Università
della
Svizzera
italiana

**Software
Institute**

CODI

A Conversation Disentanglement Microservice

Edoardo Riggio

June 2022

Supervised by
Prof. Dr. Michele Lanza

Co-Supervised by
Marco Raglianti

BACHELOR PROJECT

Abstract

Instant Messaging Applications (IMAs) – such as Discord, Gitter, and Slack – are becoming popular means of communication. However, IMAs lack features to disambiguate and identify conversations. A conversation is a sequence of time-stamped messages representing an interactive discussion between multiple people. A message is said to be part of a conversation if it answers a previously posed question or discusses a similar topic as other messages. We implemented an accessible and user-friendly REST microservice that can automate the disambiguation of a set of messages to form conversations by leveraging machine learning algorithms.

Contents

Chapter 1

Introduction

Instant Messaging Applications have evolved into one of the most revolutionary means of communication, gradually supplanting conventional asynchronous media (e.g., *email*). People thousands of kilometers apart can communicate in real-time thanks to these services. Thus, some instant messaging apps, such as *Slack* and *Discord*, might have extremely high throughput. This means that hundreds of messages could be transmitted every second by people all around the world. High throughput in large chats may cause numerous conversations to overlap and be separated by interleaving messages.

Researchers have found algorithms that could mimic what human beings do to separate conversations [1–6]. Of these algorithms, we decided to implement the one proposed by Elsner and Charniak [1,3] and later extended by Chatterjee et al. [2]. This is a two-step algorithm in which pairs of messages and their features are first fed to a *max-entropy classifier* – which determines the relatedness of messages. The output of this classifier is given as input, together with the list of all the messages, to the second step, the *correlation clustering algorithm* which will group messages based on the previously found relations into conversations.

With CODI, we attempted to achieve ease of use. We aimed to make it easy for researchers to annotate and perform conversation disentanglement on given datasets. *CODI* is a simple web application with a client module that allows users to drag and drop datasets to perform *training*, *validation*, or *prediction* operations, as well as graphically pick which feature-sets to employ in the disentanglement process. Users can also directly access the REST API endpoints, which can be invoked using scripts, for example. Moreover, we have created a view for dataset annotation. This was done so that even researchers with limited technical knowledge could readily manually label the dataset that would be given to the disentangler.

1.1 Contributions

There are several contributions to this project. Below we will identify all of them.

- **Replication Contribution**
  This consisted in reproducing the algorithm developed by Elsner and Charniak [1,3]. This two-step disentangling approach, which includes both the *max-entropy* and *correlation clustering* algorithms, is used in *CODI*'s backend.

- **Engineering Contribution**
  This is the development of *CODI* itself. *CODI* can accept input files containing raw or structured dumps of Discord and Slack chats, with or without conversation annotation. The input dataset is pre-processed before being sent to the two-step disentanglement algorithm. Once the algorithm has generated the conversations, they are displayed to the user along with statistics on the disentanglement's effectiveness, such as *accuracy*, *precision*, *recall*, and *F1 score* for the max-entropy algorithm; and *micro-averaged F1 score* for the correlation clustering algorithm.

- **Dataset Contribution**
  In addition to the datasets made public by Elsner and Charniak [1,3] and by Chatterjee et. al [2], we have annotated our own dataset. This dataset comprises 294 messages taken from a Pharo Discord server. Two people manually annotated the messages, which were then merged to form the final annotated dataset. We have used this dataset to evaluate our implementation of the disentanglement algorithm on a new ground truth.

- **Validation Contribution**
  In the final phase of the implementation of *CODI*, we used some annotated datasets to verify our implementation's correctness and understand its limitations. In addition to two datasets offered by Elsner and Charniak and Chatterjee et al., we also use our custom dataset in the evaluation. The two original datasets were mainly used to verify the correctness of the implementation. The results obtained were compared with the ones obtained by the original authors. The third dataset, the custom one, was used to understand the limitations of such an algorithm.

1.2 Document Structure

This document has been divided into several different chapters. Each chapter describes one or more of the steps that allowed us to complete this project. More specifically, we have four chapters.

- **State of the Art**
  In this chapter, we talk about the research done in this field prior to our project. We have also inserted a section dedicated to the history of instant messaging and its applications.

- ***CODI* Implementation**
  This chapter explains in detail how *CODI* works. It talks about how we structured *CODI* and what a user can do to disentangle their datasets.

- **Result Analysis**
  In this chapter, we analyze the experiments we performed on *CODI* with several different datasets and report the results of such experiments.

- **Conclusion and Future Work**
  In this last chapter, we have a brief concluding summary of the project and suggestions on how to further improve *CODI*.

Chapter 2

State of the Art

2.1 Instant Messaging

Instant messaging is a set of protocols for communication between two or more people over the Internet. This technology differs from others – such as email – because conversations happen in real-time rather than asynchronously, hence the word *instant*.

2.1.1 Brief History

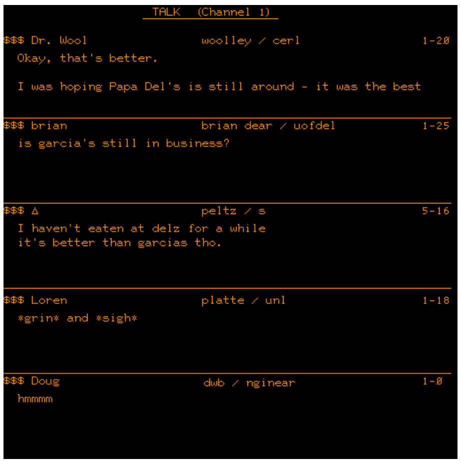
The first instant messaging platform dates back to 1973. Its name was *Talkomatic* (Figure 2.1a), and it allowed five people to have each a section of the screen – each user could write only five lines of text and chat in real-time. Differently from modern-day instant messaging applications, *Talkomatic* was a real-time text service. This means the messages were broadcasted letter-by-letter as the users typed them in. This type of technology characterized most of the early real-time messaging services.

In the early 1990s, however, a company named *Quantum Link* – which later became *America On-Line* – developed and released an online service for the Commodore 64 in which concurrently connected users could exchange messages in real-time. This concept was further refined and integrated by *America Online* in one of the first and most important mass-distributed instant messaging platforms, *AOL Instant Messenger* (AIM) [1] (Figure 2.1c). This platform was also one of the first with a *Graphic User Interface* (GUI).

Today, instant messaging has become one of the most prevalent communication mediums. Now we have platforms such as *WhatsApp*, *Discord* (Figure 2.1d), *Slack*, *Skype*, *Telegram*... These platforms have billions of monthly active users [2] [3] and have reshaped how we communicate. *Gitter*, *Slack*, and *Discord* are currently some of the most used instant messaging platforms and have also been studied as possible data mining sources for software-related information [7–13] among software developers.

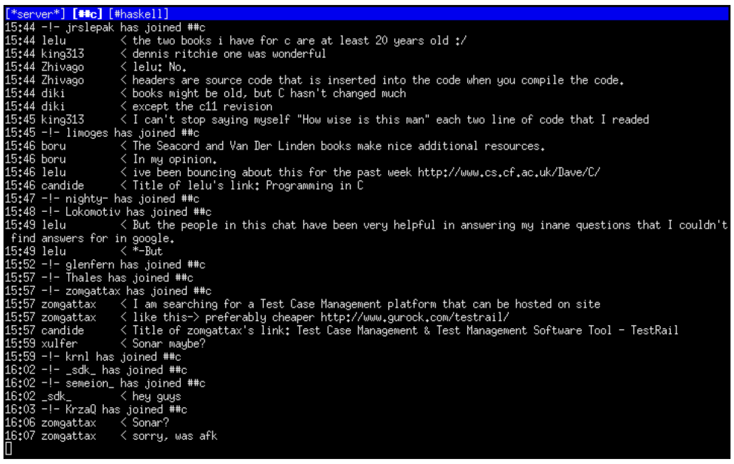
IM platforms allow users to create communities with several different channels where people can discuss specific subtopics. In many cases, such communities and channels are related to software development. These channels often follow a Q&A approach. This means that developers ask questions related to the subtopic of the channel and receive answers from other fellow developers. Users may participate in such conversations by sending messages. Code blocks, mentions to other users or channels, multimedia, or attachments can sometimes accompany such messages. These additional elements can help the disentanglement process, particularly in software development environments.

[1] https://en.wikipedia.org/wiki/Instant_messaging
[2] https://www.businessofapps.com/data/slack-statistics/
[3] https://www.businessofapps.com/data/discord-statistics/



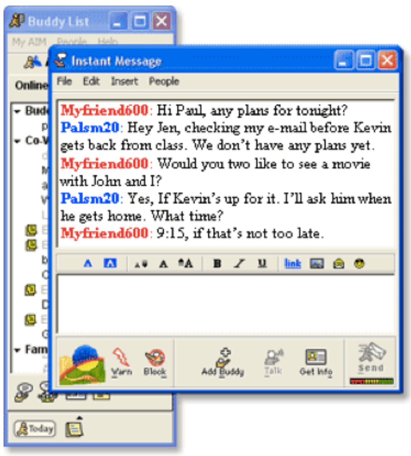
```
                    TALK   (Channel 1)

$$$ Dr. Wool                  woolley / cerl              1-20
  Okay, that's better.

  I was hoping Papa Del's is still around - it was the best

$$$ brian                     brian dear / uofdel         1-25
  is garcia's still in business?

$$$ Δ                         peltz / s                   5-16
  I haven't eaten at delz for a while
  it's better than garcias tho.

$$$ Loren                     platte / unl                1-18
  *grin* and *sigh*

$$$ Doug                      dwb / nginear               1-0
  hmmmm
```

(A) *Talkomatic* interface - 1973 [4]



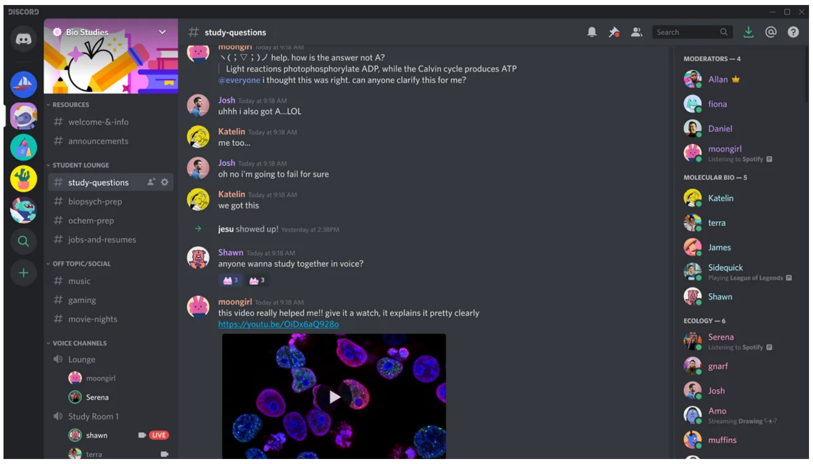
```
[*server*] [##c] [#haskell]
15:44 -!- jrslepak has joined ##c
15:44 lelu         < the two books i have for c are at least 20 years old :/
15:44 king313      < dennis ritchie one was wonderful
15:44 Zhivago      < lelu: No.
15:44 Zhivago      < headers are source code that is inserted into the code when you compile the code.
15:44 diki         < books might be old, but C hasn't changed much
15:44 diki         < except the c11 revision
15:45 king313      < I can't stop saying myself "How wise is this man" each two line of code that I readed
15:45 -!- limoges has joined ##c
15:46 boru         < The Seacord and Van Der Linden books make nice additional resources.
15:46 boru         < In my opinion.
15:46 lelu         < ive been bouncing about this for the past week http://www.cs.cf.ac.uk/Dave/C/
15:46 candide      < Title of lelu's link: Programming in C
15:47 -!- nighty- has joined ##c
15:48 -!- Lokomotiv has joined ##c
15:49 lelu         < But the people in this chat have been very helpful in answering my inane questions that I couldn't
 find answers for in google.
15:49 lelu         < *-But
15:52 -!- glenfern has joined ##c
15:57 -!- Thales has joined ##c
15:57 -!- zomgattax has joined ##c
15:57 zomgattax    < I am searching for a Test Case Management platform that can be hosted on site
15:57 zomgattax    < like this-> preferably cheaper http://www.gurock.com/testrail/
15:57 candide      < Title of zomgattax's link: Test Case Management & Test Management Software Tool - TestRail
15:59 xulfer       < Sonar maybe?
15:59 -!- krnl has joined ##c
16:02 -!- _sdk_ has joined ##c
16:02 -!- semeion_ has joined ##c
16:02 _sdk_        < hey guys
16:03 -!- KrzaQ has joined ##c
16:06 zomgattax    < Sonar?
16:07 zomgattax    < sorry, was afk
```

(B) *IRC client* interface - 1988 [5]



Myfriend600: Hi Paul, any plans for tonight?
Palsm20: Hey Jen, checking my e-mail before Kevin gets back from class. We don't have any plans yet.
Myfriend600: Would you two like to see a movie with John and I?
Palsm20: Yes, If Kevin's up for it. I'll ask him when he gets home. What time?
Myfriend600: 9:15, if that's not too late.

(C) *AIM* interface - 1997 [6]



moongirl Today at 9:18 AM
ヽ(；▽；)ノ help. how is the answer not A?
Light reactions photophosphorylate ADP, while the Calvin cycle produces ATP
@everyone i thought this was right. can anyone clarify this for me?

Josh Today at 9:18 AM
uhhh i also got A...LOL

Katelin Today at 9:18 AM
me too...

Josh Today at 9:18 AM
oh no i'm going to fail for sure

Katelin Today at 9:18 AM
we got this

jesu showed up! Yesterday at 2:38PM

Shawn Today at 9:18 AM
anyone wanna study together in voice?
3 3

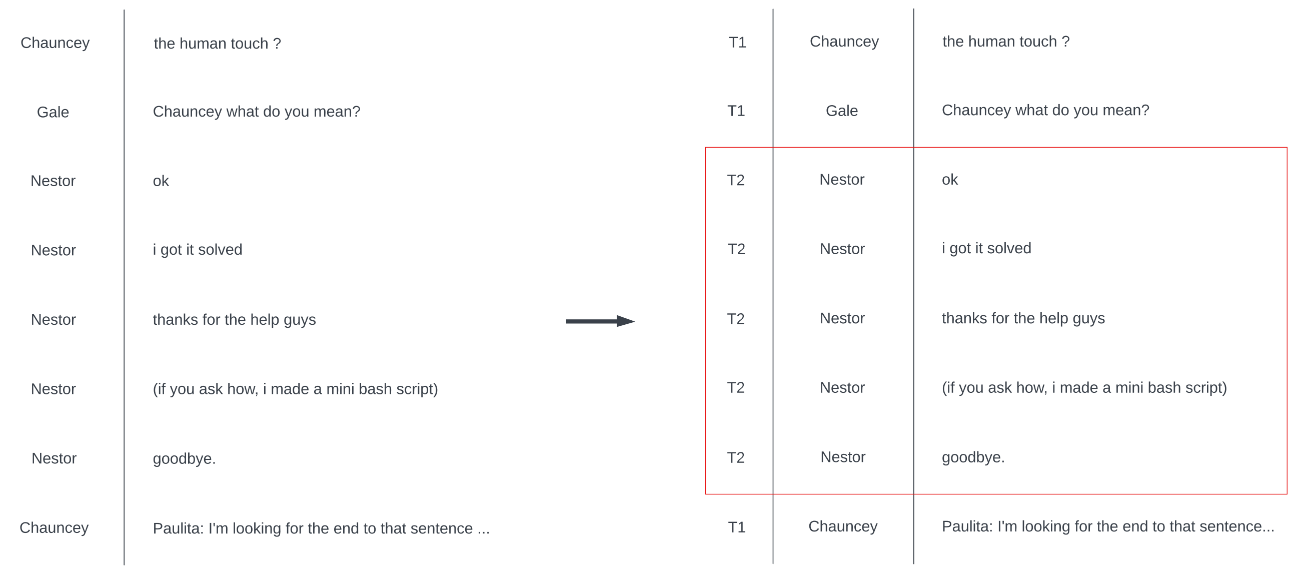
moongirl Today at 9:18 AM
this video really helped me!! give it a watch, it explains it pretty clearly
https://youtu.be/OiDx6aQ928o

(D) *Discord* interface - 2015 [7]

FIGURE 2.1: Interfaces of instant message platforms through the years

2.2 Conversation Disentanglement

Conversation disentanglement is the task of clustering messages into a set of conversations. Even for a human reader, it is challenging to reconstruct the conversation flow in real-time. But, for a machine, it represents a highly complex task. IM platforms can reach throughputs of several hundred messages per hour during active conversations. This means that multiple messages can arrive at almost the same time, resulting in interleaving messages about different simultaneous conversations. As we can see in Figure 2.2 (taken from an annotated dataset in the source code of [2]), the conversation between *Chauncey* and *Gale* is interrupted by *Nestor*, who is following up to a possible coding question he had sent previously. The first conversation is resumed as soon as *Nestor* finishes.



Chauncey	the human touch ?
Gale	Chauncey what do you mean?
Nestor	ok
Nestor	i got it solved
Nestor	thanks for the help guys
Nestor	(if you ask how, i made a mini bash script)
Nestor	goodbye.
Chauncey	Paulita: I'm looking for the end to that sentence ...

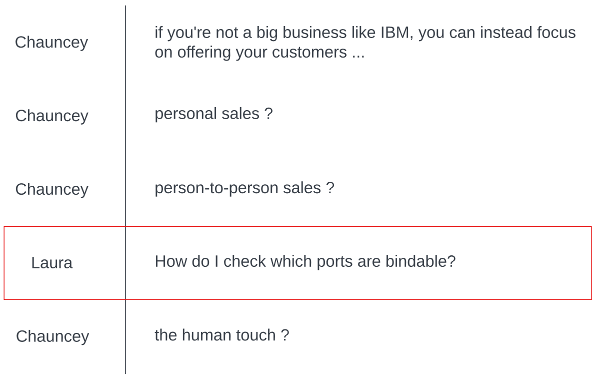
T1	Chauncey	the human touch ?
T1	Gale	Chauncey what do you mean?
T2	Nestor	ok
T2	Nestor	i got it solved
T2	Nestor	thanks for the help guys
T2	Nestor	(if you ask how, i made a mini bash script)
T2	Nestor	goodbye.
T1	Chauncey	Paulita: I'm looking for the end to that sentence...

FIGURE 2.2: Example of fragmented conversation

Another interesting thing to notice is that on Instant Messaging platforms, people tend to write many short messages one after the other rather than sending one long message. This can also be seen in Figure 2.2, where *Nestor*, instead of sending one long message, decides to send five messages, each less than ten words long. This can make the task of creating conversations more difficult. For example, in Figure 2.3, the messages sent by *Chaucey* – which are part of the same conversation – are interrupted by *Laura* asking a completely unrelated question. A possible algorithm should have a way of referring to previous conversations to decide which conversation to assign that last message from *Chaucey*. Aoki et al. [14] demonstrated the complexity of conversation disentanglement by analyzing voice conversations between 10 people. Elsner and Charniak [1, 3] investigated IRC logs disentanglement and proposed the core of the algorithm implemented in this project. Further improvements were introduced by Chatterjee et al. [2], adapting the algorithm to Slack conversations of software developer communities.

According to Liu et al. [5] conversation disentanglement algorithms can be divided into two main categories. The first is composed of two-step algorithms, where the relatedness of message pairs is computed, and then messages are clustered based on this metric. The other category is end-to-end algorithms, where

[4] https://en.wikipedia.org/wiki/File:PLATO-Talkomatic.png
[5] https://c9x.me/irc/irc.c.png
[6] https://techcrunch.com/wp-content/uploads/2017/10/aim.gif
[7] https://www.protocol.com/media-library/discord-chat.png?id=24629797



Chauncey — if you're not a big business like IBM, you can instead focus on offering your customers ...

Chauncey — personal sales ?

Chauncey — person-to-person sales ?

Laura — How do I check which ports are bindable?

Chauncey — the human touch ?

FIGURE 2.3: Example of fragmented conversation

global properties of the conversation flow are captured at once [6]. Elsner and Charniak were among the first researchers to propose a two-step solution, which they used to disentangle IRC chats [3]. Later, this algorithm was adapted by Chatterjee et al. to support Slack chat disentanglement [2] and, just recently, Subash et al. applied to the original algorithm to disentangle Discord chats [6]. An example end-to-end approach has been proposed by Jiang et al. [4] who used a Siamese Hierarchical Convolutional Neural Network to identify threads of conversations. Although of great interest and achieving comparable or even better performances, end-to-end approaches usually offer a black-box tool. We decided to start by implementing a two-step algorithm for its explainability and extensibility, particularly with new features.

2.3 Summary and Outlook

We have seen in this chapter how challenging it is for humans and machines to disentangle conversations. Researchers have found several algorithms using supervised and unsupervised machine learning strategies to ease this task. We decided to implement a two-step approach proposed by Elsner and Charniak [1,3]. In the next chapter, we will present the implementation of *CODI*. We will explain how the service works and how we implemented the original two-step algorithm.

Chapter 3

CODI Implementation

The following chapter will provide an overview of how we developed *CODI*. We present the backend implementing the core services. In the sections dedicated to the frontend, we introduce a web interface to interact and experiment with the backend. The frontend also provides information to evaluate the results of the disentanglement and the performance of the process.

3.1 Architecture

CODI is a web server composed of both frontend and backend and divided into several modules (Figure 3.1). The fronted is composed of the *client* module. The backend includes both the *disentangling* and *REST API* modules. On the one hand, the *REST API* module is responsible for connecting the frontend with the *disentangling* module. On the other hand, the *disentangling* module provides all the logic and algorithms for pre- and post-processing the messages and clustering them into conversations.

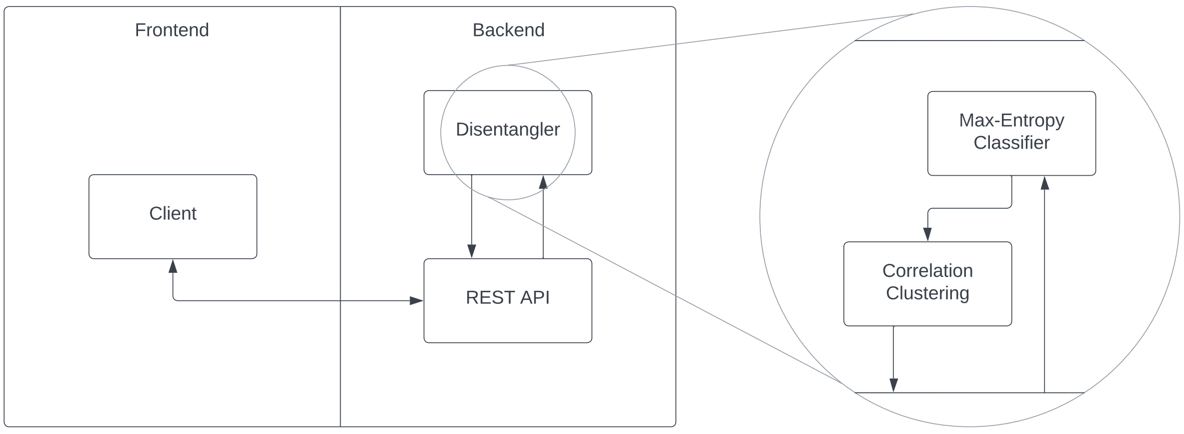


FIGURE 3.1: *CODI* internal architecture

CODI receives as input a JSON file containing the dump of either a Slack or Discord community. The structure of the JSON file can be seen in Figure 3.2. Each file must correspond to one community, which must have an *ID*, a *platform name*, a *name*, a *feature set*, a *list of members*, and a *list of channels*. In the case of the feature set, it is represented as an array. The first element of the array indicates if *chat* features should be included when extracting features from messages. This is a bit, and it can be set to 1 to include this feature group or 0 otherwise. The other two elements of the array are for the *discourse* features and the *content* related features.

The list of members should contain all *authors* – i.e., members who wrote at least one message on one of the channels – and *non-authors*. Furthermore, *CODI* will partially initialize a member not in the list only if another member mentions it in one of the community messages. This new member will either only have an *ID* – since a Discord member mention is of the form `<@123456>` – or an *ID* and a *name* – since a Slack member mention is of the form `<@123456|username>`.

The list of channels must contain channel objects. Each channel must have an *ID*, a *name*, a *path*, and a *list of messages*. Optionally, they can also contain a *list of topics* – which needs to have a *list of keywords*, and a *description*. These messages must each have an *ID*, an *author ID* (which represents the author's ID of the message), a *content*, a *conversation ID* (which is explained in the following paragraph), and a *timestamp* (which has to be in a standard time format). Optionally, messages can also have a *list of attachments* – which have an *url* field of the multimedia content attached to the message

The dataset format to be sent to *CODI* is slightly different based on the operation that the user needs to carry out. These operations are the following:

- **Training**
  This operation is used to train the disentangler's classification model.
- **Validation**
  This operation creates the conversation clusters and validates them with respect to a gold dataset.
- **Prediction**
  This operation creates the conversation clusters.

In the case of training or validation, the dataset must contain a `"conversation": "T1"` field for each message. The format of the `"conversation"` content must be `T + number`. This field is used in training and validation operations to provide a ground truth for the disentangled conversations.

Given the input dataset, the client will call the respective REST API endpoint, which will call the methods necessary for the disentanglement of the conversations. As soon as the disentanglement algorithm finishes, the conversations will be visible on the website.

3.2 Backend

The backend of this project was developed entirely in Python 3.10, with the aid of *Django*[1] – a Python web framework. We decided to use Django because it is a de-facto standard framework for Python web development. In Particular, we used the package *Django REST Framework*[2] to create all the endpoints for our application.

The implementation of the backend follows an Object-Oriented approach. We modeled the domain of instant messaging conversations and translated this model into Python classes supporting the corresponding data structures and operations needed (Figure 3.3). The format of the input JSON file described earlier (Figure 3.2) emerges from this modeling and closely matches the essential aspects of the domain.

[1]Django: The web framework for perfectionists with deadlines `https://www.djangoproject.com/`
[2]Django REST Framework `https://www.django-rest-framework.org/`

```
{
    "platform": "slack",
    "features": [1, 1, 1],
    "id": "b4138f14-af37-4c23-9bda-289bfc36a7fb",
    "name": "training-set",
    "members": [
       {
           "id": "d1ff9c5b-f1fc-47c4-a49d-e2691d103b57",
           "name": "Jermaine"
       },
       ...
    ],
    "channels": [
        {
            "id": "c65d238f-d987-49fe-844b-eef3cfceee4c",
            "name": "discussion",
            "path": "agile/discussion",
            "topics": [
                {
                    "keywords": [
                        "Agile",
                        ...
                    ],
                    "description": "Agile Visualization to its greatest!"
                },
                ...
            ],
            "messages": [
                {
                    "id": "34ce13f1-6577-4710-a007-da4e12d523ef",
                    "authorId": "d1ff9c5b-f1fc-47c4-a49d-e2691d103b57",
                    "content": "But $(date) is in large format",
                    "conversation": "T35",
                    "timestamp": "2022-02-06T19:24:23.777+00:00",
                    "attachments" : [
                        {
                            "url" : "https://cdn.discordapp.com/..."
                        },
                        ...
                    ]
                },
                ...
            ]
        },
        ...
    ]
}
```

FIGURE 3.2: Input JSON dataset

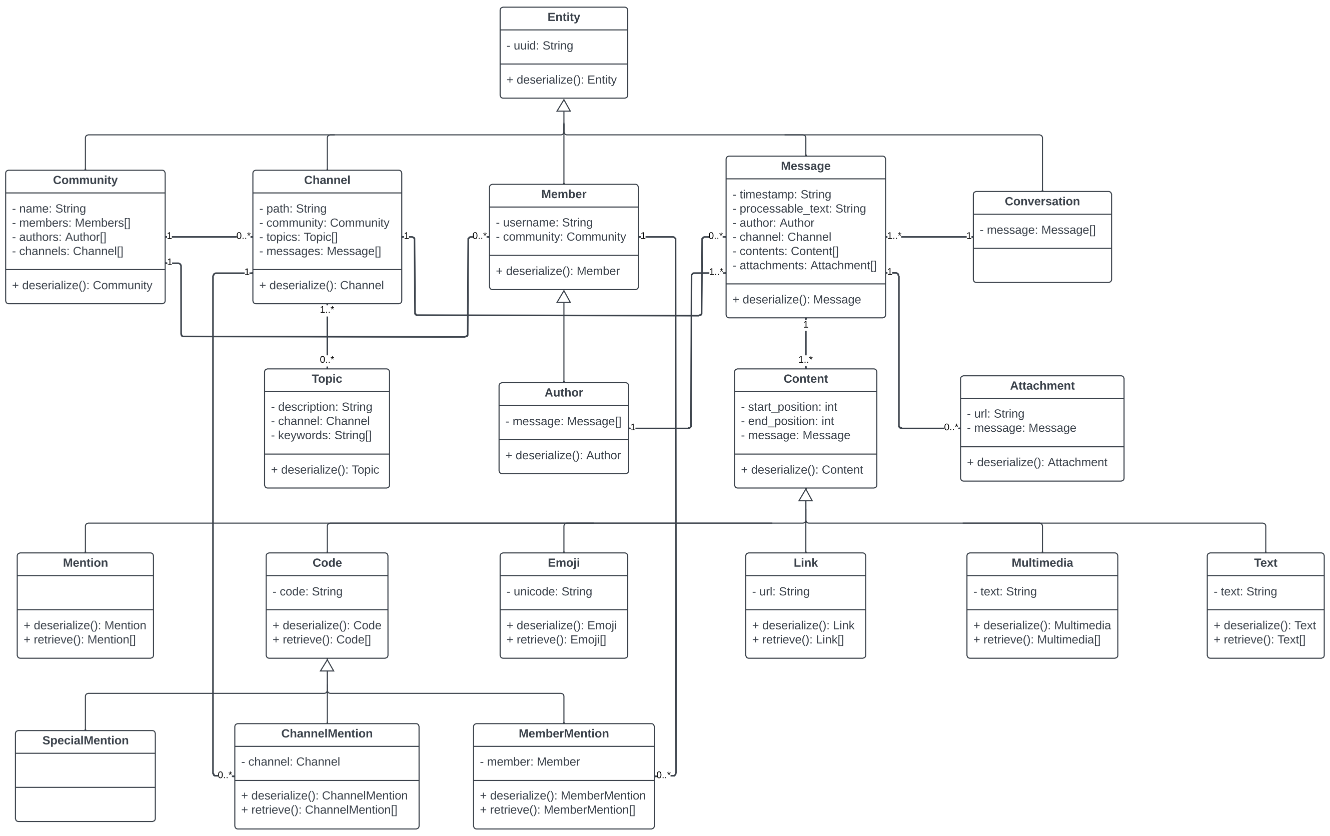


FIGURE 3.3: Class Diagram

3.2.1 REST API

The REST API is built with *Django REST Framework*[2]. This package allows for the creation of endpoints. When a new API endpoint is added, the package will automatically create a frontend view for that endpoint which will only accept the specified HTTP operations.

The API is used to connect the frontend with the disentangling module. The frontend makes API calls every time the user navigates to one of the pages or performs an action. In addition, the API can also be called separately with *cURL* or any other API client for REST (such as *Postman* or *Insomnia*). After an exploratory phase, this feature could be leveraged by researchers or users to integrate the results provided by the application into a custom pipeline for processing conversations.

Table 3.1 illustrates the different endpoints of the microservice. In addition, Figure 3.4 shows the flowchart of the three operations that can be carried out by *CODI*.

3.2.2 Disentangler

The disentangler is the core module of the microservice. Specifically, it is used to group messages into conversations. Conversation grouping is a two-step process consisting of a classifier and a clustering algorithm. For the classification, we used a *max-entropy classifier*. The second step consists of a *correlation clustering* algorithm. It provides as output the groups of messages divided into conversations.

Path	**Type**	**Description**
api/convert	POST	Converts the annotated *ANNOT* dataset to an annotated *JSON* dataset
api/train	POST	Sends the input annotated *JSON* dataset to the disentangler. The latter will use this dataset for training
api/validate	POST	Sends the input annotated *JSON* dataset to the disentangler. The trained disentangler will predict the conversations and compare them to those given by the annotation – i.e., the gold set
api/predict	POST	Sends the input non-annotated *JSON* dataset to the trained disentangler. The latter will predict the conversations
api/statistics	GET	Retrieves the disentangled conversations and – when validating – statistics about the disentangled conversations

TABLE 3.1: REST API endpoints

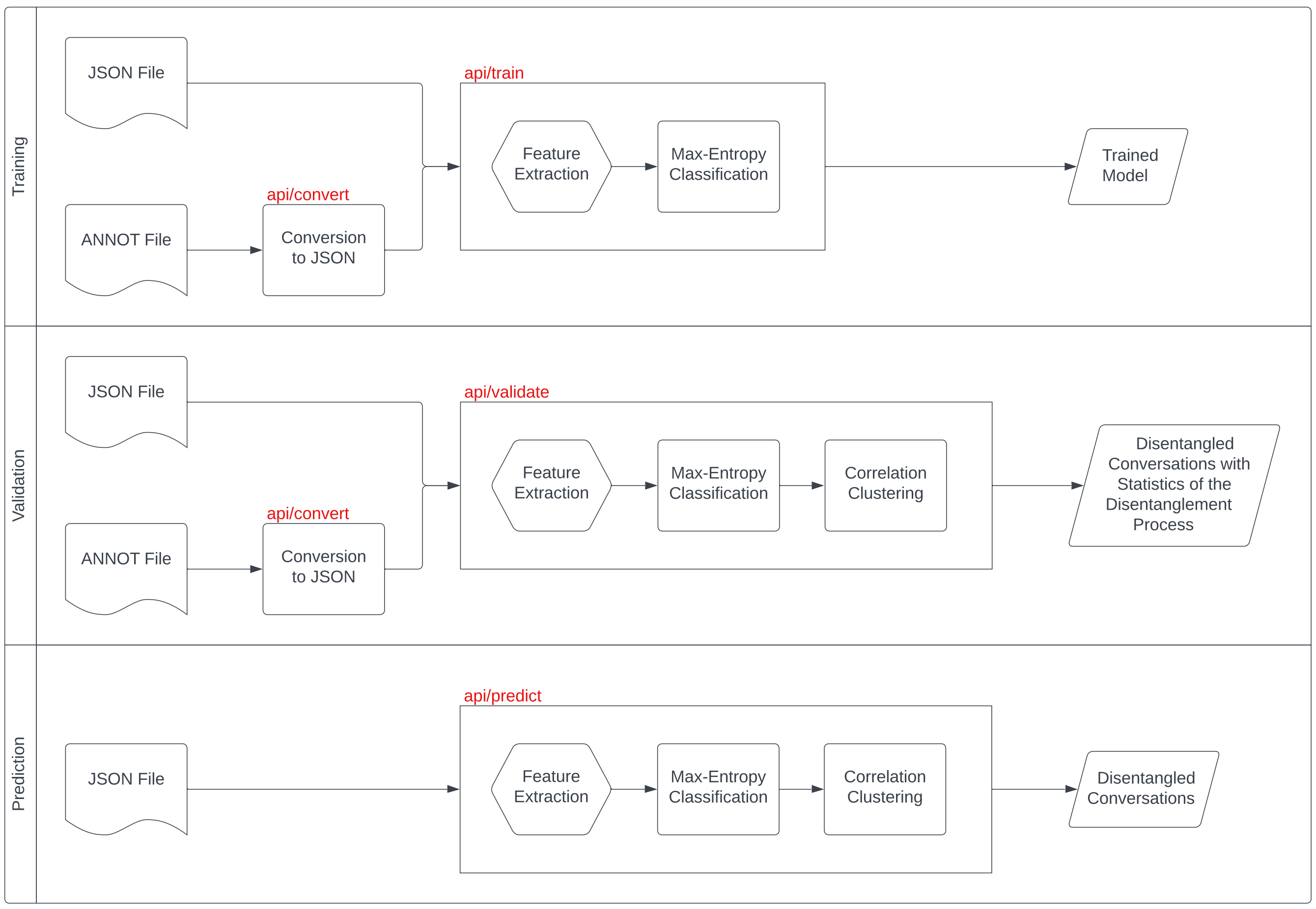


FIGURE 3.4: *Training*, *validation*, and *prediction* flowcharts

Feature Extraction

Before feeding the messages to the classifier, we need a preprocessing phase. Two steps must be performed. In the first step, we extract all pairs of messages given some constraints. For the pair of messages to be examined, they need to be within a time window because "for utterances further apart than this, the classifier has no significant advantage over the majority baseline" [1]. Moreover, as suggested by Chatterjee et al. [3], we also consider message pairs outside the previously defined time window. The four messages preceding the current one are always analyzed independently of their arrival time for relatedness.

In the second step of the pre-processing phase, we extract the features from the pairs of messages previously generated. As done in [1–3], we extract the following features:

- **Chat-Specific Features**
  - **Time**
    The time difference between the first and second messages. This difference is discretized in logarithmically sized bins – which are one-hot encoded.
  - **Speaker**
    If the two messages have the same author or not.
  - **Mention**
    If the message of author one mentions author two, or vice versa. The one-hot encoding of this feature is made up of two bits. The first bit indicates if the first message mentions the author of message two, while the second bit indicates if the second message mentions the author of message one.
  - **Mention Same**
    If both messages mention the same member of the community.
  - **Mention Other**
    If either message one or message two mentions another member of the community who is not the author of either message one or message two. The one-hot encoding of this feature is made up of two bits. The first bit indicates if the first message mentions another community member, while the second bit indicates if the second message mentions another community member.
- **Discourse Features**
  - **Cue Words**
    Either message one or message two uses a greeting (e.g. *hello*), a one-word answer (e.g. *yes*, *no*, *thanks*), or a thanking answer (e.g. *makes sense*, *got it*). This feature is made up of six bits. Two bits are used for each group of cue words – i.e., *greetings*, *one-word answers*, and *thanking answers*. Of these two bits, the first one indicates if the first message contains one of the cue words of that group, while the second bit indicates if the second message contains one of the cue words of that group.
  - **Question**
    Either message one or message two contains a question – which is explicitly marked by a *?*, or it contains a question word (e.g. *what*, *who*). This feature is composed of two bits. The first indicates if the first message contains a question mark or question word, while the second indicates if the second message contains a question mark or question word.
  - **Long**
    Either message one or message two are longer than ten words. This feature is composed of two bits. The first indicates if the first message is longer than ten words, while the second indicates if the second message is longer than ten words.

- **Content Features**
  - **Repeat**
    After building a unigram probability dictionary – of which the first 50 most frequent words are removed, and retrieving all the common words between message one and message two, bin the common words logarithmically based on their probability. Finally, the bins are one-hot encoded into ten bits.
  - **Tech**
    If either message contains technical jargon. This feature is composed of two bits. The first indicates if the first message has technical jargon, while the second indicates if the second message contains technical jargon.
  - **Code**
    If either message one or message two contains a code snippet. This feature is composed of two bits. The first bit indicates if the first message has a code snippet. While the second bit indicates if the second message includes a code snippet.
  - **Link**
    If either message one or message two contains a URL. This feature is composed of two bits. The first bit indicates if the first message contains a URL. while the second bit indicates if the second message contains a URL.

Given these features, their one-hot encodings are condensed in matrix form to be used as input for the classifier. Each row of the matrix corresponds to a pair of messages, and each column corresponds to a bit of the one-hot encoding of a feature. This matrix can now be used as input for the *max-entropy classifier*.

Max-Entropy Classifier

In the first step of the actual disentangling algorithm, we use a binary classification regression model to determine whether the pair of messages are related to one another or not. A *logistic regressor* carries out this classification. Before starting to use *CODI* to disentangle conversations, the model must be trained. The user can use either the frontend or the API to send the regressor an annotated dataset through the `api/train` endpoint. Since this is a supervised machine-learning algorithm, the dataset must be annotated. When the model has finished training, the user can now perform validations and predictions on datasets. To perform validations, an annotated dataset must be sent to the `api/validate` endpoint. In contrast, in the case of a prediction, a non-annotated dataset must be sent to the `api/predict` endpoint of *CODI*.

Correlation Clustering

The final step of the disentangling algorithm consists in clustering the messages. Such a clustering algorithm aims to reconstruct groups of related messages based on the output of the relatedness classifier. This algorithm uses the previously found relations to create clusters of messages most closely related. Such a result can be obtained by implementing a *correlation clustering* algorithm. Being such an algorithm NP-complete, our reference paper [1] opted for a heuristic approach to the problem. In particular, the algorithm they used is a *greedy voting algorithm* [15].

The *greedy voting algorithm* (Algorithm 1) loops on all of the messages in a channel of a community (line 3). We loop on all clusters in C for each of these messages. For each cluster, we compute the sum of the weights – i.e., $w_{ij} = p_{ij} - 0.5$, where p_{ij} is the probability that messages i and j are related – and append it to array Q (line 6). After exhausting all clusters, we must find the clustered index with the highest quality value (line 8). Finally, if the value of $C[c^*]$ is greater than 0, then we can append the current message i to cluster c^* (line 10); otherwise, we create a new cluster and append message i to this new one (line 12).

Algorithm 1 Greedy voting algorithm

```
1:  $C \leftarrow array[\,]$                                  ▷ The array of clusters
2:  $k \leftarrow 0$                                          ▷ The number of clusters
3:  for $i = 1 \dots n$ do
4:      $Q \leftarrow array[\,]$                              ▷ The array of qualities of each cluster
5:      for $c = 1 \dots k$ do
6:          $Q[c] \leftarrow \{\sum_{j \in C[c]} w_{ij}\}$
7:      end for
8:      $c^* \leftarrow \arg\max_{1 \leq c \leq k} Q[c]$
9:      if $Q[c^*] > 0$ then
10:         $C[c^*] \leftarrow C[c^*] \cup \{i\}$             ▷ Add message $i$ to cluster $C[c^*]$
11:     else
12:         $C[k{+}{+}] \leftarrow \{i\}$                     ▷ Create a new cluster and add message $i$ to it
13:     end if
14: end for
```

3.3 Frontend

In the *frontend* module of *CODI*, we have created several pages for users to interact with the backend quickly (Table 3.2). Users can visualize the generated conversations and valuable statistics on the validation process. This module allows the user to select only a subset of the features and perform *training*, *validation*, or *prediction* operations again to evaluate the impact of the selected feature groups on the disentanglement process.

Path	**Name**	**Description**
/	Home	Here the users can upload a JSON file containing the messages of the community.
statistics/validation	Validation Statistics	Here the user can check the performance and statistics of the most recent validation operation performed.
statistics/prediction	Prediction Statistics	Here the user can check the conversations of the most recent prediction operation performed.
annotation	Dataset Annotation	Here the user can manually annotate a dataset given as input.

TABLE 3.2: Frontend pages

3.3.1 Homepage

In the *homepage* of *CODI* (Figure 3.5), on the left, we have the section of the page where the user can drop the JSON file of the community. Instead, we can find the section dedicated to some file settings on the right. The user can specify the type of operation to perform, which can be *training*, *validation* or *prediction*). It can also specify which platform the message was taken from, which can be *auto* – this means that the platform is inferred from the `platform` field of the JSON file, *Discord*, or *Slack*. Finally, we have three checkboxes representing the group of features we want to extract from the messages. These groups are *chat-specific features*, *discourse features*, and *content features*.

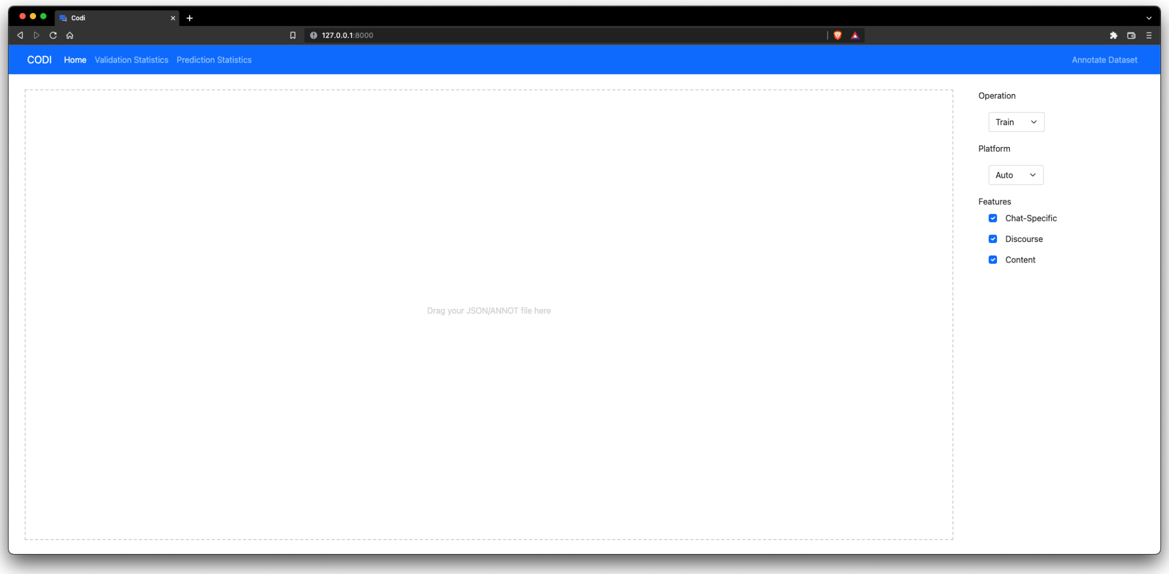
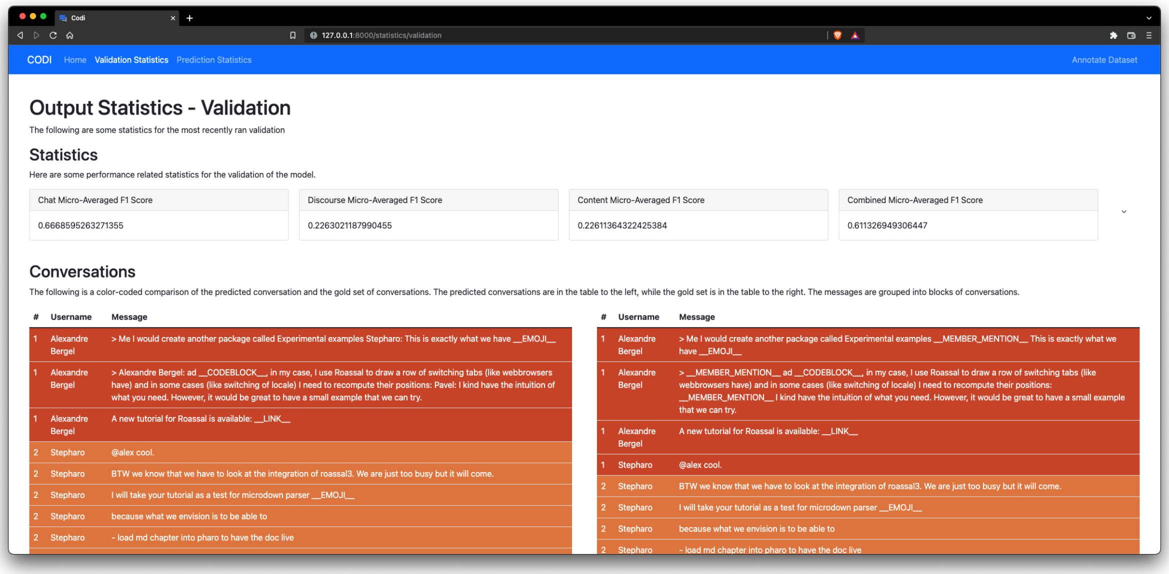


FIGURE 3.5: *Homepage* view

3.3.2 Validation Statistics

In the *validation statistics* section (Figure 3.7), the user can view the statistics of the most recently performed *validation*. The statistics include the *time performance* of the algorithm, the *micro-averaged f1 score* of the disentanglement, and the *accuracy*, *precision*, *recall*, and *f1 score* of the single feature groups – as well as the combined one – of the max-entropy classifier. Finally, the messages are shown with the resulting conversation assignment (Figure 3.6). On the left side, the output of *CODI* can be compared with the ground thruth on the right side. Conversation groups are color coded to help identify them better than by simply looking at their conversation id.



Output Statistics - Validation

The following are some statistics for the most recently ran validation

Statistics

Here are some performance related statistics for the validation of the model.

Chat Micro-Averaged F1 Score	Discourse Micro-Averaged F1 Score	Content Micro-Averaged F1 Score	Combined Micro-Averaged F1 Score
0.6668595263271355	0.2263021187990455	0.22611364322425384	0.611326949306447

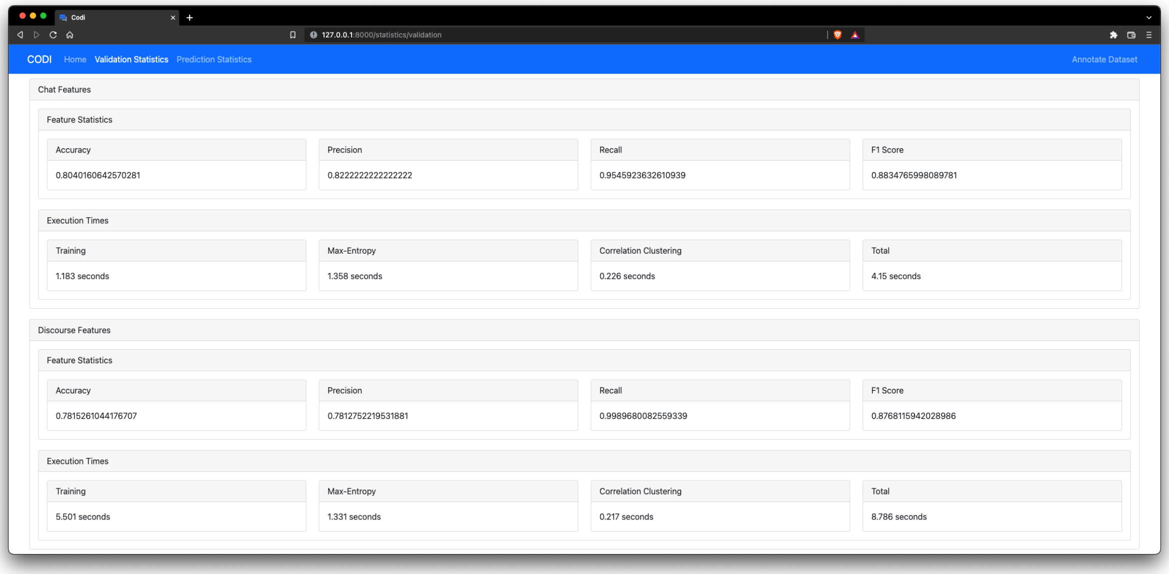
Conversations

The following is a color-coded comparison of the predicted conversation and the gold set of conversations. The predicted conversations are in the table to the left, while the gold set is in the table to the right. The messages are grouped into blocks of conversations.

#	Username	Message
1	Alexandre Bergel	> Me I would create another package called Experimental examples Stepharo: This is exactly what we have __EMOJI__
1	Alexandre Bergel	> Alexandre Bergel: ad __CODEBLOCK__, in my case, I use Roassal to draw a row of switching tabs (like webbrowsers have) and in some cases (like switching of locale) I need to recompute their positions: Pavel: I kind have the intuition of what you need. However, it would be great to have a small example that we can try.
1	Alexandre Bergel	A new tutorial for Roassal is available: __LINK__
2	Stepharo	@alex cool.
2	Stepharo	BTW we know that we have to look at the integration of roassal3. We are just too busy but it will come.
2	Stepharo	I will take your tutorial as a test for microdown parser __EMOJI__
2	Stepharo	because what we envision is to be able to
2	Stepharo	- load md chapter into pharo to have the doc live

#	Username	Message
1	Alexandre Bergel	> Me I would create another package called Experimental examples __MEMBER_MENTION__ This is exactly what we have __EMOJI__
1	Alexandre Bergel	> __MEMBER_MENTION__ ad __CODEBLOCK__, in my case, I use Roassal to draw a row of switching tabs (like webbrowsers have) and in some cases (like switching of locale) I need to recompute their positions: __MEMBER_MENTION__ I kind have the intuition of what you need. However, it would be great to have a small example that we can try.
1	Alexandre Bergel	A new tutorial for Roassal is available: __LINK__
1	Stepharo	@alex cool.
2	Stepharo	BTW we know that we have to look at the integration of roassal3. We are just too busy but it will come.
2	Stepharo	I will take your tutorial as a test for microdown parser __EMOJI__
2	Stepharo	because what we envision is to be able to
2	Stepharo	- load md chapter into pharo to have the doc live

FIGURE 3.6: *Validation statistics* collapsed view


Codi

127.0.0.1:8000/statistics/validation

CODI Home Validation Statistics Prediction Statistics Annotate Dataset

Chat Features

Feature Statistics

Accuracy	Precision	Recall	F1 Score
0.8040160642570281	0.8222222222222222	0.9545923632610939	0.8834765998089781

Execution Times

Training	Max-Entropy	Correlation Clustering	Total
1.183 seconds	1.358 seconds	0.226 seconds	4.15 seconds

Discourse Features

Feature Statistics

Accuracy	Precision	Recall	F1 Score
0.7815261044176707	0.7812752219531881	0.9989680082559339	0.8768115942028986

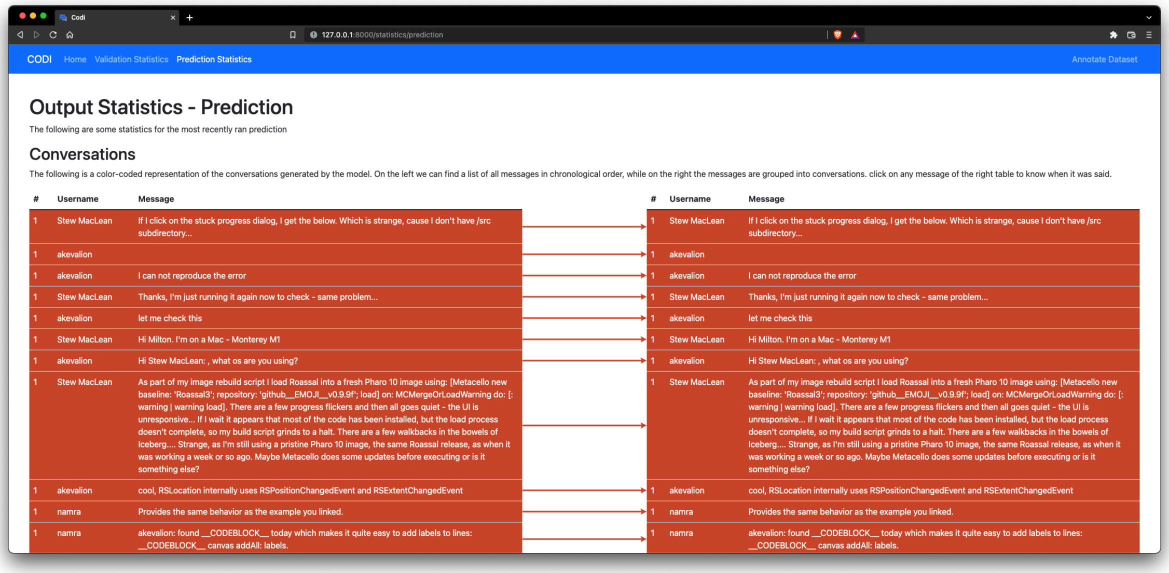
Execution Times

Training	Max-Entropy	Correlation Clustering	Total
5.501 seconds	1.331 seconds	0.217 seconds	8.786 seconds

FIGURE 3.7: *Validation statistics* extended view

3.3.3 Prediction Statistics

In the *prediction statistics* page (Figure 3.8), we have a similar layout as the one seen on the previous page. Unlike the other page, here, the two tables both represent the messages coming from the disentangler. The table on the left represents messages in chronological order, while on the right, we have the same messages but grouped in their corresponding conversation. Moreover, if a user clicks on one of the messages on the left, it will be guided to the same message as assigned to a conversation on the right – and vice-versa.


Codi

127.0.0.1:8000/statistics/prediction

CODI Home Validation Statistics Prediction Statistics Annotate Dataset

Output Statistics - Prediction

The following are some statistics for the most recently ran prediction

Conversations

The following is a color-coded representation of the conversations generated by the model. On the left we can find a list of all messages in chronological order, while on the right the messages are grouped into conversations. click on any message of the right table to know when it was said.

#	Username	Message
1	Stew MacLean	If I click on the stuck progress dialog, I get the below. Which is strange, cause I don't have /src subdirectory...
1	akevalion	
1	akevalion	I can not reproduce the error
1	Stew MacLean	Thanks, I'm just running it again now to check - same problem...
1	akevalion	let me check this
1	Stew MacLean	Hi Milton. I'm on a Mac - Monterey M1
1	akevalion	Hi Stew MacLean: , what os are you using?
1	Stew MacLean	As part of my image rebuild script I load Roassal into a fresh Pharo 10 image using: [Metacello new baseline: 'Roassal3'; repository: 'github__EMOJI__v0.9.9f'; load] on: MCMergeOrLoadWarning do: [: warning | warning load]. There are a few progress flickers and then all goes quiet - the UI is unresponsive... If I wait it appears that most of the code has been installed, but the load process doesn't complete, so my build script grinds to a halt. There are a few walkbacks in the bowels of Iceberg.... Strange, as I'm still using a pristine Pharo 10 image, the same Roassal release, as when it was working a week or so ago. Maybe Metacello does some updates before executing or is it something else?
1	akevalion	cool, RSLocation internally uses RSPositionChangedEvent and RSExtentChangedEvent
1	namra	Provides the same behavior as the example you linked.
1	namra	akevalion: found __CODEBLOCK__ today which makes it quite easy to add labels to lines: __CODEBLOCK__ canvas addAll: labels.

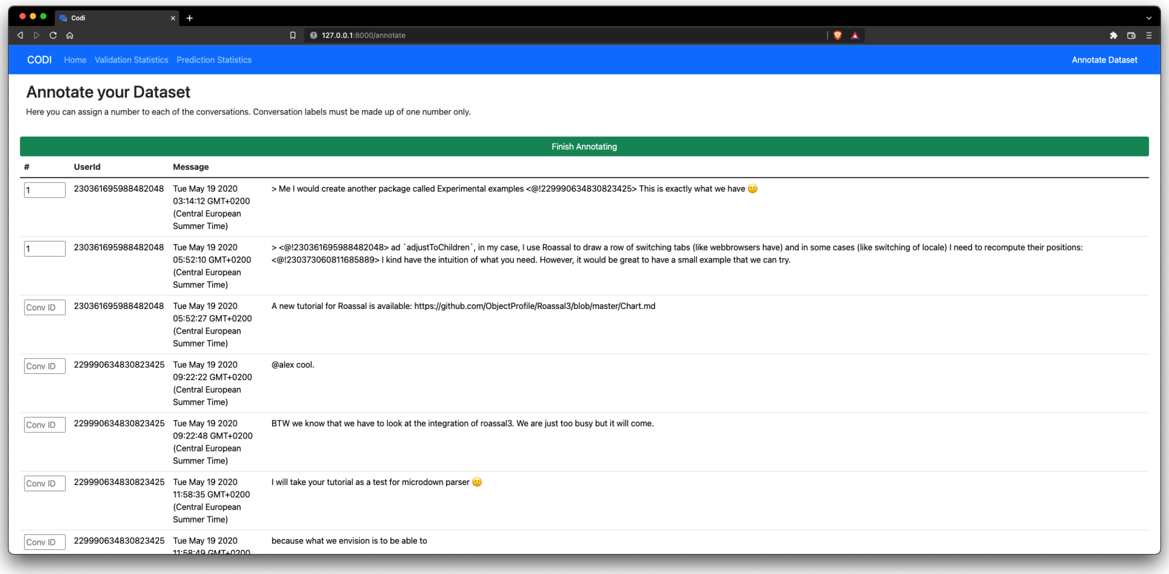
#	Username	Message
1	Stew MacLean	If I click on the stuck progress dialog, I get the below. Which is strange, cause I don't have /src subdirectory...
1	akevalion	
1	akevalion	I can not reproduce the error
1	Stew MacLean	Thanks, I'm just running it again now to check - same problem...
1	akevalion	let me check this
1	Stew MacLean	Hi Milton. I'm on a Mac - Monterey M1
1	akevalion	Hi Stew MacLean: , what os are you using?
1	Stew MacLean	As part of my image rebuild script I load Roassal into a fresh Pharo 10 image using: [Metacello new baseline: 'Roassal3'; repository: 'github__EMOJI__v0.9.9f'; load] on: MCMergeOrLoadWarning do: [: warning | warning load]. There are a few progress flickers and then all goes quiet - the UI is unresponsive... If I wait it appears that most of the code has been installed, but the load process doesn't complete, so my build script grinds to a halt. There are a few walkbacks in the bowels of Iceberg.... Strange, as I'm still using a pristine Pharo 10 image, the same Roassal release, as when it was working a week or so ago. Maybe Metacello does some updates before executing or is it something else?
1	akevalion	cool, RSLocation internally uses RSPositionChangedEvent and RSExtentChangedEvent
1	namra	Provides the same behavior as the example you linked.
1	namra	akevalion: found __CODEBLOCK__ today which makes it quite easy to add labels to lines: __CODEBLOCK__ canvas addAll: labels.

FIGURE 3.8: *Prediction statistics* view

3.3.4 Dataset Annotation

The user can upload a JSON file containing a *Discord* or *Slack* community dump to the dataset annotation page. When *CODI* has finished analyzing the data, it will present a view similar to Figure 3.9. On this tab, the user can manually annotate the conversations for each of the community's messages.

Let's assume that the user wishes to complete annotating the data later. In that instance, the partially annotated dataset can be saved by clicking the "Finish Annotating" button at the top or bottom of the message list. When saving the conversation numbers, a `None` will be added wherever a label has not been yet assigned. Otherwise, the user-specified conversation number will be saved. When the user decides to upload the partially annotated file again, *CODI* will automatically recognize whether a message has already been assigned to a conversation or not and display it accordingly – as shown in Figure 3.9.



Annotate your Dataset

Here you can assign a number to each of the conversations. Conversation labels must be made up of one number only.

#	UserId	Message	
1	230361695988482048	Tue May 19 2020 03:14:12 GMT+0200 (Central European Summer Time)	> Me I would create another package called Experimental examples <@!229990634830823425> This is exactly what we have 🙂
1	230361695988482048	Tue May 19 2020 05:52:10 GMT+0200 (Central European Summer Time)	> <@!230361695988482048> ad `adjustToChildren`, in my case, I use Roassal to draw a row of switching tabs (like webbrowsers have) and in some cases (like switching of locale) I need to recompute their positions: <@!230373060811685889> I kind have the intuition of what you need. However, it would be great to have a small example that we can try.
Conv ID	230361695988482048	Tue May 19 2020 05:52:27 GMT+0200 (Central European Summer Time)	A new tutorial for Roassal is available: https://github.com/ObjectProfile/Roassal3/blob/master/Chart.md
Conv ID	229990634830823425	Tue May 19 2020 09:22:22 GMT+0200 (Central European Summer Time)	@alex cool.
Conv ID	229990634830823425	Tue May 19 2020 09:22:48 GMT+0200 (Central European Summer Time)	BTW we know that we have to look at the integration of roassal3. We are just too busy but it will come.
Conv ID	229990634830823425	Tue May 19 2020 11:58:35 GMT+0200 (Central European Summer Time)	I will take your tutorial as a test for microdown parser 🙂
Conv ID	229990634830823425	Tue May 19 2020	because what we envision is to be able to

FIGURE 3.9: Dataset annotation

Chapter 4

Result Analysis

In this section, we will describe the experiments we carried out to assess the correctness and performance of our version of the two-step method implemented by Elsner and Charniak [1,3], and Chatterjee et al. [2]. The first experiments were carried out utilizing annotated datasets provided by the original authors, which can be obtained from their GitHub repository. More tests were subsequently carried out, this time using the Discord dataset we annotated to determine any potential limitation of such an implementation.

4.1 Evaluation Methods

We use several types of methods to evaluate the solution produced by *CODI*. We use *accuracy*, *precision*, *recall* and *F1 score* for the binary classification – i.e., the max entropy. We use *microaveraged F1 score* for the evaluation of the clustering part of the algorithm – i.e., correlation clustering.

These measurements were useful to compare our implementation with the original one. Since we re-implemented from scratch the *message pairs creation*, the *feature extraction*, the *max-entropy classification* and the *correlation clustering*, we needed to make sure that everything was working correctly and that the performance of *CODI* was on par with the performance of the original implementation by Elsner and Charniak [3] – and build up from there.

4.1.1 Max Entropy

The *accuracy* measures how many times the model guessed correctly. *Precision* measures how precise the model is by considering all of the predicted positives – true positives and false positives. *Recall*, on the other hand, measures how precise the model is by considering all of the actual positives – false negatives and true positives. Finally, the *F1 score* is the harmonic mean between precision and recall. This measurement is used to determine which classifier produces better results.

As done by Elsner and Charniak [1] and Shen et al. [16], let's assume we have a gold conversation i with size n_i, a predicted conversation j with size n_j, and an overlap between the two of size n_{ij}. Then we can write the formulas for *precision*, *recall* and *F1 score* as follows:

$$P = \frac{n_{ij}}{n_j} \qquad R = \frac{n_{ij}}{n_i} \qquad F1(i,j) = 2 \cdot \frac{P \cdot R}{P + R} \tag{4.1}$$

4.1.2 Correlation Clustering

For the evaluation of the clustering algorithm, we cannot use the *F1 score* described above. After clustering the messages, we are not dealing with two classes anymore. The number of classes is now equal to the number of conversations created by the clustering algorithm. Moreover, the conversations generated might not be perfectly lined up with the "gold" conversations. If we were to use the *F1 score*, then a slight misalignment could cause the score to drop and be inaccurate.

Thus, we need to use another evaluation measure that will compute a *global average F1 score*. This evaluation measure is called *micro-averaged F1 score* (Formula 4.2).

$$F1 = \sum_i \frac{n_i}{n} \max_j F1(i,j) \tag{4.2}$$

4.2 Disentanglement Performance

4.2.1 Original Datasets

To verify the correctness of our implementation with respect to the state-of-the-art [1–3], we've tested *CODI* with extracts from both the Python and Clojure datasets present in the literature [6]. We've found that the two implementations give comparable results by running such tests. In Table 4.1 we have the performances of the combined features for both the max-entropy classifier – *accuracy*, *precision*, *recall*, and *F1 score*; and the performance of the correlation clustering algorithm – *micro-averaged f1 score*.

	Accuracy	**Precision**	**Recall**	**F1 Score**	**Micro-Averaged F1 Score**
`python`	60	86	62	72	78
`clojure`	67	94	68	79	88

TABLE 4.1: *CODI* results from original datasets

4.2.2 Custom Dataset

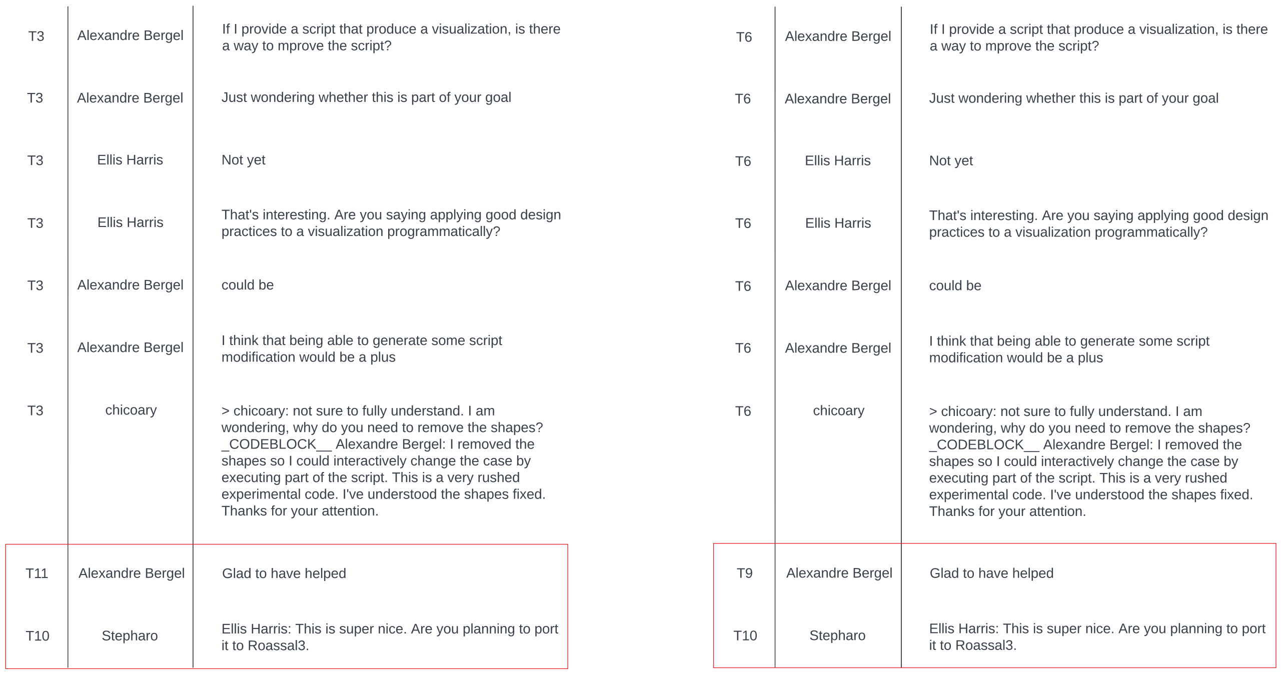
To further test our implementation's robustness and accuracy, we chose to annotate 294 messages from a Discord server about *Roassal* development. We've purposefully chosen these messages because many conversations are interleaving one another. After training the model with the `training_slack.annot` annotated dataset – which has been used by the original authors [1–3] as well to train their models; by performing a validation operation on the `pharo.annot` dataset, we obtained the results displayed in Table 4.2.

	Chat Features	**Discourse Features**	**Content Features**	**Combined Features**
Accuracy	68	64	69	68
Precision	82	80	78	68
Recall	76	72	83	77
F1 Score	79	76	81	79
Micro-Averaged F1	66	53	47	63

TABLE 4.2: *CODI* results from the *Roassal* dataset

After training both *CODI* and Chatterjee et al.'s implementations with the original `training_slack.annot`, we fed `pharo.annot` to both the modified version of the original algorithm [2], and to our algorithm. By doing so, we obtained a *micro-averaged F1 score* of 61 for Chatterjee et al.'s implementation and a *micro-averaged F1 score* of 63 for *CODI*. This result suggests that the algorithm proposed by Chatterjee et al. can sometimes find it challenging to disambiguate certain conversations.

The inability to properly disambiguate conversations may be caused by the fact that this algorithm does not work well with long one-argument conversations, where two or more people talk about the same topic for an extended period of time. We have noticed that, in some cases, it seems like the algorithm forces the separation of messages, which in reality, are part of the same conversation. For example, we can see in Figure 4.1 that both *CODI* and Chatterjee et al.'s algorithm perform similarly. In the red boxes of the two images, we can see how the two messages are being forced into a new conversation – even though they are part of the previous conversation.



T3	Alexandre Bergel	If I provide a script that produce a visualization, is there a way to mprove the script?
T3	Alexandre Bergel	Just wondering whether this is part of your goal
T3	Ellis Harris	Not yet
T3	Ellis Harris	That's interesting. Are you saying applying good design practices to a visualization programmatically?
T3	Alexandre Bergel	could be
T3	Alexandre Bergel	I think that being able to generate some script modification would be a plus
T3	chicoary	> chicoary: not sure to fully understand. I am wondering, why do you need to remove the shapes? _CODEBLOCK__ Alexandre Bergel: I removed the shapes so I could interactively change the case by executing part of the script. This is a very rushed experimental code. I've understood the shapes fixed. Thanks for your attention.
T11	Alexandre Bergel	Glad to have helped
T10	Stepharo	Ellis Harris: This is super nice. Are you planning to port it to Roassal3.

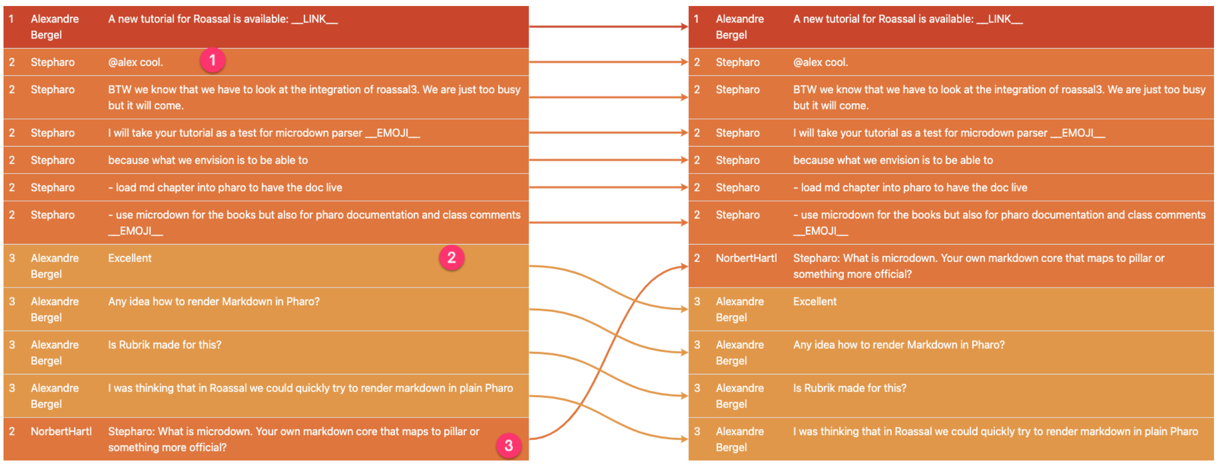
T6	Alexandre Bergel	If I provide a script that produce a visualization, is there a way to mprove the script?
T6	Alexandre Bergel	Just wondering whether this is part of your goal
T6	Ellis Harris	Not yet
T6	Ellis Harris	That's interesting. Are you saying applying good design practices to a visualization programmatically?
T6	Alexandre Bergel	could be
T6	Alexandre Bergel	I think that being able to generate some script modification would be a plus
T6	chicoary	> chicoary: not sure to fully understand. I am wondering, why do you need to remove the shapes? _CODEBLOCK__ Alexandre Bergel: I removed the shapes so I could interactively change the case by executing part of the script. This is a very rushed experimental code. I've understood the shapes fixed. Thanks for your attention.
T9	Alexandre Bergel	Glad to have helped
T10	Stepharo	Ellis Harris: This is super nice. Are you planning to port it to Roassal3.

FIGURE 4.1: Conversation disentanglement comparison – on the *left CODI*, on the *right* Chatterjee et al.

4.3 Disentanglement Quality

In addition to performance analysis of *CODI*, we also perform a qualitative analysis of the generated conversations. This qualitative analysis was performed on the conversations generated by a model trained with `training_slack.annot` and predicted on `pharo.annot`. Figure 4.2 shows a conversation prediction made by *CODI*. In this case, we can see that the algorithm correctly separates conversations 2 and 3. Conversation 2 is resumed with the last message (3). As we can see also on the right of the image, the interleaving conversation is separated from the original conversation (2), and the last message is appended to conversation 2. The second message (1) defines another of the feature extraction problems performed by *CODI*. The problem is that *Stepharo* tries tagging user *Alexandre Bergel* but uses the wrong name. This causes *Discord* to not recognize it as a valid mention, and the exported dataset contains `@alex` instead of `<@ID_OF_ALEXANDRE>`.

Figure 4.3 shows another section of the disentangled conversation. This dataset is the same as the one used in 4.2. In this case, we have a correct disentanglement again. Here we can see that conversation 3 is interrupted by *Ellis Harris*. Correctly, *CODI* considers this a separate conversation and gives it an ID of 9.

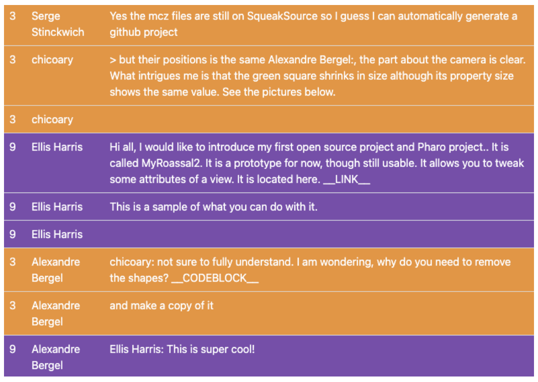


1	Alexandre Bergel	A new tutorial for Roassal is available: __LINK__
2	Stepharo	@alex cool.
2	Stepharo	BTW we know that we have to look at the integration of roassal3. We are just too busy but it will come.
2	Stepharo	I will take your tutorial as a test for microdown parser __EMOJI__
2	Stepharo	because what we envision is to be able to
2	Stepharo	- load md chapter into pharo to have the doc live
2	Stepharo	- use microdown for the books but also for pharo documentation and class comments __EMOJI__
3	Alexandre Bergel	Excellent
3	Alexandre Bergel	Any idea how to render Markdown in Pharo?
3	Alexandre Bergel	Is Rubrik made for this?
3	Alexandre Bergel	I was thinking that in Roassal we could quickly try to render markdown in plain Pharo
2	NorbertHartl	Stepharo: What is microdown. Your own markdown core that maps to pillar or something more official?

1	Alexandre Bergel	A new tutorial for Roassal is available: __LINK__
2	Stepharo	@alex cool.
2	Stepharo	BTW we know that we have to look at the integration of roassal3. We are just too busy but it will come.
2	Stepharo	I will take your tutorial as a test for microdown parser __EMOJI__
2	Stepharo	because what we envision is to be able to
2	Stepharo	- load md chapter into pharo to have the doc live
2	Stepharo	- use microdown for the books but also for pharo documentation and class comments __EMOJI__
2	NorbertHartl	Stepharo: What is microdown. Your own markdown core that maps to pillar or something more official?
3	Alexandre Bergel	Excellent
3	Alexandre Bergel	Any idea how to render Markdown in Pharo?
3	Alexandre Bergel	Is Rubrik made for this?
3	Alexandre Bergel	I was thinking that in Roassal we could quickly try to render markdown in plain Pharo

FIGURE 4.2: Prediction on *pharo.annot*

This conversation is then interrupted by *Alexandre Bergel* – who is answering to *chicoary*. Finally, *Alexandre Bergel* answers to *Ellis Harris*, making the last message part of conversation 9.



3	Serge Stinckwich	Yes the mcz files are still on SqueakSource so I guess I can automatically generate a github project
3	chicoary	> but their positions is the same Alexandre Bergel:, the part about the camera is clear. What intrigues me is that the green square shrinks in size although its property size shows the same value. See the pictures below.
3	chicoary	
9	Ellis Harris	Hi all, I would like to introduce my first open source project and Pharo project.. It is called MyRoassal2. It is a prototype for now, though still usable. It allows you to tweak some attributes of a view. It is located here. __LINK__
9	Ellis Harris	This is a sample of what you can do with it.
9	Ellis Harris	
3	Alexandre Bergel	chicoary: not sure to fully understand. I am wondering, why do you need to remove the shapes? __CODEBLOCK__
3	Alexandre Bergel	and make a copy of it
9	Alexandre Bergel	Ellis Harris: This is super cool!

FIGURE 4.3: Prediction on *pharo.annot*

4.4 Time Performance

In addition to computing the disentangling performance, *CODI* also computes the time each step of the algorithm takes. Table 4.3 summarizes the times of the three datasets we've tested *CODI* on (the results are in seconds).

	Training	**Max-Entropy**	**Correlation Clustering**	**Total**
`python`	103.6	8.5	0.1	166.3
`clojure`	104.1	3.5	0.06	122.5
`pharo`	110	3	0.02	120.5

TABLE 4.3: *CODI* times for all datasets

The *training* column indicates how much time the training of the model took in total. All models were trained with the `training_slack.annot` dataset. The *max-entropy* column indicates the time *CODI* takes to extract all the features from the pairs of messages and make a prediction using the trained max-entropy classifier. The *correlation clustering* column represents the time the algorithm uses to perform the greedy voting algorithm that creates the clusters of conversations. Finally, the *total* column represents the time *CODI* used to train the max-entropy model, extract the features from the message pairs, run the max-entropy classifier on the extracted pairs, and finally cluster the messages into conversations.

4.5 Summary and Outlook

In this section, we've presented comparisons with the reference implementation to assess the correctness of CODI. We've also seen limitations of the algorithm. In some cases, it performs very well, as in the case of the *python* and *clojure* datasets. However, the custom dataset we annotated highlighted its limitations. This is mainly because the messages in the dataset have some problematic sections to disentangle. In the next chapter, we will give some concluding remarks on the projects and suggest some enhancements that can be done to *CODI*.

Chapter 5

Conclusion and Future Work

With this project, we implemented a REST API capable of disentangling conversations using state-of-the-art algorithms. The service, in addition to an API, has a graphical client, which makes the tasks of training the model, validating, and predicting simple and fast. *CODI* also offers – in the case of a validation operation – the possibility to receive statistics regarding the performance of the algorithm and the quality of the conversations. Its ease of use makes it the best tool for even non-technical researchers to choose in the exploratory phases of their research. In addition to being able to disentangle conversations, *CODI* also offers the possibility to upload a non-annotated dataset. This can be manually annotated by the user and exported directly as a JSON file.

5.1 Future Work

Many aspects of *CODI* can be further expanded upon. The following are some possible enhancements:

- **Customizable First Step**
  The first step of the disentangling algorithm can be made customizable. This means that we could offer several algorithms and models to choose from, and the user – from a specific view – can decide which of the algorithms to use.

- **Selectable Hyperparameters**
  Another possible enhancement is to make the hyperparameters selectable. This means that from a particular view, as in the case of the pluggable algorithm discussed earlier, the user can select the values of all the hyperparameters used by *CODI*.

- **API Keys**
  Finally, another possible enhancement would be implementing access based on API keys. This would mean that each user can train and predict disentanglements with *CODI* as if it had many isolated compartments assigned to a key – i.e., a user. By doing so, the models trained and used by each user would not be able to influence any other model trained by any other user of the service.

5.2 Concluding Remarks

As shown throughout this report, our implementation of the two-step algorithm works very well with some datasets and not that well with others. This means that this algorithm is not an all-around good and changes significantly based on the dataset we use to validate the model. Finally, some conversations are challenging – if not impossible – to disentangle even by a human being. This encourages extending *CODI* with new state-of-the-art models, leveraging its architecture to simplify comparisons and extensive experimentation on different datasets.

Bibliography

[1] M. Elsner and E. Charniak, "Disentangling chat," *Computational Linguistics*, vol. 36, no. 3, pp. 389–409, 2010.

[2] P. Chatterjee, K. Damevski, N. A. Kraft, and L. Pollock, "Software-related slack chats with disentangled conversations," in *Proceedings of MSR 2020 (International Conference on Mining Software Repositories)*, pp. 588–592, 2020.

[3] M. Elsner and E. Charniak, "You talking to me? A corpus and algorithm for conversation disentanglement," in *Proceedings of ACL-HLT 2008 (Association for Computational Linguistics: Human Language Technologies)*, pp. 834–842, 2008.

[4] J.-Y. Jiang, F. Chen, Y.-Y. Chen, and W. Wang, "Learning to disentangle interleaved conversational threads with a siamese hierarchical network and similarity ranking," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1812–1822, 2018.

[5] H. Liu, Z. Shi, and X. Zhu, "Unsupervised conversation disentanglement through co-training," *arXiv preprint arXiv:2109.03199*, 2021.

[6] K. M. Subash, L. P. Kumar, S. L. Vadlamani, P. Chatterjee, and O. Baysal, "DISCO: A dataset of discord chat conversations for software engineering research," 2022.

[7] P. Chatterjee, K. Damevski, L. Pollock, V. Augustine, and N. A. Kraft, "Exploratory study of slack q&a chats as a mining source for software engineering tools," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 490–501, IEEE, 2019.

[8] B. Lin, A. Zagalsky, M.-A. Storey, and A. Serebrenik, "Why developers are slacking off: Understanding how software teams use slack," in *Proceedings of the 19th acm conference on computer supported cooperative work and social computing companion*, pp. 333–336, 2016.

[9] E. Parra, A. Ellis, and S. Haiduc, "Gittercom: A dataset of open source developer communications in gitter," in *Proceedings of the 17th International Conference on Mining Software Repositories*, pp. 563–567, 2020.

[10] M. Raglianti, R. Minelli, C. Nagy, and M. Lanza, "Visualizing discord servers," in *2021 Working Conference on Software Visualization (VISSOFT)*, pp. 150–154, IEEE, 2021.

[11] L. Shi, X. Chen, Y. Yang, H. Jiang, Z. Jiang, N. Niu, and Q. Wang, "A first look at developers' live chat on gitter," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 391–403, 2021.

[12] V. Stray and N. B. Moe, "Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and slack," *Journal of Systems and Software*, vol. 170, p. 110717, 2020.

[13] O. Ehsan, S. Hassan, M. E. Mezouar, and Y. Zou, “An empirical study of developer discussions in the gitter platform,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 1, pp. 1–39, 2020.

[14] P. M. Aoki, M. H. Szymanski, L. Plurkowski, J. D. Thornton, A. Woodruff, and W. Yi, “Where’s the "party" in "multi-party"? Analyzing the structure of small-group sociable talk,” in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pp. 393–402, 2006.

[15] M. Elsner and W. Schudy, “Bounding and comparing methods for correlation clustering beyond ILP,” in *Proceedings of the Workshop on Integer Linear Programming for Natural Language Processing*, pp. 19–27, 2009.

[16] D. Shen, Q. Yang, J.-T. Sun, and Z. Chen, “Thread detection in dynamic text message streams,” in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 35–42, 2006.