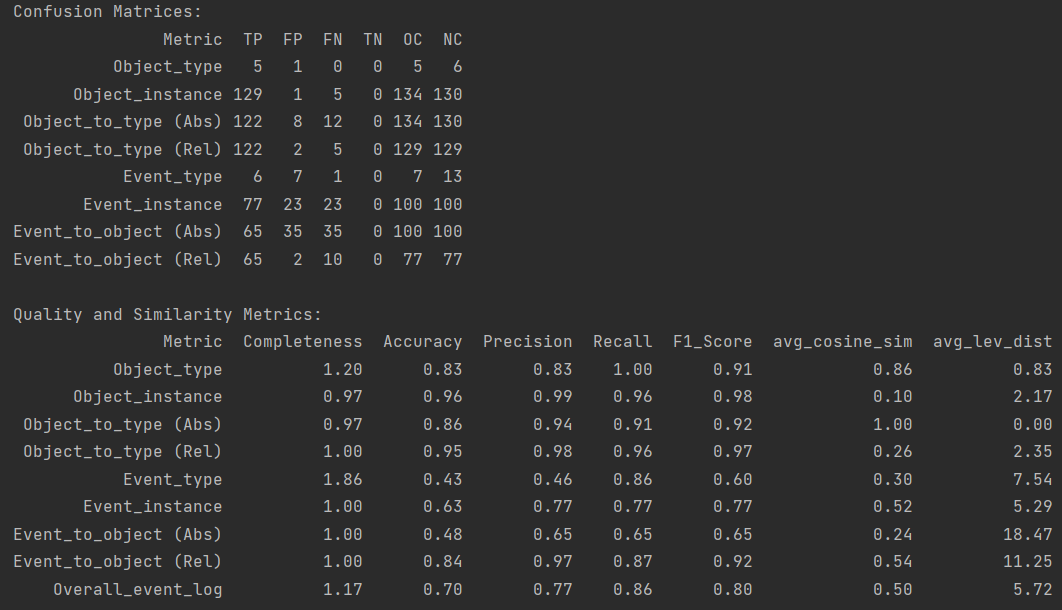
Besides that, we calculate different measures per comparison level. These different measures can be divided into confusion-matrix based measures and similarity measures. For the confusion-matrix based measures, we first calculate a confusion-matrix per comparison level by defining the original log as the gold standard (the correct values) and the new log as the prediction (the predicted values). Furthermore, we define True Positives (TP) as entities in the old log that have been correctly extracted in the new log, False Negatives (FN) as entities in the old log that are missing in the new log, and False Positives (FP) as entities in the new log that does not exist in the original log and are therefore additional. True Negatives (TN) are not calculated in our Use Case as the search for entities that are correctly not captured in the new log would result in infinity. Instead, True Negatives are automatically defined as equal to 0.

For the calculation of the confusion matrix, we start at the parent levels. Within the parent levels, we first try to create a match between an entity in the new log to an entity in the old log. The matching is thereby conducted using a similarity threshold over 70% to account for synonymous entities. If a match is found, it is assigned to the category of True Positives. Thereby, it is ensured that one entity in the old log is only matched to the best fitting entity in the new log. After all matches are found and counted as True Positives (TP), the remaining entities in the new log are counted as superfluous and therefore False Positives (FP), while the remaining entities in the old log are count as missing and therefore False Negatives (FN). This logic applies also to the child levels in absolute form. Within the absolute object-to-type-level, we for example search for mappings that match each other in the object instance ID as well as in the object type name and count them as True Positives. Entities in the original and extracted log that are not part of this mapping are then accordingly assigned to False Negatives and False Positives. This calculation, however, differs for relative child levels as these only analyze the previously identified matches in the parent level instead of the whole logs. This means for the relative object-to-type-level that only object instances that have been counted as True Positive in the object instance parent level are analyzed regarding their correct object-to-type matching. Correct matches of object types per object instance are then further counted as True Positives while wrong matches are further analyzed. If a wrong matching appears, because no object\_type at all has been found for this object instance, the entity is counted as missing and therefore False Negative. In contrast, if a wrong match appears, because a wrong object type has been assigned to the object instance, it is counted as False Positive. This logic stretches over all relative child levels. After all confusion matrices for all levels have been calculated, the accepted formulas for accuracy (), precision (), recall () and F1-Score () can be used to calculate those measure per comparison level. Furthermore, a completeness-measure is calculated that corresponds to the count of the new log divided by the count of the old log.

Beside these confusion-matrix based measure, we also calculate similarity-based measures. These measures comprise the cosine similarity of two entities as well as the levenshtein distance between two entities. Therefore, we compare for example on object-type-level all object types in the old event log with all object types in the new event log and try to find the matches with the highest cosine similarity or the smallest levenshtein distance. After finding this maximum similarity or minimum distance per match, we calculate the average maximum similarity or average minimum distance per complexity level. This similarity calculation is conducted again on the whole log for the parent levels and the absolute child level, while the relative child levels only integrate instances that have been previously classified as correct by its parent level. The combination of these different measures over different complexity levels gives us a good indication of the quality of the extractor.

#### Results on the validation log at the end of design iteration 1.1

After reaching satisfactory comparison results on the training dataset, we conduct a final evaluation of our current pipeline using the validation dataset that hasn’t been used during the development of the framework. The following results have been obtained:



As the F1-Score over all categories reaches results >50% and the overall event log F1-Score even reaches a score of 80%, we decide to continue with the next design iteration.

### **Design iteration 1.2** - Application of the first complexity level on the logistics process

#### Status of the development pipeline at the end of design iteration 1.2

Due to the switch from the simplified recruiting process to the more complex logistics process, multiple adaptions had to be made to all parts of the development pipeline.

##### Status and adaptions of the generator instance

The first problems arose, when we tried to use the original generator instance to create textual descriptions of the logistics process. As for the recruiting process, we firstly separated the logistics training dataset per event into sub event logs and then tried to send these sub event logs to the OpenAI API to create new textual descriptions per event log. However, the generation of the textual descriptions failed partially as it exceeded the maximum allowed token limit of the API. In general, we found out that some of the sub event logs were simply too big for the API to be processed correctly. Therefore, we tried different workarounds to handle this size limit constraint, however, it often resulted in an unreliable generation of the textual descriptions that were either wrong or lost important information on the fly. As the proper, lossless generation of textual descriptions is a key requirement for the proper functioning of the development pipeline, we therefore decided to limit the allowed size of the sub event logs to 2kb. Event logs exceeding this limit are simply deleted from the pipeline. For the training dataset of the logistics process this resulted in 5 excluded event logs leading to a total size of the logistics training dataset of 95 events. This requirement as well as some more exception handling functions were integrated into the generator instance to ensure the proper generation of textual descriptions. Besides that, this requirement is also the reason why we decided for the logistics process and against the order-management and P2P-process as their sub event logs were on average significantly bigger than the sub event logs of the logistics process. To ensure a proper generation of the textual descriptions, we further adapted the system and user prompts to also catch the additional attributes and relationships:

instructions = """You are a process mining expert. Use your knowledge base to summarize the provided OCEL2.0-logs in natural language. The report should describe all activities in the event log as well as all objects that were involved in the corresponding activity. Please make it sound as natural and human-like as possible by writing everything in one text without using bullet points. Ensure that you summarize all activities mentioned in the OCEL2.0-log properly, but don't come up with any new activities. Integrate all timestamps, object types and object labels/IDs, relationships and attributes in an intuitive way into the text. For the relationships, also include the qualifier. Use only standard utf-8 characters. Don't mention the events, objects, types, relationships, and attributes specifically, but rather integrate them implicitly into the text. Make the description as concise and as short as possible and leave out unnecessary descriptions that are not specified within the log. Write everything in an objective tone where you simply describe what happened. Don't refer to the uploaded log, but rather describe it in a natural way without leaving any details out. Don't put any events or types into parentheses. It is very important that the generation is lossless, and all IDs, attributes and relationships are included.

Here are some examples on how your answers could look like:

'The manager named XY opened a vacancy for the position of Manager on May 20, 2019 at 12:26:57 UTC, with the ID Vacancy[550001] - Manager.'

'A vehicle with ID "vh5" was booked for a transport document with ID "td3" on May 26, 2023, at 09:53:25 UTC. The transport document with ID "td3" had 3.0 containers and was in transit. However, on June 13, 2023, at 09:00:00 UTC, its status changed to "shipped". The vehicle with ID "vh5" had a departure date of June 13, 2023, at 11:00:00 UTC.'

"""

summary\_request = """Please create based on the OCEL2.0-logs in your knowledge base textual descriptions of the event logs. Your answer should consist only of the textual description without mentioning the OCEL-log or its objects and events it is based on. The descriptions should be as short as possible. Nevertheless, be very exact with all details provided in the log and integrate all timestamps, events, object types, object labels/IDs, attributes, and relationships in an intuitive way into the text. Don't refer specifically to the uploaded log, but rather describe it in a natural way without leaving any details out. It is very important that the generation is lossless, and all IDs are included."""."""

However, even though we use the system and user prompt to guide the LLM in its generation and instruct it to include all details, it still sometimes hallucinates or leaves important information out. Therefore, an 100% lossless and correct generation of the textual descriptions is not longer guaranteed. But as there are no replacement datasets that could be used for the evaluation of our artefact, we opted to stay with the current approach.

##### Status and adaptions of the extractor instance

As the goal of design iteration 1.2 is to also extract object-attributes and object-to-object-relationships several adaptions had to be applied to the HEU-HEU extractor instance, consisting of the heuristic collector-subcomponent and the heuristic refiner-subcomponent.

Using NLP-techniques, the heuristic collector-subcomponent now also extracts candidate values for attribute names, attribute values and relationship qualifier. Furthermore, instead of only extracting one candidate timestamp and one candidate activity that is matched to each other, it extracts now multiple candidate values. This adaption was necessary as attributes also contain a timestamp component that must be mapped accordingly. Besides that, due to the longer texts with in general more information the direct extraction of the correct activity wasn’t possible anymore and was therefore replaced with later refinement steps. After all candidate values (timestamps, activities, object labels, object types, attribute labels, attribute values, and relationship qualifier) have been extracted, the mapping of those values to each other must be conducted. To conduct this mapping, we retrieve for all candidate values their related tokens in the form of children and adolescent tokens from the text. Using these related token relationships as well as the position of one element in the text compared to the other elements, we try to conduct the mapping between the different candidate values. Within this mapping step, object labels are mapped to object types, attribute values are mapped to attribute labels, timestamps are mapped to activities, and timestamps are mapped to attribute values. Furthermore, object labels are mapped to other object labels to retrieve object-to-object relationships including their relationship qualifiers and activities are mapped to attribute values to exclude wrongly extracted activities. Lastly, attribute values are also mapped to object labels to then derive the correct object type – attribute label mapping using the prior transitive relationships. All these mappings are necessary to finally create one event log per textual description.

After all different event logs have been extracted, they are concatenated and refined in the heuristic refiner subcomponent. Firstly, we adapted the existing functions in the refiner subcomponent so that they also handle the new event log structure correctly. Then we integrated numerous additional refinement steps into the subcomponent to also effectively refining object attributes and object-to-object-relationships, while improving the other functions. These functions include the cleaning of the attribute names; the removal of attribute names and values that correspond to other objects, object types, events, or event types; the merging of attributes that have similar attribute names; the merging of identical attribute values for different attribute types; the merging of equivalent numerical attribute values that only differ in their representational form (e.g. one attribute is in the form of a float and the other in the form of an integer); the removal of duplicate attribute types, and the adaption of the coherence of attribute types in the object types list and the objects list in the event log. Besides that, we included some more functions for the better handling of object types and object names that became necessary because of the increased complexity. Specifically, we adapt the object-label-to-object-type mapping based on similar object names (similarity groups) in the event log. Conversely, we adapt object names based on the object types, if the name structure does not fit the rest of the similarity group. Therefore, we screen the object attributes for values that should be the replacement for the object name. This step is necessary as sometimes object labels are wrongly identified as attribute values. Lastly, we screen all attribute values for a fit to a similarity group and convert them to objects if necessary. Within the event refinement step, we furthermore remove object instances with the label ‘object\_instance\_not\_found’ from the event-to-object-relationship, remove events that do not feature any event-to-object-relationships, and merge similar event types based on the connected event-to-object-relationships (this function thereby replaced the general merging of event types with similar names as this merging step did not accounted for the importance of related objects). After we included all these refinement steps, we achieved on event log that was ready for the evaluation within the comparison instance.

##### Status and adaptions of the comparison instance

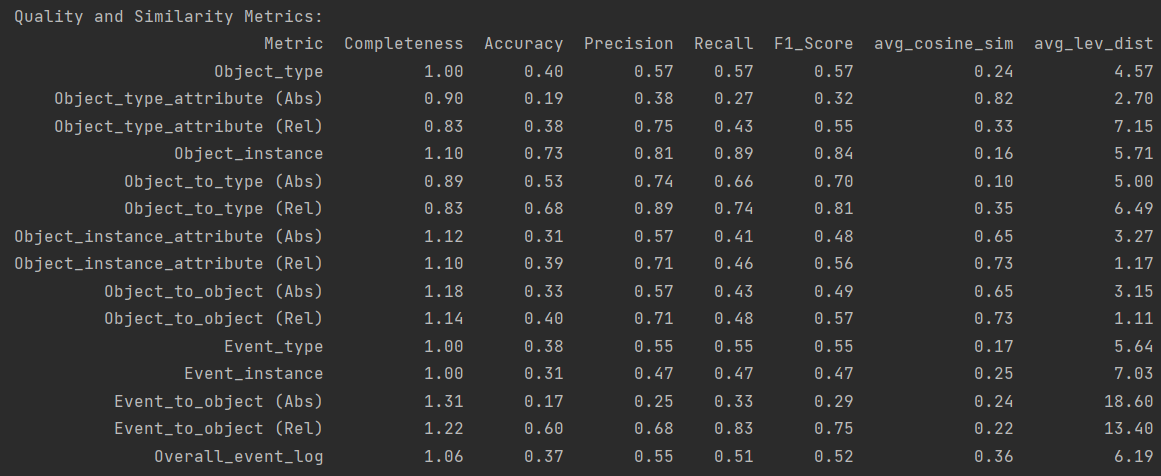
Due to the increased complexity of the logistics process, we decided to integrate three additional child levels in absolute and relative form into our comparison instance to account for the increased information richness. These child levels account for the correct mapping of attribute names to object types, attribute values to object instances, and object-to-object-relationships. The logic behind the calculation of the absolute and relative child level corresponds to the logic used in the previous child levels. Furthermore, to increase the efficiency and correctness, we corrected identified inconsistencies in the pipeline and integrated the Hungarian bi-partite matching algorithm, also known as the Kuhn-Munkres algorithm, into our comparison instance.

The Hungarian algorithm is an optimization algorithm that is used to solve the assignment problem by assigning a set of tasks to a set of agents such that the total cost or time is minimized (or the profit is maximized). By providing the similarity between two entities as weights to the Hungarian algorithm, it helps us to find the optimal combination of matches to maximize the overall similarity. To derive the similarity of two entities, we calculate the syntactic similarity (using the fuzzy ratio similarity) as well as the semantic similarity (using the spacy similarity) between those entities and accept the higher value of both similarities. In the prior design iteration, we only calculated the syntactic similarity based on the fuzzy ratio algorithm. After the Hungarian algorithm provided us with the optimal combinations, we re-test, if we accept these matchings as well. A matching is accepted if its maximum value is either >70% or if its semantic and syntactic similarity if >50%.

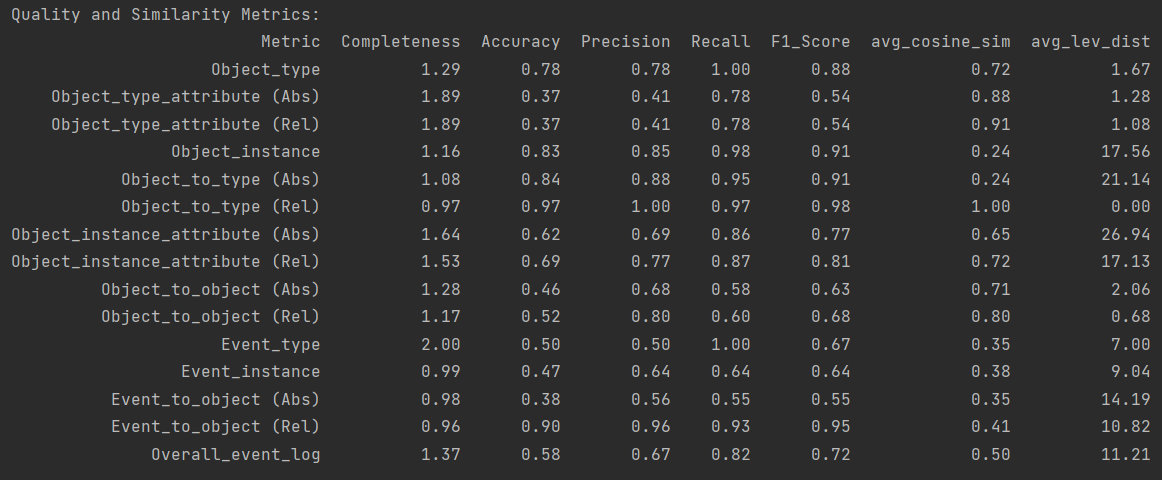
Lastly, we also integrated a more detailed analysis of the child levels into the comparison instance; specifically, the calculation of the confusion matrix for child levels where the elements under supervision consist of lists. Examples for such child levels are the object\_instance-attribute-levels and the object-to-object-level were multiple attributes or object relationships are assigned to one object instance. While beforehand, a perfect match for all attributes and object relationships was demanded, we now also accept partial correctness, if for example, 2 out of 3 relationships were correctly detected. Therefore, we create new tuples consisting of one element in the parent and one element on the child level. This means that for example obj1 with 3 attributes that was original counted as {ob1: [attr1, attr2, attr3]} is converted to {obj1: attr1}, {obj1: attr2}, and {obj1: attr3}. These newly extracted tuples are then used to re-calculate the TP, FP, and FN values according to the same rules that apply to the other levels.

#### Results on the validation log at the beginning and at the end of design iteration 1.2

Within design iteration 1.2, we conducted two validation rounds. The first round was conducted at the beginning of the design iteration 1.2 with the old HEU-HEU extractor instance that was the result of design iteration 1.1 (at this point only the necessary adaptions of the generator and comparison instance have been conducted). The second round was then conducted at the end of design iteration 1.2 with the new, adapted HEU-HEU extractor instance to see the effect of the adaptions. In the following figure, you can see the results for the logistics process validation set using the old extractor instance. It must be noted that the size of the validation dataset has been reduced from 100 events to 96 event due to the size restrictions.



Analyzing these results, it becomes clear that the status of the HEU-HEU extraction instance at the beginning of design iteration 1.2 did not yield satisfactory results (Multiple comparison levels yielded F1-Scores < 0.5). Therefore, we decided to further adapt the extraction instance to account for the increased complexity, specifically also extracting object type attribute names, object instance attribute values, and object-to-object relationships as these components of the OCEL-log have been neglected beforehand. The True Positive (TP) values in these categories at the beginning of design iteration 1.2 can be attributed to the fact that some object types and object instances simply didn’t have any attributes or relations at all and therefore, the non-existence of attributes and relations in the extracted log was counted as correct. After adapting the HEU-HEU extraction instance, we were able to achieve the following results:



Here it becomes apparent that we received an improvement over all categories with the F1-Score of the overall event log getting 0.2 Points higher, while also extracting additional object attributes and object-to-object relationships. As every F1-Score yields results >50%, we deem the results as satisfactory and continue with the next design iteration.

### **Design iteration 1.3** - Application of the first complexity level on the P2P-process

In design iteration 1.3 we focus on the P2P process that not only features objects, object types, events, event types, object attributes, object-to-object relationships, and event-to-object relationships, but also captures event attributes, a category that has been neglected in the previous logs and design iterations. Therefore, the goal of this design iteration is to make the extraction complete on the lowest complexity level by also extracting event attributes.

#### Status of the development pipeline at the end of design iteration 1.3

##### Status and adaptions of the generator instance

Even though we decided in design iteration 1.2 against the P2P process due to its sheer size, we were forced to use this one in design iteration 1.3 as it was the only publicly available event log on the OCEL2.0-standard website that features also event attributes, a category previously neglected. However, this event log in its original form was not usable due to the max 2kb size requirement that was always exceeded. Therefore, we integrated an event reducer at the beginning of the generator pipeline that reduces the size of the event log. Specifically, the event reducer allows only one non-empty attribute value per object, only one object-to-object relationship per object, and a maximum of five event-to-object-relationships in the reduced event log. Exceeding attributes or object-to-object relationships get simply deleted. With this adaption, we were able to achieve an event log that still features all possible categories while being small enough to be processes most of the time.

Besides that, we again adapted the system prompt slightly to account for the additional event attributes. The new system prompt is as follows:

instructions = """ You are a process mining expert. Use your knowledge base to summarize the provided OCEL2.0-logs in natural language. The report should describe all activities in the event log as well as all objects that were involved in the corresponding activity. Please make it sound as natural and human-like as possible by writing everything in one text without using bullet points. Ensure that you summarize all activities mentioned in the OCEL2.0-log properly, but don't come up with any new activities. Integrate all timestamps, object types and object labels/IDs, relationships and attributes in an intuitive way into the text. Don't add the event ID. Use only standard utf-8 characters. Don't mention the events, objects, types, relationships, and attributes specifically, but rather integrate them implicitly into the text. Never put events into quotation marks or mention the word 'event' itself. Make the description as concise and as short as possible and leave out unnecessary descriptions that are not specified within the log. Write everything in an objective tone where you simply describe what happened. Don't refer to the uploaded log, but rather describe it in a natural way without leaving any details out. Don't put any events, attributes or types into parentheses. It is very important that the generation is lossless, and all object IDs, attributes and relationships are included.

Here are some examples on how your answers could look like:

'The manager named XY opened a vacancy for the position of Manager on May 20, 2019 at 12:26:57 UTC, with the ID Vacancy[550001] - Manager.'

'A vehicle with ID "vh5" was booked for a transport document with ID "td3" on May 26, 2023, at 09:53:25 UTC. The transport document with ID "td3" had 3.0 containers and was in transit. However, on June 13, 2023, at 09:00:00 UTC, its status changed to "shipped". The vehicle with ID "vh5" had a departure date of June 13, 2023, at 11:00:00 UTC.'

"""

Furthermore, due to changes in the OpenAI Api, we were now able to change from the gpt 3.5 turbo model to the gpt 4o mini model that promises a faster generation combined with a higher generation quality.

##### Status and adaptions of the extractor instance

To reach the goal of also extracting event attributes, we adapted the heuristic collector and refiner subcomponent of the HEU-HEU extractor instance accordingly. Event attributes are custom attributes that are specific to an event instead of an object. Typical event attributes are the attributes ‘lifecycle’ and ‘resource’ according to ‘C. W. Gunther and H. M. W. Verbeek, “XES-standard definition’. The lifecycle event attribute can thereby take on different standard values, namely ‘schedule’, ‘assign’, ‘withdraw’, ‘reassign’, ‘suspend’, ‘resume’, ‘pi\_abort’, ‘ate\_abort’, ‘complete’, ‘autoskip’, ‘manualskip’, and ‘unknown’, while the ‘resource’ attribute indicates ‘the name, or identifier, of the resource having triggered the event’. As these event attributes are the most prominent ones, we put a special focus on extracting these. Therefore, we integrated specific event-lifecycle-status and event-resource-extractor into our collector-subcomponent. These are then later mapped to the final activity. Furthermore, we adapted our attribute\_value-to-object-mapper function to also map attribute-values to events, if possible, to account also for custom event attributes. After mapping all event attribute values to events and to event attribute types, we use the transitive relationship to also map event attribute types to event types. Lastly, the newly extracted event attributes get also integrated into the OCEL structure of the event logs so that we receive event logs per event that also contain event attributes.

Next to these main adaptions, we further improved the overall functionality of the heuristic collector instance by analyzing different Use Cases. Besides that, we restructured the heuristic collector instance so that it now differentiates between different steps. Specifically, the collector is now divided into the components NLP Preprocessing, Candidate extraction, Candidate mapping, and OCEL construction. The NLP Preprocessing step thereby parses once the complete document and thereby derives the tokens, their dependencies, their part-of-speech-tags, their entity types and their related tokens in the form of children and ancestor tokens. Afterwards, the candidate extractor uses this preprocessed text to extract candidate values for timestamps, activities, object labels, object types, attribute types, attribute values, event resources, event lifecycle status, and relationship qualifier and conducts a mutual exclusion step to directly exclude invalid candidate values (E.g. the category object types is not allowed to bear values that are part of the extracted timestamps). Furthermore, it extracts for all candidate values their related tokens. The candidate mapper then uses the related token relationships between values as well as the relative positions of tokens to each other in the text to conduct the mapping. In total, object labels are mapped to object types, attribute values are mapped to attribute types, object labels are mapped to other objects to extract O2O-relationships, activities are mapped to timestamps, attributes are mapped to timestamps, attribute values are mapped to objects and events, and attribute types are mapped object types and event types. Building on these mappings, the OCEL constructor then constructs the categories object types, objects, event types and events in combination with all their associated attributes and relationships and combines them to the sub-event log. This results in one sub-event log per document at the end of the collector step.

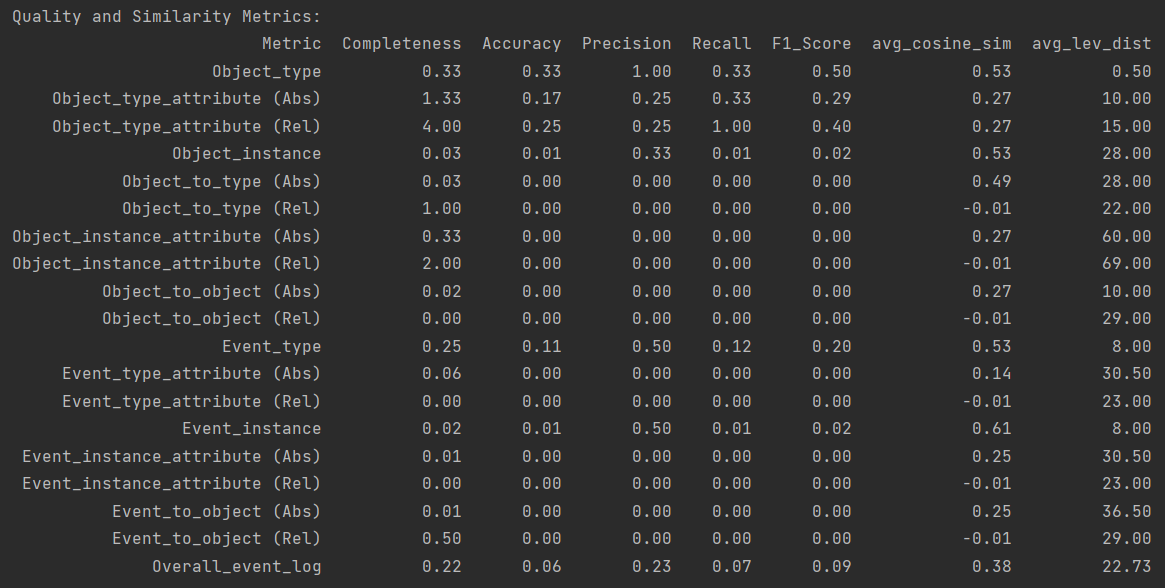
These sub-event logs are then provided to the heuristic refiner subcomponent. The refiner subcomponent was as well adapted to the new structure of the event log containing event attributes as well as improved and restructured for better readability. The refiner now first concatenates the previously extracted sub-event logs and then applies several cleaning and refinement steps on the concatenated event log in two iterations. These refinement steps included the cleaning of all names, another mutual exclusion step, the removal of identical or very similar entities over all categories, specific object and event refinement steps, as well as a last step that ensures that the object types in the log are aligned to the objects and the event types are aligned to the events. All these refinement step results in the final event log that will be used for the comparison.

##### Status and adaptions of the comparison instance

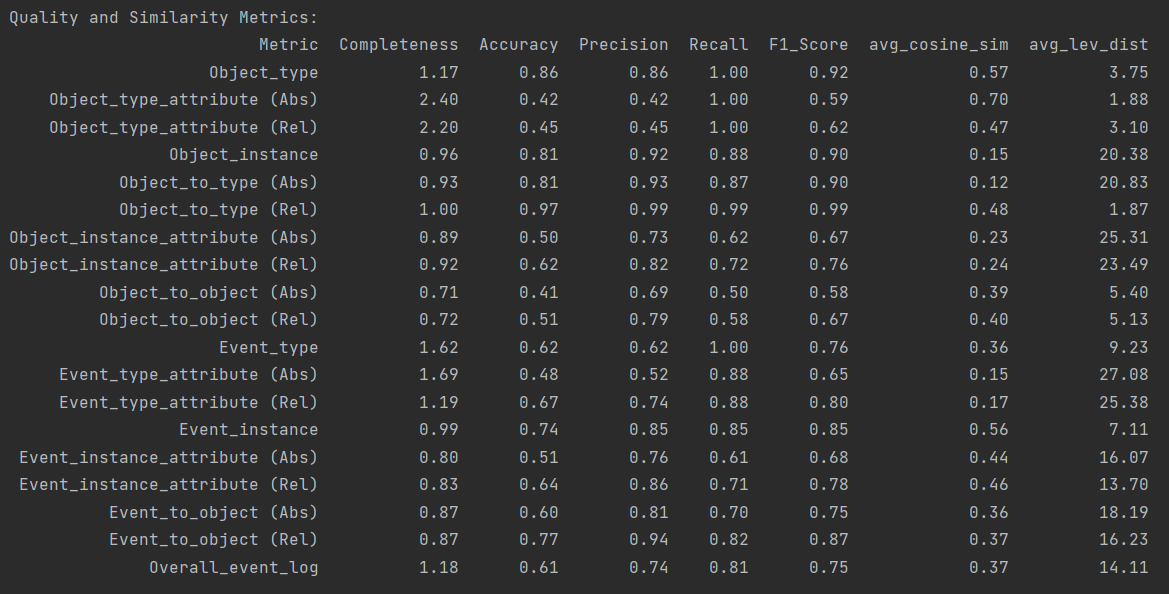
Similar to the adaptions in design iteration 1.2, we also had to integrate two additional child level in absolute and relative form into our comparison instance in design iteration 1.3 to account for the newly extracted event attributes. Specifically, the new child levels firstly analyze the mapping of event types to event attribute types and secondly compare the mapping of event instances to event attribute values. The logic behind the evaluation is thereby equivalent to the evaluation of object\_type-to-object\_attribute\_type-mapping and object\_instance-to-object\_attribute\_value-mapping. Besides that, we loosened our requirements for the object-to-object relationships. The reason therefore is that in the original log we saw sometimes the relationship from object 1 to object 2 but extracted it as a relationship from object 2 to object 1 due to the phrasing. In the last iteration, we counted such cases as FP and FN. However, we now decided to accept it as TP as the relationship itself is still correct.

#### Results on the validation log at the beginning and at the end of design iteration 1.3

We again conducted an evaluation at the beginning of design iteration 1.3 with the old HEU-HEU extractor instance (the extractor instance at the end of design iteration 1.2) and an evaluation at the end of design iteration 1.3 with the new HEU-HEU extractor instance to analyze the advancements achieved through design iteration 1.3. Therefore, we used the reduced validation dataset of the P2P process that has not been used for the adaption of the extractor instance. At the beginning of design iteration 1.3 we thereby received the following results:



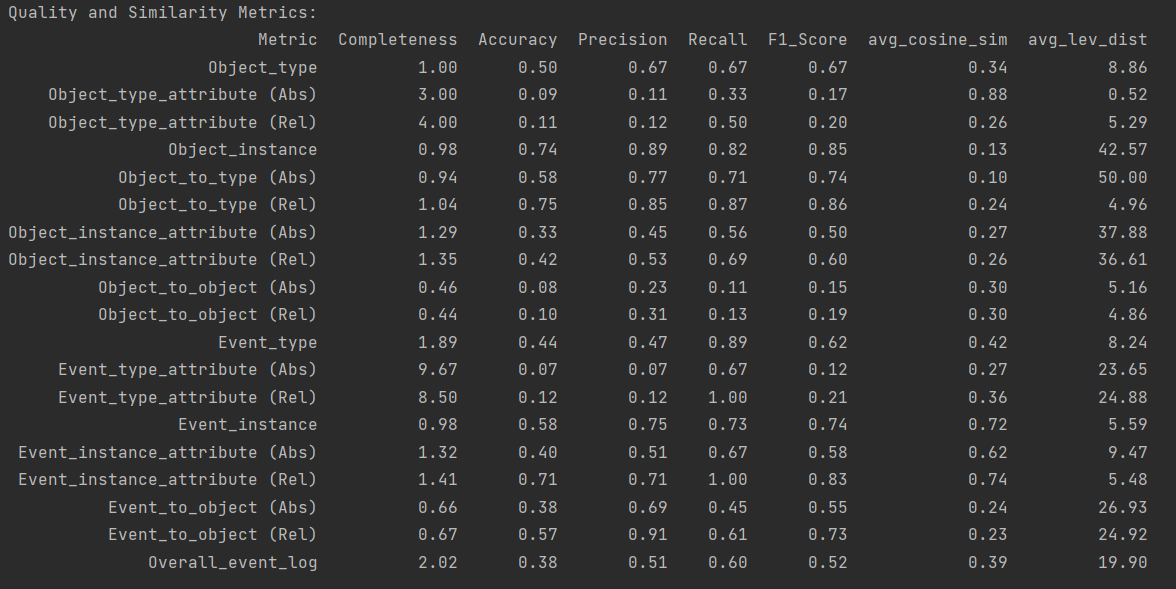
In this case, we can see that the results were terrible. This is probably due to the fact that we switched from gpt3.5 to gpt4o mini and therefore the way the descriptions were phrased changed. Therefore, a lot of adaptions were necessary to the collector pipeline that not only focused on the extraction of additional event attributes. However, after we found the problems (in this case mainly the missing extraction of object labels), we adapted the pipeline accordingly and were able to receive the following results:



These results are now again satisfactory. Therefore, we decided to continue to a level closing evaluation.

### **Level closing evaluation for complexity level 1** - Application of the first complexity level on the Order-management-process

At the end of each level, we want to conduct a final closing evaluation with a dataset that has never been used before to assess the generalizability of our pipeline. Therefore, we use a validation dataset of the order-management process that initially consists of 100 events. For this validation dataset, we receive the following results:



From this overview it becomes clear that the HEU-HEU extraction instance performs worse on the order-management-process than it does on the other datasets. This can be attributed to the fact that the order-management-process has never been analyzed in detail. We perceive an overall event log F1-Score of over 50% nevertheless acceptable. However, we can also see here that we should focus a bit more on the categories attribute types and object-to-object-relationships in the next design iterations.

## **Design iteration 2** - Second Complexity level (Textual descriptions for grouped, but intersection-free events)

While we implemented in design iteration 1 the extraction of all attributes of an object-centric event log, we increase in design iteration 2 now the complexity of the textual description. Previously, we only had one event per textual description. However, this changes now as we allow for multiple events per description. This change will also require several changes in the HEU-HEU extraction pipeline. We will again conduct three design iterations with the tree training logs of the previous complexity level – namely the recruitment process, the logistics process, and the P2P-process, and will close with a level closing evaluation with the order-management log that has not been used in the training itself.

### **Design iteration 2.1** - Application of the second complexity level on the recruiting process

#### Status of the development pipeline at the end of design iteration 2.1

The main adaptions of complexity level 2 – the acceptance of multiple events per textual descriptions – must take place in design iteration 2.1. Therefore, we adapted the generator and extractor instance accordingly.

##### Status of the generator instance

The goal of the generator instance is now to summarize more than one event per textual description. Due to the size constraints that we experienced in earlier iterations, we decided to not directly pass multiple event logs to the generator instance. Instead, we are still using the original generator pipeline to first create per event one textual descriptions. These textual descriptions are then grouped on a daily basis and are passed over to a second generator step that is asked to summarize the events that happened on that day based on the provided event-level descriptions. In this way, we were able to receive daily reports of all events happening on the corresponding day. For the generation, the following prompts have been used:

Prompts within the event-level generator step:

instructions = """You are a process mining expert. Use your knowledge base to summarize the provided OCEL2.0-logs in natural language. The report should describe all activities in the event log as well as all objects that were involved in the corresponding activity. Please make it sound as natural and human-like as possible by writing everything in one text without using bullet points. Ensure that you summarize all activities mentioned in the OCEL2.0-log properly,  
but don't come up with any new activities. Integrate all timestamps, object types and object labels/IDs, relationships and attributes in an intuitive way into the text. Don't add the event ID. If an object corresponds to a person, always mention the person togehter with its object type.  
Use only standard utf-8 characters. Don't mention the events, objects, types, relationships, and attributes specifically, but rather integrate them implicitly into the text. Never put events into quotation marks or mention the word 'event' itself.  
Make the description as concise and as short as possible and leave out unnecessary descriptions that are not specified within the log.   
Write everything in an objective tone where you simply describe what happened. Don't refer to the uploaded log, but rather describe it in a natural way without leaving any details out. Don't put any events, attributes or types into parentheses.   
It is very important that the generation is lossless, and all object IDs, types, attributes and relationships are included.  
  
Here are some examples on how your answers could look like:  
'The manager named XY opened a vacancy for the position of Manager on May 20, 2019 at 12:26:57 UTC, with the ID Vacancy[550001] - Manager.'  
'The applicant XY submitted an application on May 20, 2019 at 12:26:57 UTC, with the ID Application[550001].'  
'A vehicle with ID "vh5" was booked for a transport document with ID "td3" on May 26, 2023, at 09:53:25 UTC. The transport document with ID "td3" had 3.0 containers and was in transit. However, on June 13, 2023, at 09:00:00 UTC, its status changed to "shipped". The vehicle with ID "vh5" had a departure date of June 13, 2023, at 11:00:00 UTC.'  
"""  
  
summary\_request = """Please create based on the OCEL2.0-logs in your knowledge base textual descriptions of the event logs. Your answer should consist only of the textual description without mentioning the OCEL-log or its objects and events it is based on. The descriptions should be as short as possible. Nevertheless, be very exact with all details provided in the log and integrate all timestamps, events, object types, object labels/IDs, attributes, and relationships in an intuitive way into the text.   
If you mention a person that corresponds to an object, always also mention the persons object type, e.g. if the person is an applicant, a manager or a recruiter.  
Don't refer specifically to the uploaded log, but rather describe it in a natural way without leaving any details out. Never put events into quotation marks or mention the word 'event' itself. It is very important that the generation is lossless, and all timestamps, object types and IDs are included."""

Prompts within the grouped-daily-report generator step:

instructions = """You are a process mining expert. You will receive a couple of textual descriptions from the user with each describing an event that happened on the same date.   
Please combine those textual descriptions for the same date within one big daily report. The daily report should sound as natural as possible, but make sure that you don't forget any information (Timestamps, IDs, and object types are very important).   
Furthermore, make sure that it stays clear which objects were involved in which specific event and return ONLY the summarized daily report without any other information. Don't use bulletpoints.  
"""  
  
# Initialize the summary\_request string  
summary\_request = """In the following I give you a couple of event descriptions that all happened on the same date. Please summarize those descriptions as naturally as possible in a daily report.  
However, make sure that you don't forget any information like timestamps, object types or IDs on the way. Please also mention the object type 'applicant' for persons that submit an application to a vacancy.   
  
Here are the single-event textual descriptions."""

##### Status of the extractor instance

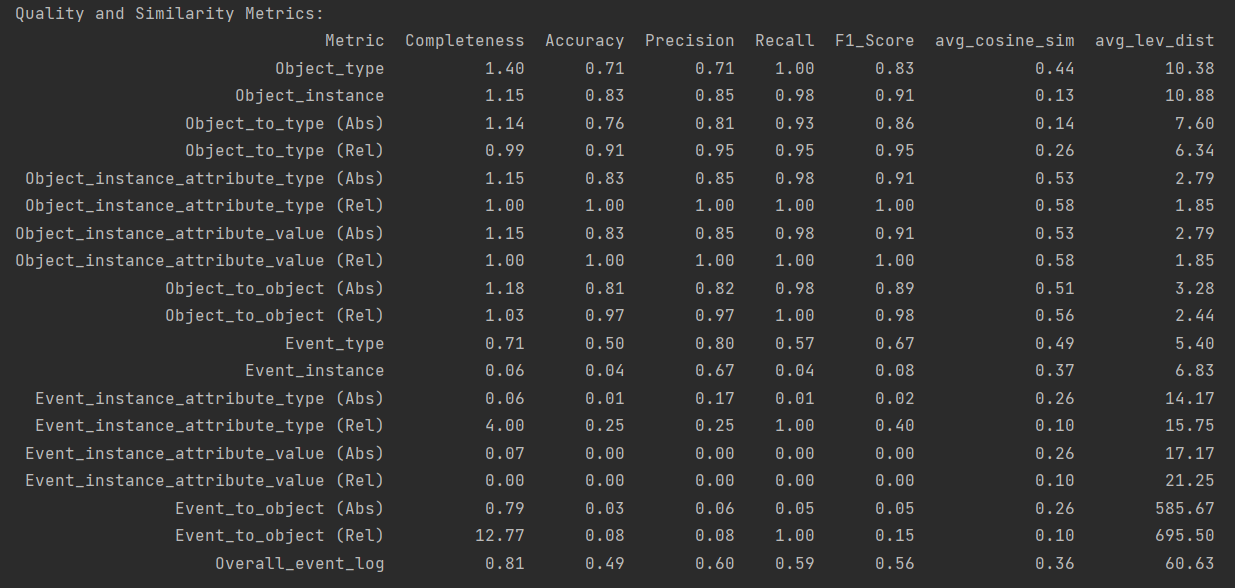
Within this design iteration, we mainly adapted the heuristic collector subcomponent of the HEU-HEU extractor instance to enable the extraction of multiple events per textual description. Therefore, we firstly allowed that multiple events could be mapped to multiple timestamps, while we insisted beforehand that only one timestamp-event-combination was retrieved. However, as now multiple events could be retrieved, we also had to integrate a proper event-to-object-relationship mapping and a proper event-to-event-attribute-mapping. Besides that, we improved the timestamp extraction component. With these adaptions we were able to receive satisfactory results on the recruitment log basis. That’s why we decided to postpone further improvement steps and adaptions of the refiner subcomponent to later design iterations.

#### Status of the comparison instance

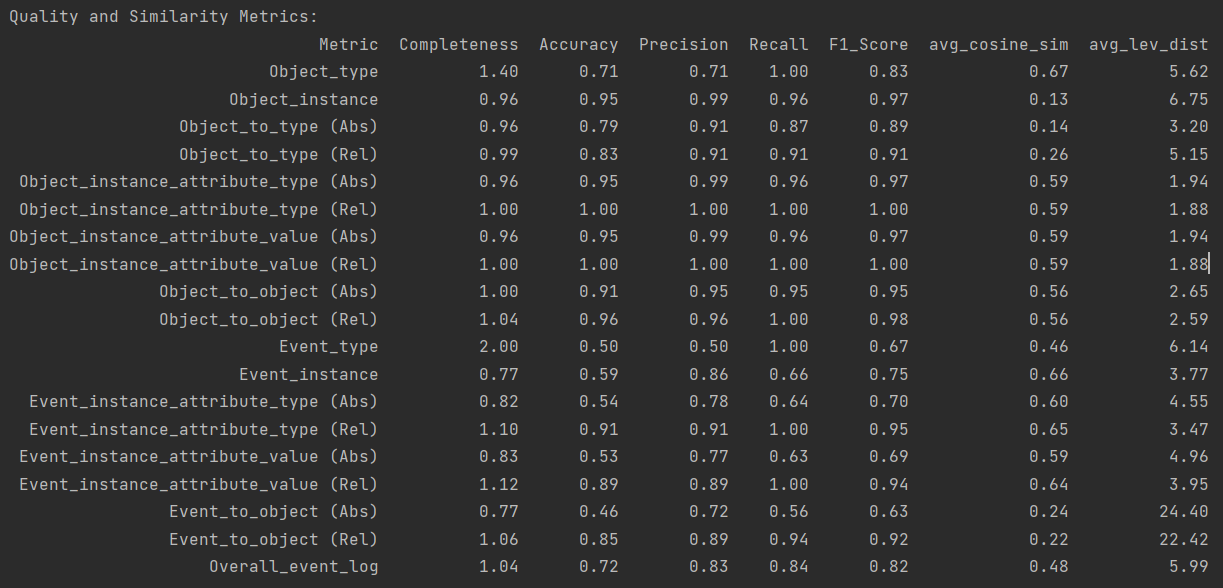
Even though no adaptions of the comparison instance were necessary in this design iteration, we decided to adapt it as we realized that the current status resulted in misleading results. Specifically, we decided to move the child levels that analyzed the object and event attribute type mapping from the parent levels object types and event types to the parent levels object instance and event instance. This adaption is due to the fact the child levels under the type-parent-level exhibit unproportional sensitivity to False-Positive-elements while often neglecting False-Negative-elements. To ensure a proper evaluation, we therefore removed these child levels from the types-parent-levels and re-added the to the instance-parent-levels.

#### Results on the validation log at the end of design iteration 2.1

At the beginning of design iteration 2.1, we were able to retrieve the following results on the recruitment validation log.



Here it becomes obvious that the extraction of most categories works properly, however, the extraction of the parent level ‘event instances’ as well as all connected child level suffers a lot. This is due to the fact that we now only provided six daily reports to the extractor that describe a total of 100 events. However, this status of the HEU-HEU extractor is only able to retrieve on event instance per description. After we enabled the extraction of multiple events per description and included the proper mapping of objects to events and event attributes to events, we were able to retrieve the following results:



With every F1-Score-category being higher than 0.5, we deemed this design iteration as successful and decided to proceed to the next iteration.

### **Design iteration 2.2** - Application of the second complexity level on the logistics process

From this design iteration on, no more major changes must be made. Instead, the design iterations now are more about improving the extraction, mapping, and refinement steps of the different components of the HEU-HEU extractor.

#### Status of the development pipeline at the end of design iteration 2.2

##### Status of the generator instance

Within the generator instance two small adaptions took place. Firstly, as it happened that all 100 events took place on the same day, something the OpenAI-API cannot handle properly, we separated the grouped, disjunct daily reports into chunks with one chunk describing a maximum of five events. Furthermore, we specified the instruction prompt for the grouped report creation to avoid ambiguity with respect to timestamp and dates. The new instruction prompt is:

instructions = """You are a process mining expert. You will receive a couple of textual descriptions from the user with each describing an event that happened on the same date.   
 Please combine those textual descriptions for the same date within one big daily report. The daily report should sound as natural as possible, but make sure that you don't forget any information (Timestamps, IDs, and object types are very important).   
 Furthermore, make sure that it stays clear which objects were involved in which specific event and return ONLY the summarized daily report without any other information.   
 If you mention other dates as well, make sure that you re-mention the original date again so that no ambiguity between timestamps and dates can occur. Don't use bulletpoints.  
 """

##### Status of the extractor instance

One major change that took place in this design iteration in regard to the HEU-HEU extractor instance was the extraction of object labels and object values as part of the collector subcomponent. As per definition an object label is only another attribute value that is later selected as a key, we decided to firstly extract candidate values for object labels and attribute values together. Afterwards, we decide if a value is clearly an ID (e.g. if it is a combination of letters and numbers) or if it is an attribute value (e.g. if it is a cardinal or an adjective). The remaining candidate values are later tested for their attributability to an event. If a value is a Noun or Proper Noun and can be directly attributed to an event it will be treated as an object label. Otherwise, it will be treated as an attribute value. Furthermore, we included the exception case if a timestamp is an attribute value for example for the case ‘departure date’. This exception case is triggered if a clear timestamp indicator exists in the extracted attributes (e.g. like the word 'date’). If that’s the case, we try to match the corresponding attribute to a timestamp and will forth on treat the timestamp as an attribute value. Besides that, improvements on all other parts of the collector and refiner took place with a special focus on the proper extraction of events consisting of timestamps and event types.

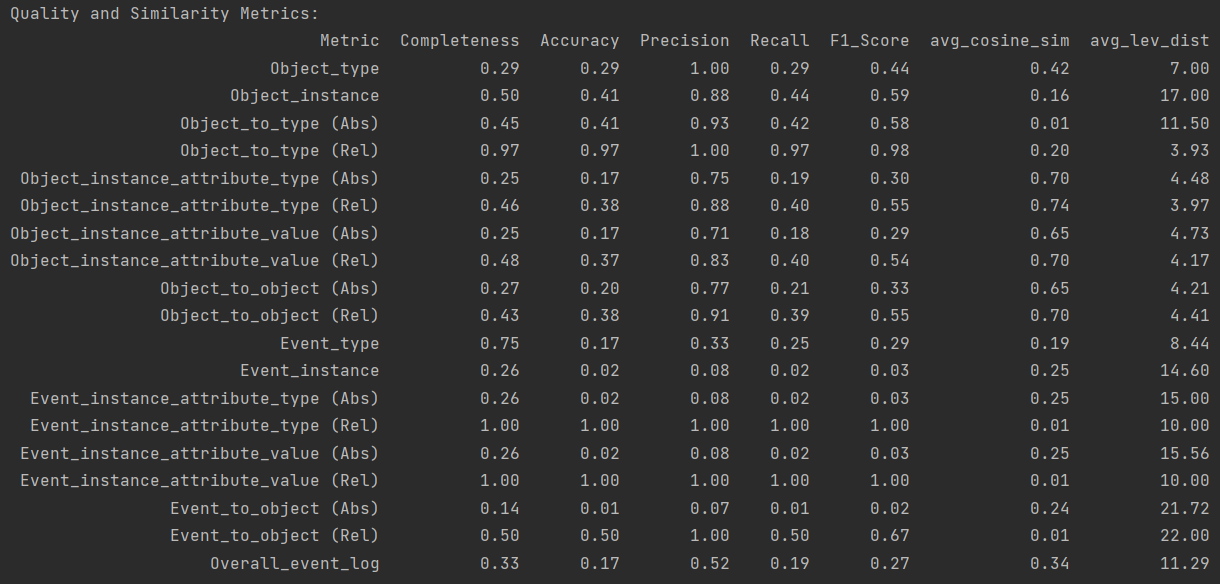
##### Status of the comparison instance

The only change that took place in the comparison instance was a small refinement on the way how matchings between entities in the old and new log are detected.

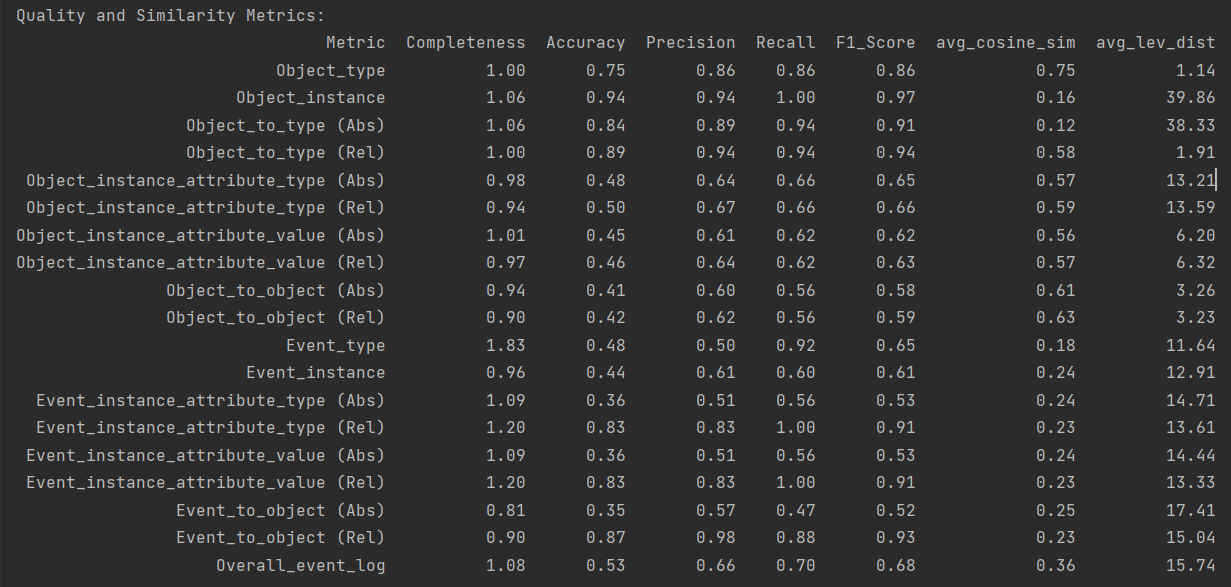
#### Results on the validation log at the end of design iteration 2.2

In the following, the results for the validation log before and after the adaptions in design iteration 2.2 can be analyzed.

Pre-adaption results:



Post-adaption results:



It becomes clear that previous issues could be resolved, paving the way for design iteration 2.3.

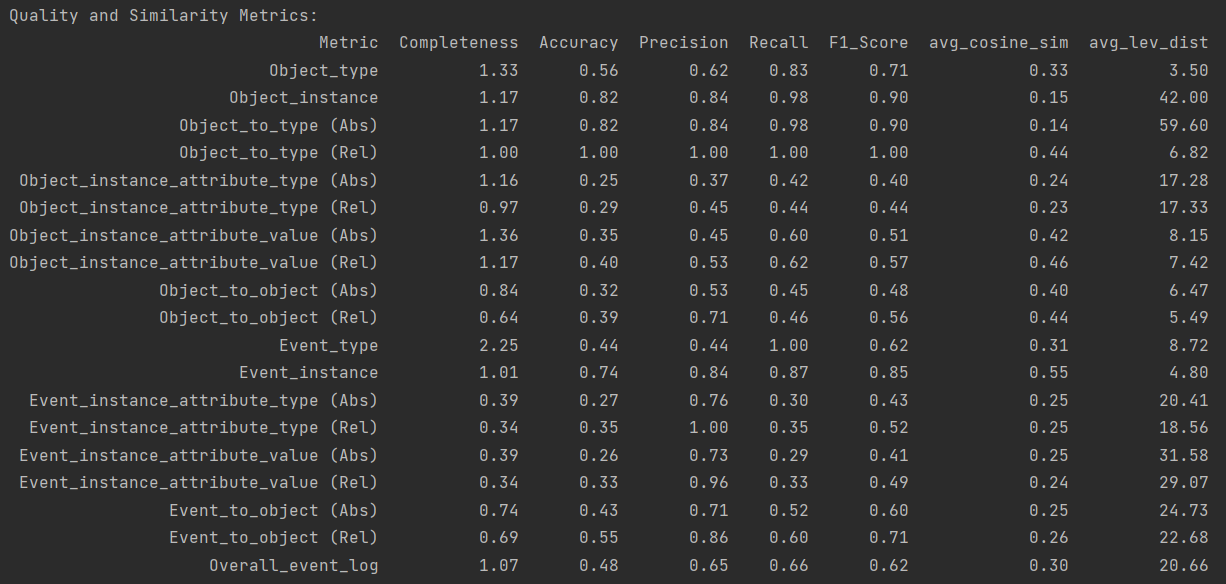
### **Design iteration 2.3** - Application of the second complexity level on the P2P process

#### Status of the development pipeline at the end of design iteration 2.3

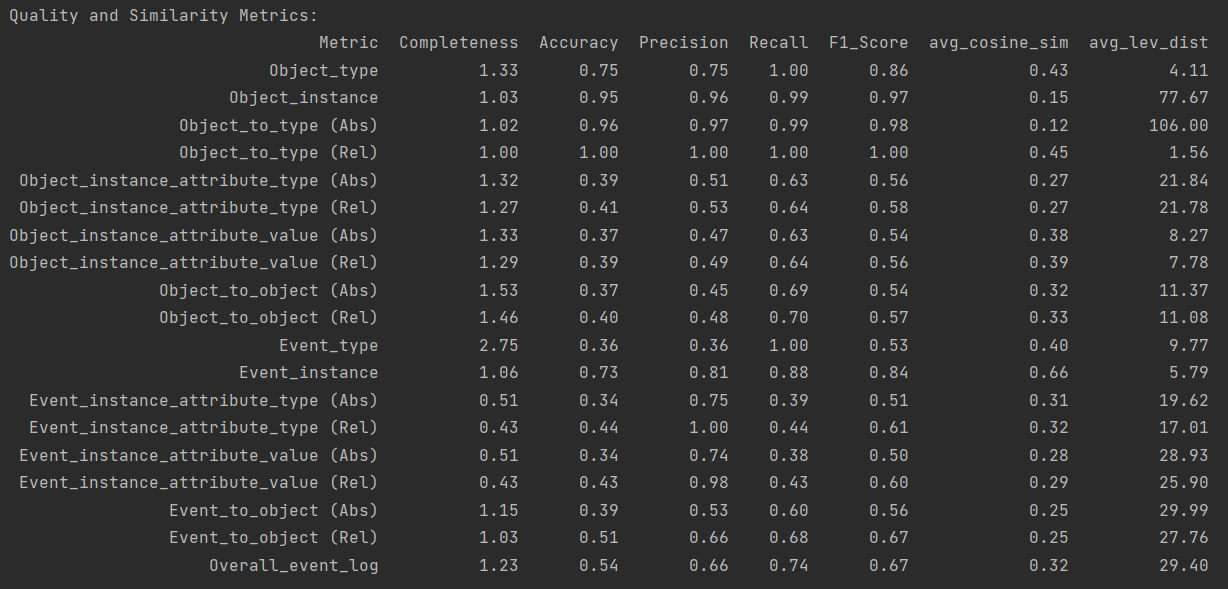
As the results at the beginning of design iteration 2.3 where already quite promising only minor refinements were necessary at the HEU-HEU extractor instance. Thereby, the focus lied mainly on the improvement of attribute extractions and object-to-object-relationship extractions.

#### Results on the validation log at the end of design iteration 2.3

As mentioned before the results at the beginning of design iteration 2.3 were already pretty good with some smaller issues in the categories attribute extraction and object-to-object-relationship extraction. The corresponding results can be found in the following image:



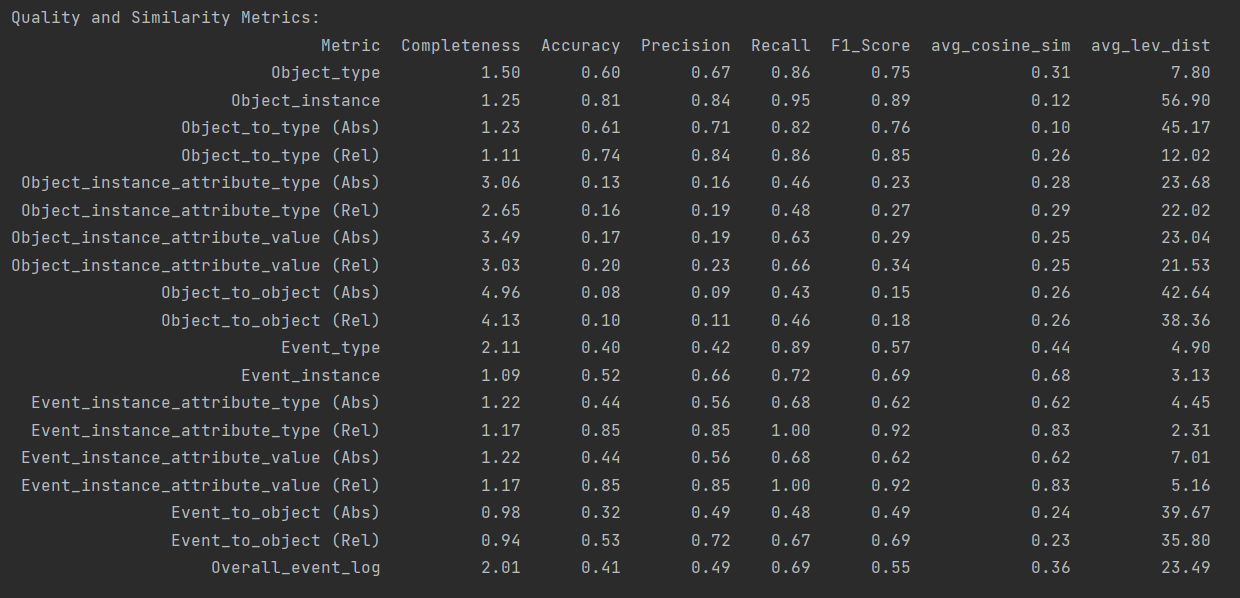
After some minor improvements were able to receive the following results on the validation log:



Satisfied by these results, we continued to the level closing evaluation.

### **Level closing evaluation for complexity level 2** - Application of the second complexity level on the Order-management-process

Within the level closing evaluation, we again applied the current status of our pipeline on the unused order-management process. Thereby we were able to produce the following results.



Even though the results on the order-management process still doesn’t reach the quality of the other event logs, we are satisfied with an overall F1-Score of over 0.5. As a result, we switch to the next higher complexity level.

## **Design iteration 3** - Third Complexity level (Textual descriptions for grouped and intersecting event sets)

Within the third complexity level, we still create textual descriptions that describe more than one event. However, while we previously only received on textual descriptions per event, we now create multiple descriptions per event from different perspectives.

### **Design iteration 3.1** - Application of the third complexity level on the recruiting process

#### Status of the development pipeline at the end of design iteration 3.1

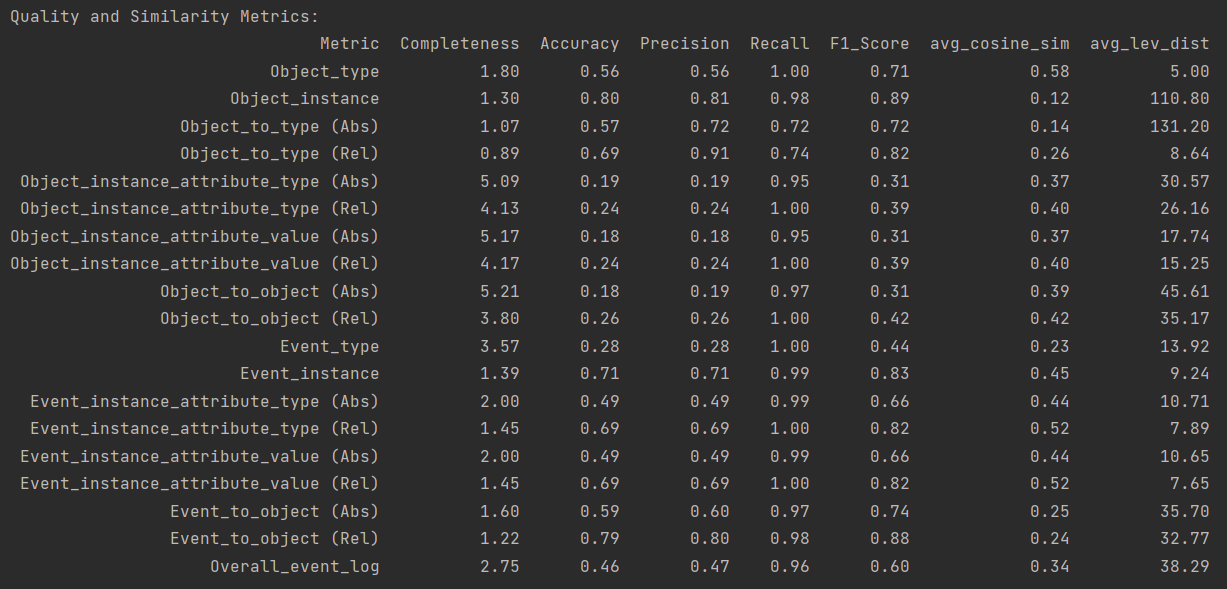
Within design iteration 3.1 we mainly adapted the generator instance to the new use case. Thereby, we now create textual descriptions for events from different perspectives by grouping the events to the related objects. Therefore, instead of receiving daily reports, we now receive reports that describe specifically on object and all events this object is related to from its perspective. For the generation of those textual reports, we used the following prompts:

instructions = """You are a process mining expert. You will receive a couple of textual descriptions from the user, each describing an event related to the same object.   
 Please combine those textual descriptions for the same object within one report. The report should sound as natural as possible, but make sure that you don't forget any information   
 (Timestamps, IDs, and object types are very important). Mention relationships between objects clearly, and ensure that it is clear which objects were involved in which specific event.   
 Return ONLY the summarized report without any additional information."""  
  
summary\_request = f"""In the following, I give you a couple of event descriptions that are all related to the object {object\_id}. Please summarize those descriptions as naturally as possible. sure that you don't forget any information like timestamps, object types, or IDs.  
Here are the single-event textual descriptions."""

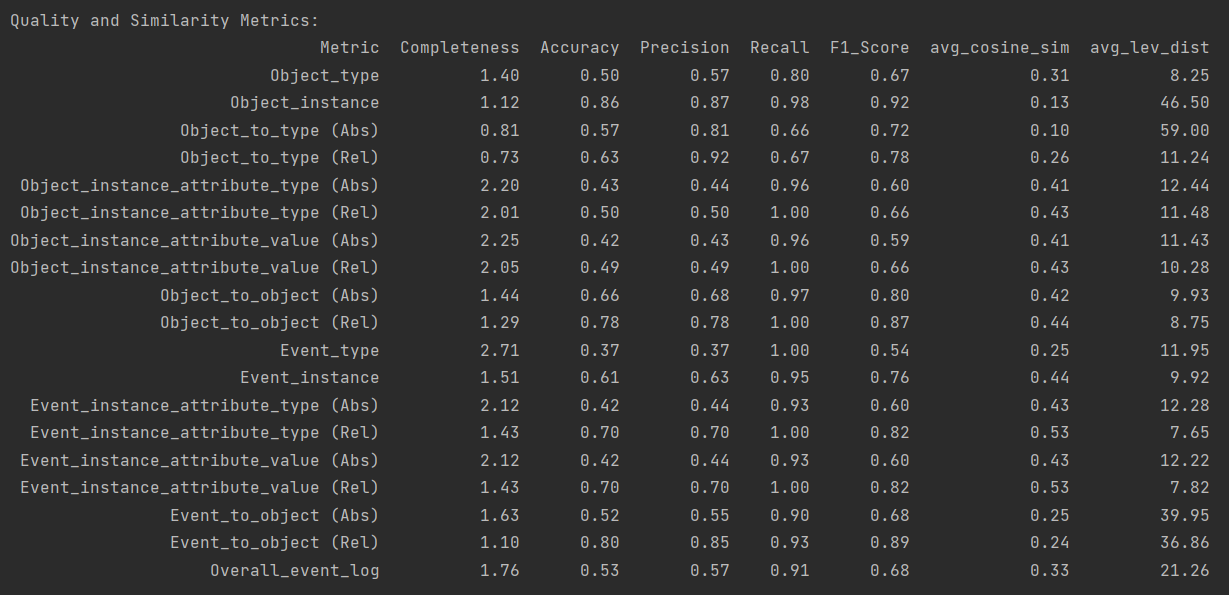
After some minor adjustments in the extractor instance, we were then able to receive satisfactory results again.

#### Results on the validation log at the beginning and end of design iteration 3.1

Pre-adaption results:



Post-adaption results:



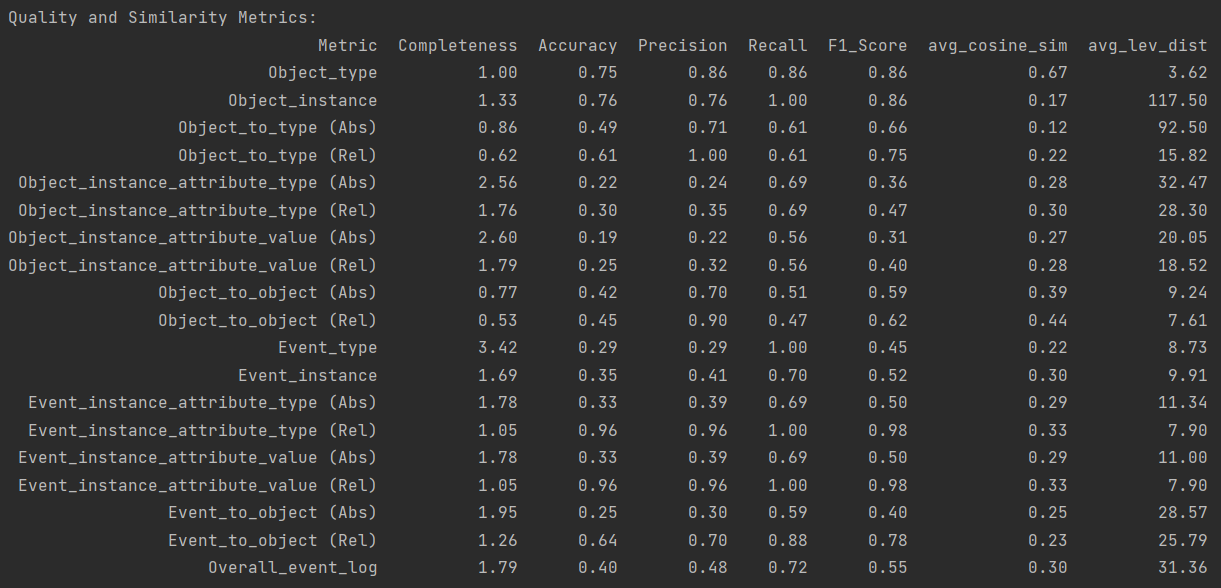
### **Design iteration 3.2** - Application of the third complexity level on the logistics process

#### Status of the development pipeline at the end of design iteration 3.2

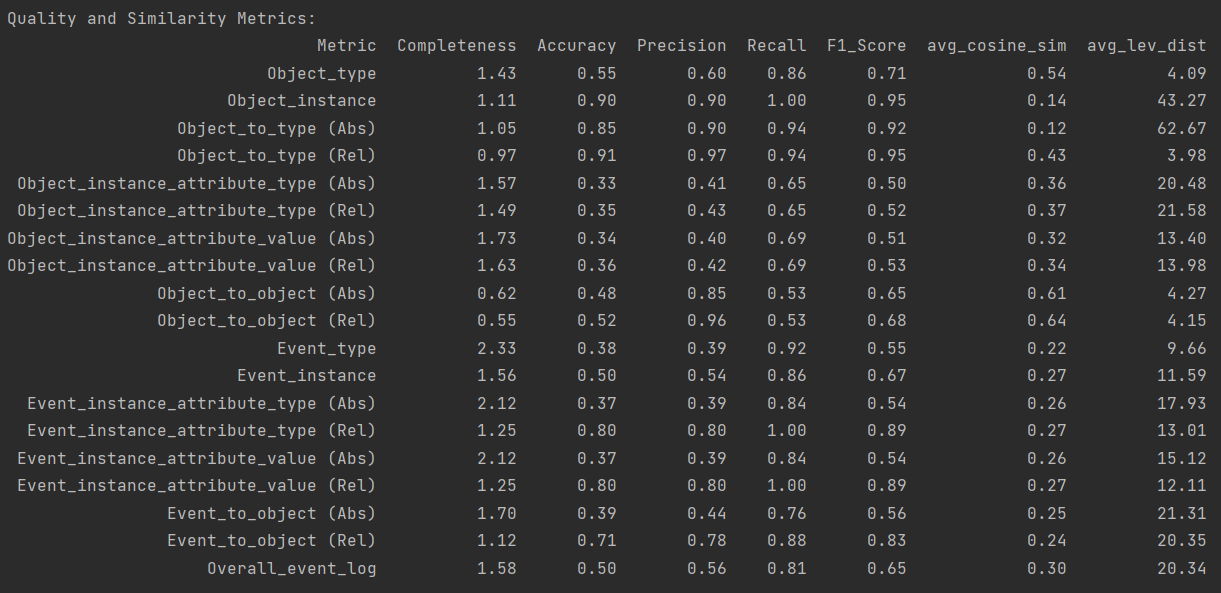
Within design iteration 3.2 only minor adjustments at the HEU-HEU extractor instance took place to ensure satisfactory results before continuing with design iteration 3.3.

#### Results on the validation log at the beginning and end of design iteration 3.2

Pre-adaption results:



Post-adaption results:



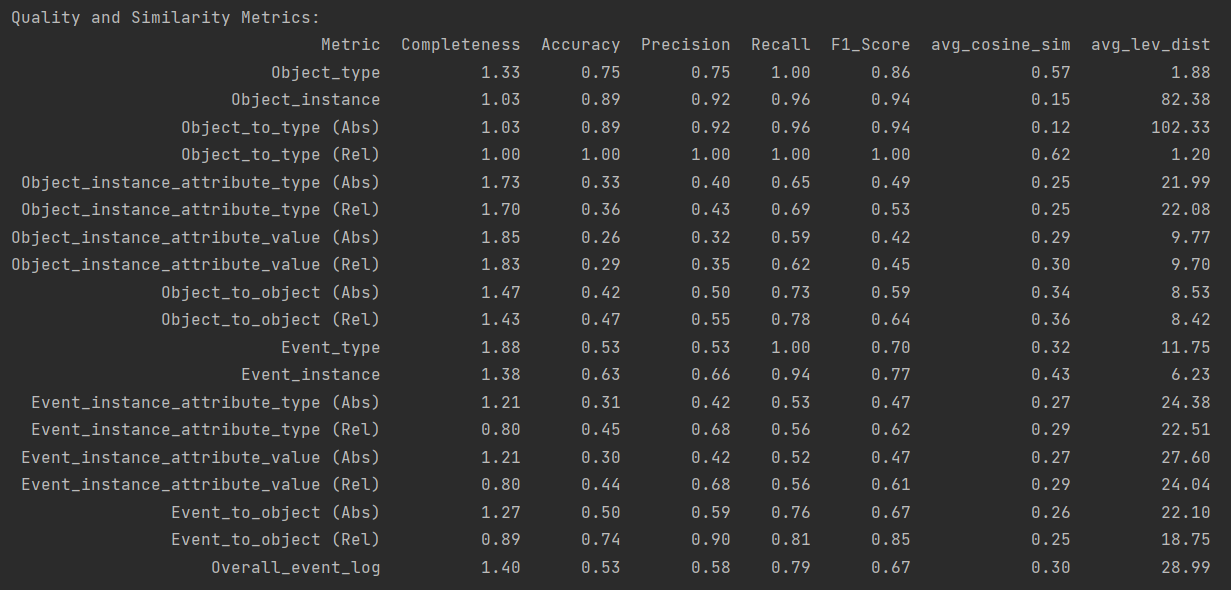
### **Design iteration 3.3** - Application of the third complexity level on the P2P process

#### Status of the development pipeline at the end of design iteration 3.3

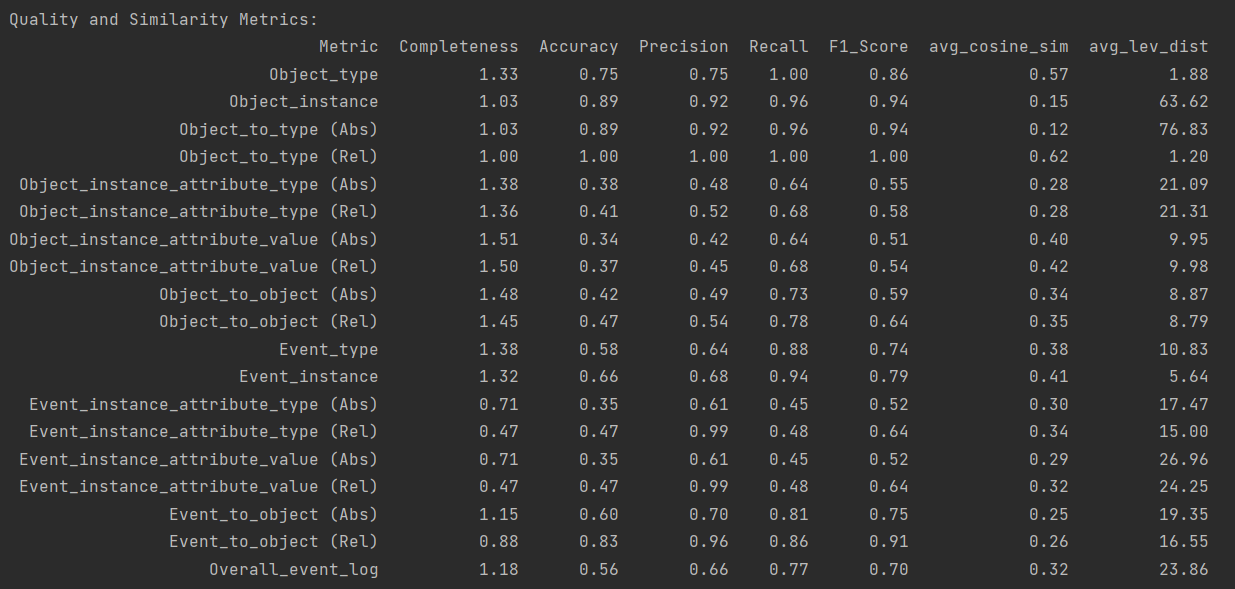
Only minor adjustments necessary.

#### Results on the validation log at the beginning and end of design iteration 3.3

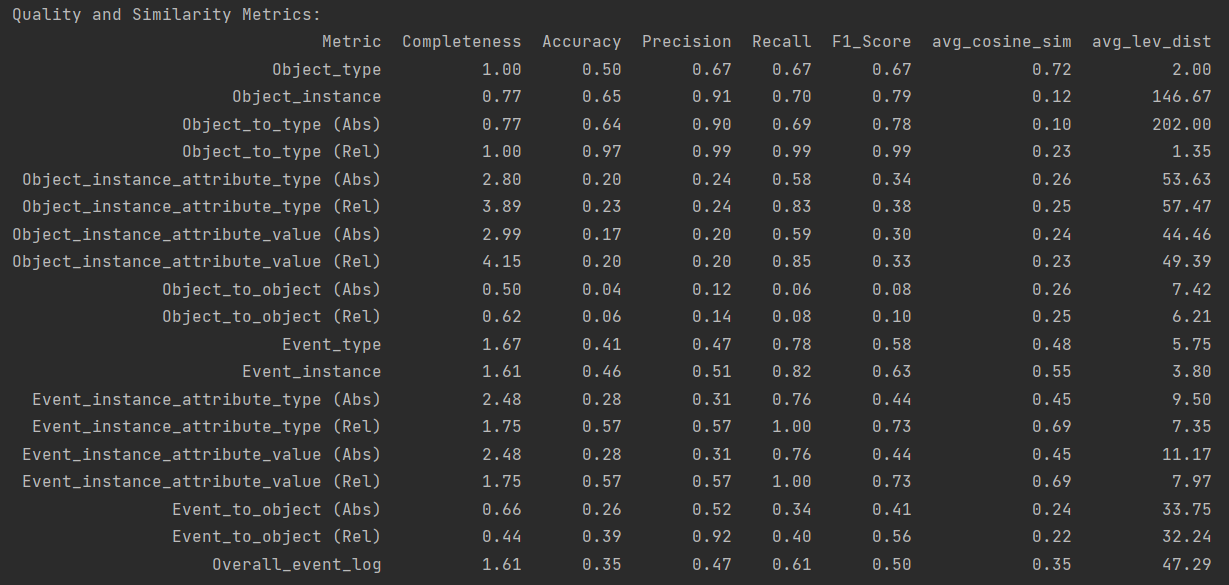
Pre-adaption results:



Post-adaption results:



### **Level closing evaluation for complexity level 3** - Application of the third complexity level on the Order-management-process



# **Development of the generative GEN-GEN extractor instance**

After we finalized the development of the heuristic HEU-HEU extractor instance, we developed a straightforward generative GEN-GEN counterpart extractor instance. It consists of a generative collector subcomponent and a generative refiner subcomponent.

Within the generative collector subcomponent, we firstly set up an LLM with file-search capabilities. As LLM we currently use the OpenAI gpt-4o-mini-API, however the specific model can be exchanged easily. Within the setup, we used the following system prompt that should guide the correct extraction of event logs from textual descriptions:

instructions = """You are a process mining expert. You will now receive a couple of textual descriptions. Please extract event logs in the OCEL2.0 json format from these textual desciptions.  
 An example on how the OCEL2.0 format looks like is in your knowledge base. However, here is an overview of the general structure:  
  
 {  
 "objectTypes": [  
 {  
 "name": "",  
 "attributes": [  
 {  
 "name": "",  
 "type": ""  
 }  
 ]  
 }  
 ],  
 "eventTypes": [  
 {  
 "name": "",  
 "attributes": [  
 {  
 "name": "",  
 "type": ""  
 },  
 {  
 "name": "",  
 "type": ""  
 }  
 ]  
 }  
 ],  
 "objects": [  
 {  
 "id": "",  
 "type": "",  
 "attributes": [  
 {  
 "name": "",  
 "time": "",  
 "value": ""  
 }  
 ],  
 "relationships": [  
 {  
 "objectId": "",  
 "qualifier": ""  
 }  
 ]  
 }  
 ],  
 "events": [  
 {  
 "id": ,  
 "type": "",  
 "time": "YYYY-MM-DDTHH:MM:SSZ",  
 "attributes": [  
 {  
 "name": "",  
 "value": ""  
 },  
 {  
 "name": "",  
 "value": ""  
 }  
 ],  
 "relationships": [  
 {  
 "objectId": "",  
 "qualifier": ""  
 }  
 ]  
 }  
 ]  
 }  
  
 You must extract objects, object types, event types, timestamps and event-to-object relationship to create a minimal OCEL2.0 log.  
 For objects, use names and IDs that you found in the text as object IDs and don't come up with own object IDs.  
 If possible and necessary, please also extract object and event attributes as well as object-to-object relationships. However, these categories are optional.  
 Return ONLY the extracted event log in OCEL2.0 format without any other text or information.  
 """

To ensure the extraction into the proper OCEL2.0 format, we firstly provide the broad structure in the system prompt and secondly provide a one-event example of an OCEL2.0 log in the vector store, the LLM can access via its file search capabilities.

After we initially setup our extractor LLM, we provide the textual descriptions iteratively to it and ask it to extract event logs from these textual descriptions. Therefore, we use the following user prompt:

extraction\_request = """You will now receive a texutal description. Please extract event logs in the OCEL2.0 format from this textual description and return ONLY the log in OCEL2.0-json-format.  
 An example of an log in OCEL2.0 json format is provided in your knowledge base."""  
  
extraction\_request += f"\n\nHere is the textual description: {text}\n"

Lastly, the generated preliminary OCEL snippet is stored in an intermediate folder. In this way, we can again create one preliminary OCEL snippet per textual description the same way as we do it with the heuristic collector.

In the next step, the different preliminary OCEL snippets are concatenated to a concatenated OCEL that is then further provided to the generative refiner subcomponent. Within the refiner subcomponent a second LLM with file-search-capabilities is setup leveraging the OpenAI gpt-4-o-mini-API as well. The following system prompt is then used to guide the refinement process:

instructions = """You are a process mining expert. You will now receive an concatenated event log in ocel2.0 format over your knowledge base.  
 Please refine this concatenated ocel2.0 log by for example cleaning the different names, merging similar entities and ensuring a correct correspondence between the different parts of the log.  
 Don't merge any events that happened at different timestamps.  
 Return ONLY the refined event log in OCEL2.0 format without any other text or information.  
 """

To start the refinement, we load the concatenated log into the vector storage of the LLM and ask it via a user prompt to refine it:

refinement\_request = """Please refine the log in your knowledge base. Return ONLY the refined event log in OCEL2.0 format without any other text or information."""

The output of the refiner-LLM will then correspond to our final OCEL2.0 log. As no “training” in the sense of fine-tuning was conducted, we only setup the generative extractor instance once without iterating over different design iteration. The generative extractor will be only evaluated once at the end against the test datasets of the different event logs.

# **Development of the hybrid extractor instances**

Lastly, we also developed two hybrid variants of the heuristic and generative extractors. The first combined variant, the GEN-HEU extractor, utilizes the collector subcomponent from the generative extractor and the refiner subcomponent from the heuristic extractor. Conversely, the second combined variant, the HEU-GEN extractor, employs the collector subcomponent of the heuristic extractor paired with the refiner subcomponent from the generative extractor. As these hybrid extractors are constructed by merging components from the existing heuristic and generative extractor models, no additional design iterations were performed.

# **Final evaluation**

In the final evaluation, all four extractor variants are evaluated and compared against each other within an artificial setting. Therefore, we use the development framework used during the development of the HEU-HEU extractor. Specifically, we create testing subsets (size: 1000 events each) for six event logs. Out of these six event logs, three event logs – recruiting log, logistics log, and P2P log - have been used during the development of the HEU-HEU extractor. The other three event logs – the order management log and an additional production log (<https://zenodo.org/records/13638681> ) and age of empires log (<https://zenodo.org/records/13365584>) – haven’t been used during training and therefore represent an opportunity for analyzing the generalization capabilities of the different extractor variants. The testing subsets of the six event logs are divided into three equal parts. Leveraging the generator instance of the development framework, textual descriptions corresponding to the three complexity level are then generated based on these testing subsets. Then, the four extractor variants take on the role of the extractor instance in the development framework and try to re-engineer the original logs. The comparison instance then compares the extracted logs with the original logs and outputs detailed results for each log-extractor combination. The detailed results as well as the consolidated results can be analyzed in the following chapter.

## **Detailed Results**

This chapter presents the detailed results for all extractor-event log-combinations.

### Recruitment log

HEU-HEU extractor:

Ein Bild, das Text, Karte Menü, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

GEN-GEN extractor:

Ein Bild, das Text, Zahl, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

GEN-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

HEU-GEN extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

### Logistics log

HEU-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

GEN-GEN extractor:

Ein Bild, das Text, Schrift, Zahl, Dokument enthält.

Automatisch generierte Beschreibung

GEN-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

HEU-GEN extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

### P2P log

HEU-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

GEN-GEN extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

GEN-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

HEU-GEN extractor:

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

### Order-Management log

HEU-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

GEN-GEN extractor:

Ein Bild, das Text, Zahl, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

GEN-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

HEU-GEN extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

### Production log

HEU-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

GEN-GEN extractor:

Ein Bild, das Text, Schrift, Zahl, Dokument enthält.

Automatisch generierte Beschreibung

GEN-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

HEU-GEN extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

### Age of empires log

HEU-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

GEN-GEN extractor:

Ein Bild, das Text, Dokument, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

GEN-HEU extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

HEU-GEN extractor:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

## **Consolidated results**

Recruitment log:



Logistics log:



P2P log:



Order management log:



Production log:



Age of empires log:



Overall:



The overall table presents the averaged precision, recall, and F1-score across all extractor variants and event logs. Analyzing these results shows that all extractor variants tend to achieve higher recall values (ranging from 52 to 70\%) compared to precision values (32 to 57\%), resulting in harmonized F1-scores between 38 and 58\% across the entire event log. This disparity is particularly noticeable in the GEN-GEN and HEU-GEN variants, while it is less pronounced in the HEU-HEU and GEN-HEU variants. The gap is also especially evident in the ‘object types' and ‘event types’ categories, where high recall values are counterbalanced by lower precision, resulting in moderate F1-scores. Only in the ‘object labels’ and ‘object labels to object type matching’ categories, all extractor variants perform equally well in both recall and precision, producing high F1-scores. In contrast, extracting and correctly mapping ‘event attributes’ proves to be particularly challenging for all extractor variants.

Overall, the GEN-HEU extractor consistently outperforms the other variants, particularly in terms of precision and F1-score, with the HEU-HEU extractor following. On the other hand, while the GEN-GEN and HEU-GEN variants fall behind in precision and F1-score, they remain competitive, even slightly outperforming the others in recall. A detailed comparison of the overall F1-scores across all event logs is presented in Figure 6.

Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Figure 6: Overall F1-Scores for all extractor variants

In line with the previous results, the GEN-GEN and HEU-GEN extractors generally underperform compared to the HEU-HEU and GEN-HEU extractors across most event logs. The HEU-HEU extractor performs particularly well on the event logs used during the development of its heuristic subcomponents. However, when analyzing unseen logs, it is surpassed by the GEN-HEU extractor, which demonstrates consistently high performance.