Jean-Thomas FIDALGO CARDOSO (71808047), Alexandre NECHAB (22001225)

Link to the Colab notebook: https://colab.research.google.com/drive/1aI5z5b3A75fFs8NemDVOyTXNtgL-qKgD?usp=sharing

Comparing methods for classifying word sense definitions

M2 Computational linguistics - ML3 - Marie Candito

NB: for this lab, work in groups of 2 students.

Report to turn in on moodle by **January 5**.

Task

The task is to classify word sense definitions into coarse semantic categories named "hypersenses".

In this lab, the objective is to explore several ways of using LLMs for this task.

Data

We will use a dataset of manually labeled word senses, adapted from (Barque et al., 2020) by N. Angleraud in his M2 internship.

These word senses were labeled with a "supersense", each supersense corresponding to a broader "hypersense".

```
%%capture
!pip install -U bitsandbytes
!pip install datasets evaluate accelerate peft

import pandas as pd
import gzip
import os
import matplotlib.pyplot as plt
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer,
pipeline, BitsAndBytesConfig, AutoModelForSequenceClassification,
TrainingArguments, Trainer
from sklearn.metrics import f1_score
import seaborn as sns
from datasets import Dataset, DatasetDict
```

```python
from peft import LoraConfig, TaskType, get_peft_model
import numpy as np
import evaluate
```

```
d:\Cours\Master\Multilingual NLP\Lab2\env\lib\site-packages\tqdm\
auto.py:21: TqdmWarning: IProgress not found. Please update jupyter
and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```python
def download_and_load_tsv_gz(url):
  """Downloads a gzipped TSV file and loads it into a pandas
DataFrame.
  Args:
    url: The URL of the gzipped TSV file.

  Returns:
    A pandas DataFrame containing the data from the file.
  """
  filename = url.split('/')[-1]
  if not os.path.exists(filename):
    !wget "$url"
  else:
    print(f"File '{filename}' already exists. Skipping download.")

  with gzip.open(filename, 'rt', encoding='utf-8') as f:
    df = pd.read_csv(f, sep='\t')

  return df


url =
'http://www.linguist.univ-paris-diderot.fr/~mcandito/divers/super_wikt
ionary_labeled_data_simplified.noex.25labels.tsv.gz'

try:
  df = download_and_load_tsv_gz(url)
  print("DataFrame loaded successfully.")
  print(df.head())  # Display the first few rows
except Exception as e:
  print(f"Error loading DataFrame: {e}")
```

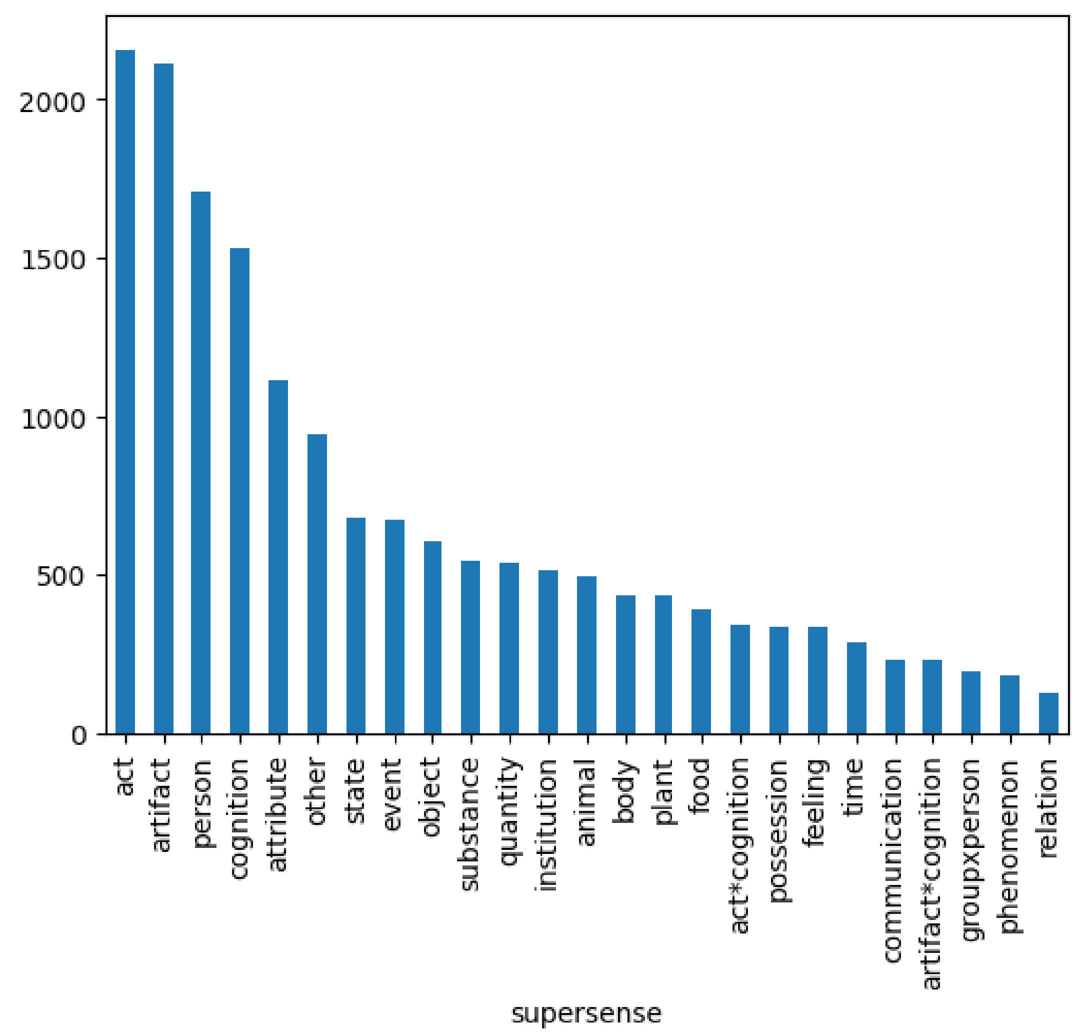
```
File 'super_wiktionary_labeled_data_simplified.noex.25labels.tsv.gz'
already exists. Skipping download.
DataFrame loaded successfully.
                    sense_id                 entry_id         lemma  \
0      ws_1_esquerme__nom__1      esquerme__nom__1      esquerme   
1    ws_1_béruguette__nom__1    béruguette__nom__1    béruguette   
2  ws_2_agroécologie__nom__1  agroécologie__nom__1  agroécologie   
3     ws_1_webcaméra__nom__1     webcaméra__nom__1     webcaméra   
4     ws_1_Reguinois__nom__1     Reguinois__nom__1     Reguinois   
```

```
   nb_occ_lemma_frsemcor supersense        set  \
0                      0      plant  rand-test
1                      0       food   rand-dev
2                      0        act  rand-test
3                      0   artifact  rand-test
4                      0     person   rand-dev

                                          definition
0                                  Plante potagère.
1            Variété d’olive allongée à pulpe ferme.
2  Pratique ou mode de production agricole appliq...
3                               Synonyme de webcam.
4  Habitant de Réguiny, commune française située ...
```

Distribution of supersenses

```
df['supersense'].value_counts().plot(kind='bar')
```

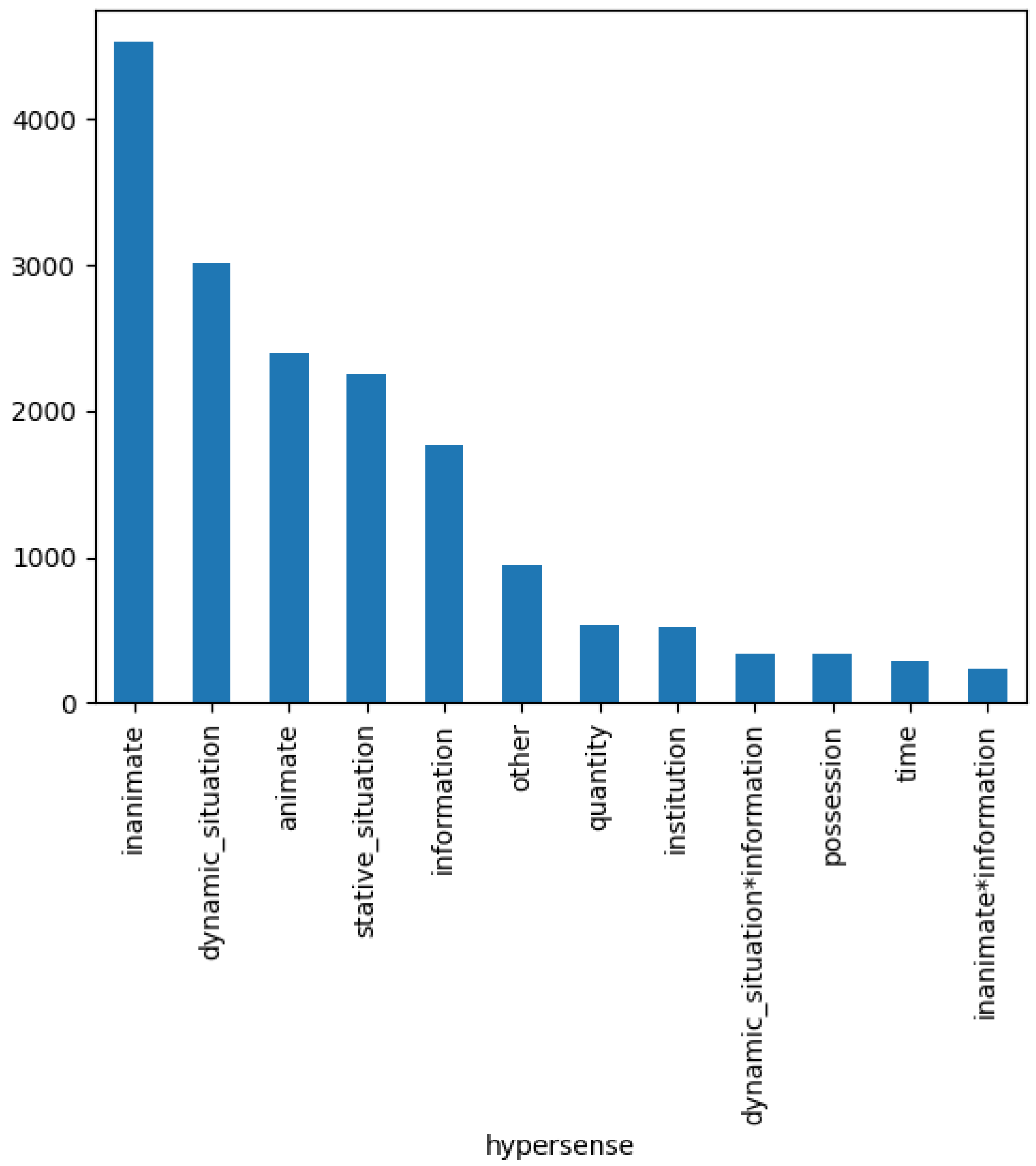
```
<Axes: xlabel='supersense'>
```



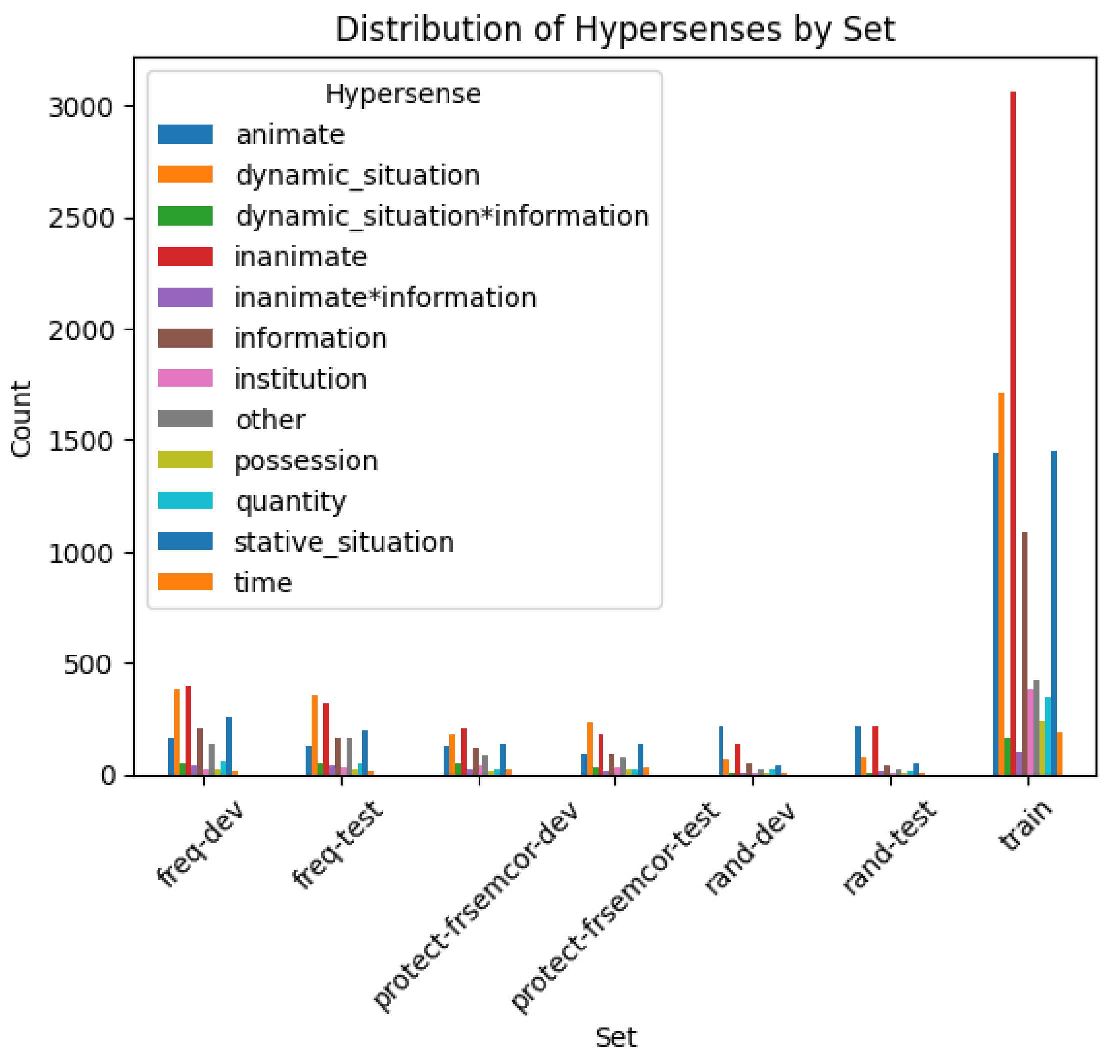
Distribution of hypersenses

```
supersenses = df['supersense'].unique()
super2hyper = {'artifact': 'inanimate',
               'body': 'inanimate',
               'food': 'inanimate',
               'object': 'inanimate',
               'plant': 'inanimate',
               'substance': 'inanimate',
               'act':  'dynamic_situation',
               'event': 'dynamic_situation',
               'phenomenon': 'dynamic_situation',
               'animal': 'animate',
               'person': 'animate',
               'groupxperson': 'animate',
               'cognition': 'information',
               'communication': 'information',
               'quantity': 'quantity',
               'attribute': 'stative_situation',
               'feeling': 'stative_situation',
               'relation': 'stative_situation',
               'state': 'stative_situation',
               'time': 'time',
               'possession': 'possession',
               'institution': 'institution',
               'act*cognition': 'dynamic_situation*information',
               'artifact*cognition': 'inanimate*information',
               'other': 'other',
               }

df['hypersense'] = df['supersense'].map(super2hyper)
df['hypersense'].value_counts().plot(kind='bar')
```

```
<Axes: xlabel='hypersense'>
```



```
# PB: note the distribution in the training set, dev and test sets are
quite different

df.groupby('set')
['hypersense'].value_counts().unstack().plot(kind='bar')
plt.title('Distribution of Hypersenses by Set')
plt.xlabel('Set')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='Hypersense')
plt.show()
```



use phi-3, or Gemma family. Try to see models online

Classifiers

You will compare

1. using a causal model with in-context learning (e.g. with N=4,8,16 examples)

- you will use a small LLM (maximum 7B), fine-tuned for instructions
  - either use a mini model (< 2G) on google colab
  - or a quantized version of a little larger model (< 7B)
    - the bitsandbytes module on huggingface can help you : https://huggingface.co/docs/transformers/main/quantization/bitsandbytes
    - there are also some pre-quantized models by unsloth.ai (faster to download, and you save the time to quantize) : https://huggingface.co/unsloth/mistral-7b-bnb-4bit
  - https://llm.extractum.io/static/llm-leaderboards/

    - explain how you chose the model
    - a possible leaderboard for evaluation in languages other than English:
        - https://huggingface.co/spaces/openGPT-X/european-llm-leaderboard
- or use the huggingface API to run the model on HF's servers
    - cf. https://huggingface.co/docs/huggingface_hub/guides/inference
    - (I don't know the exact amount of inference you can do with the free version)
- Look for prompts tested in the literature to perform text classification, and adapt it to (1) French, and to (2) the type of texts in this lab, namely word sense definitions.

1. parameter-efficient fine-tuning on the full training set

- see the numerous examples on the web for using LoRA or QLoRA in a google colab

1. optional: Flaubert (a bert for french) full fine-tuning + classification, better or not

Methodology

The dataset comprises a training set (cf. column "set" of the dataframe), and several dev and test sets.

In all your work you will ignore the "protect-frsemcor" instances. You will use rand-dev for tuning hyperparameters (including the form of prompts, the nb of demonstrations in ICL etc...), and report your FINAL results on freq-test and rand-test, for the two main methods (ICL and LoRA fine-tuning).

In context learning part: find good prompts which examples to use as example, how many

How your project will be evaluated

You will turn in a PDF file, with a report on your methodology, experiments, problems encountered, results etc...

- you should point to your colab within the pdf report (include link to notebook in pdf)
- or the pdf can be directly created from google colab, with your code and execution traces, in which case, write down in the colab the link to itself.

It is likely that you will use code seen on the web or generated via gemini, so you need to show me you perfectly master your code : write a lot of comments.

The overall performance is not what matters most, in particular because it would be a waste of compute energy to make too many experiments.

You project will be evaluated based on

- sound methodology
- clear and commented code ("Need to understand what you are doing")
- well presented and discussed results, problems you encountered, solved them ?
- error analysis (trying to figure out what kind of instances are badly classified and why?)

Simple Baseline

https://huggingface.co/microsoft/Phi-3-mini-4k-instruct

```
torch.random.manual_seed(0) # set a specific seed to generate the same
set of numbers every time we run the code
model = AutoModelForCausalLM.from_pretrained(
    "microsoft/Phi-3-mini-4k-instruct",
    device_map="cuda",
    torch_dtype="auto",
    trust_remote_code=True,
) #  load the model
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```

{"model_id":"b5533c616b1b4db28cf8f25d5a17c8a7","version_major":2,"vers
ion_minor":0}

{"model_id":"f8bfa4cff331468486f18ec01b1b834c","version_major":2,"vers
ion_minor":0}

```
A new version of the following files was downloaded from
https://huggingface.co/microsoft/Phi-3-mini-4k-instruct:
- configuration_phi3.py
. Make sure to double-check they do not contain any added malicious
code. To avoid downloading new versions of the code file, you can pin
a revision.
```

{"model_id":"f95e8df102c24f078e0300550b0e8a7f","version_major":2,"vers
ion_minor":0}

```
A new version of the following files was downloaded from
https://huggingface.co/microsoft/Phi-3-mini-4k-instruct:
- modeling_phi3.py
. Make sure to double-check they do not contain any added malicious
code. To avoid downloading new versions of the code file, you can pin
a revision.
WARNING:transformers_modules.microsoft.Phi-3-mini-4k-
instruct.0a67737cc96d2554230f90338b163bc6380a2a85.modeling_phi3:`flash
-attention` package not found, consider installing for better
performance: No module named 'flash_attn'.
```

```
WARNING:transformers_modules.microsoft.Phi-3-mini-4k-
instruct.0a67737cc96d2554230f90338b163bc6380a2a85.modeling_phi3:Curren
t `flash-attention` does not support `window_size`. Either upgrade or
use `attn_implementation='eager'`.

{"model_id":"643642322a824c7080bd82970ba62add","version_major":2,"vers
ion_minor":0}

{"model_id":"9f9ef44f2fc440ffbac2e66e16725e1c","version_major":2,"vers
ion_minor":0}

{"model_id":"917ec644a61042dbbd3f8926ed232cb2","version_major":2,"vers
ion_minor":0}

{"model_id":"86f32860fcba40c9ac045caa7248bf55","version_major":2,"vers
ion_minor":0}

{"model_id":"7de1442a395140f5b5f23dca105dd734","version_major":2,"vers
ion_minor":0}

{"model_id":"2b858590b0bd4f959fb23e1bad1a3c1d","version_major":2,"vers
ion_minor":0}

pd.options.mode.chained_assignment = None  # default='warn', supress
pandas warning when setting a value in a slice of a DataFrame

tokenizer = AutoTokenizer.from_pretrained("microsoft/Phi-3-mini-4k-
instruct") # load the tokenizer

hypersenses = ", ".join(set(df["hypersense"].unique())) # str, string
containing the possible hypersenses to put in the prompt

messages = [
    {"role": "system", "content": "You are an expert linguist of
Semantics."},
    {"role": "user", "content": f"Word:'webcaméra', Definition:
'Synonyme de webcam.', Hypersenses: {hypersenses}. From these
hypersenses, which one 'webcaméra' belongs to ? Answer only the
hypersence without additional information"},
    {"role": "assistant", "content": "inanimate"},
    {"role": "user", "content": f"Word:'agroécologie', Definition:
'Pratique ou mode de production agricole appliquant les principes de
l’agroécologie.', Hypersenses: {hypersenses}. From these hypersenses,
which one 'agroécologie' belongs to ? Answer only the hypersence
without additional information"},
    {"role": "assistant", "content": "dynamic_situation"},
    {"role": "user", "content": f"Word:'Reguinois', Definition:
'Habitant de Réguiny, commune française située dans le département du
Morbihan.', Hypersenses: {hypersenses}. From these hypersenses, which
one 'Reguinois' belongs to ? Answer only the hypersence without
additional information"},
```

```python
    {"role": "assistant", "content": "animate"},
] # few-shot prompting

pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
) # loading the pipeline for text-generation

generation_args = {
    "max_new_tokens": 50, # set the maximum number of tokens generated in an answer
    "return_full_text": False, # don't return the beginning of the prompt (text in messages) when generating text
    "do_sample": False, # do no sample words in the probability distribution when generating text, do greedy decoding
}
```

```
{"model_id":"e79cc8b59a554402bbb4e1fbbd26314e","version_major":2,"version_minor":0}

{"model_id":"58679ceb439c453094931335a8d1ebd7","version_major":2,"version_minor":0}

{"model_id":"0129eef8a4d74a1baea123179aeaab18","version_major":2,"version_minor":0}

{"model_id":"634e86fcb1824076b3c910d5c2ff6cd5","version_major":2,"version_minor":0}

{"model_id":"b30318b923694a568b84d98274630de7","version_major":2,"version_minor":0}
```

```
Device set to use cuda
```

```python
# messages: list[dict[str:str]], contains the instructions for the model
def predict_hypersense(messages, lemma, definition):
  current_messages = messages.copy() # copy messages
  # message2add: dict, the message used as the user input
  message2add = {"role": "user", "content": f"Word:'{lemma}', Definition: '{definition}', Hypersenses: {hypersenses}. From these hypersenses, which one '{lemma}' belongs to ? Answer only the hypersence without additional information"}
  current_messages.append(message2add)
  output = pipe(current_messages, **generation_args)
  return output[0]['generated_text'].strip() # we only generate one sentence because we are using greedy decoding, so access it with output[0], then from the dictionnary extract the generated text
```

```
# filter out the examples used for few-shot learning
df = df[~df["lemma"].isin({"Reguinois", "webcaméra", "agroécologie"})]
test_df = df[df["set"] == "rand-test"] # we are using rand-test as the
test set

baseline_test_df = test_df.copy()
baseline_test_df["hypersense_prediction"] =
baseline_test_df.apply(lambda row: predict_hypersense(messages,
row["lemma"], row["definition"]), axis=1)
```

Accuracy

```
# although we start by computing accuracy, f-score might be a better
metric since there is a high class imbalance (see F-Score section
below)
def compute_accuracy(df_with_pred):
  # add column with bool values (True if prediction is correct, False
otherwise)
  df_with_pred["pred_is_correct"] = df_with_pred.apply(
      lambda row: row["hypersense_prediction"].strip() ==
row["hypersense"], axis=1
  )

  # True = 1, False = 0 (by default) so mean() gets the accuracy
  accuracy = df_with_pred["pred_is_correct"].mean()

  return accuracy

print(f"Baseline accuracy: {compute_accuracy(baseline_test_df)}")
```

```
Baseline accuracy: 0.7159763313609467
```

F1 score

```
def compute_f_score(df_with_pred):
  return f1_score(df_with_pred['hypersense'],
df_with_pred['hypersense_prediction'], average="weighted")

print(f"Baseline f1 score: {compute_f_score(baseline_test_df)}")
```

```
Baseline f1 score: 0.6795089955192883
```

Error Analysis

```
dev_df = df[df["set"] == "rand-dev"] # We are using rand-dev as the
dev set

baseline_dev_df = dev_df.copy()
baseline_dev_df["hypersense_prediction"] =
baseline_dev_df.apply(lambda row: predict_hypersense(messages,
row["lemma"], row["definition"]), axis=1)

# save results
baseline_test_df.to_csv("baseline_test_df.csv", index=False)
baseline_dev_df.to_csv("baseline_dev_df.csv", index=False)
```

Accuracy for each class

```
baseline_dev_df.groupby("hypersense")
["pred_is_correct"].mean().reset_index() # "pred_is_correct" is a
column in the dataframe wih bool values, True id the gold and
predicted hypersense are the same, False otherwise, True = 1, False =
0 (by default) so mean() gets the accuracy
# reset_index(), turn the resulting series into a DataFrame
```

```
{"summary":"{\n  \"name\": \"# reset_index(), turn the resulting
series into a DataFrame\",\n  \"rows\": 12,\n  \"fields\": [\n    {\n
\"column\": \"hypersense\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 12,\n
\"samples\": [\n          \"stative_situation\",\n
\"quantity\",\n          \"animate\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"pred_is_correct\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.368817595964541,\n        \"min\": 0.0,\n        \"max\":
0.9117647058823529,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          0.24324324324324326,\n          0.796875,\n
0.6666666666666666\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

The minority classes are not well predicted

Checking if there are predicted classes not part of the possible gold classes

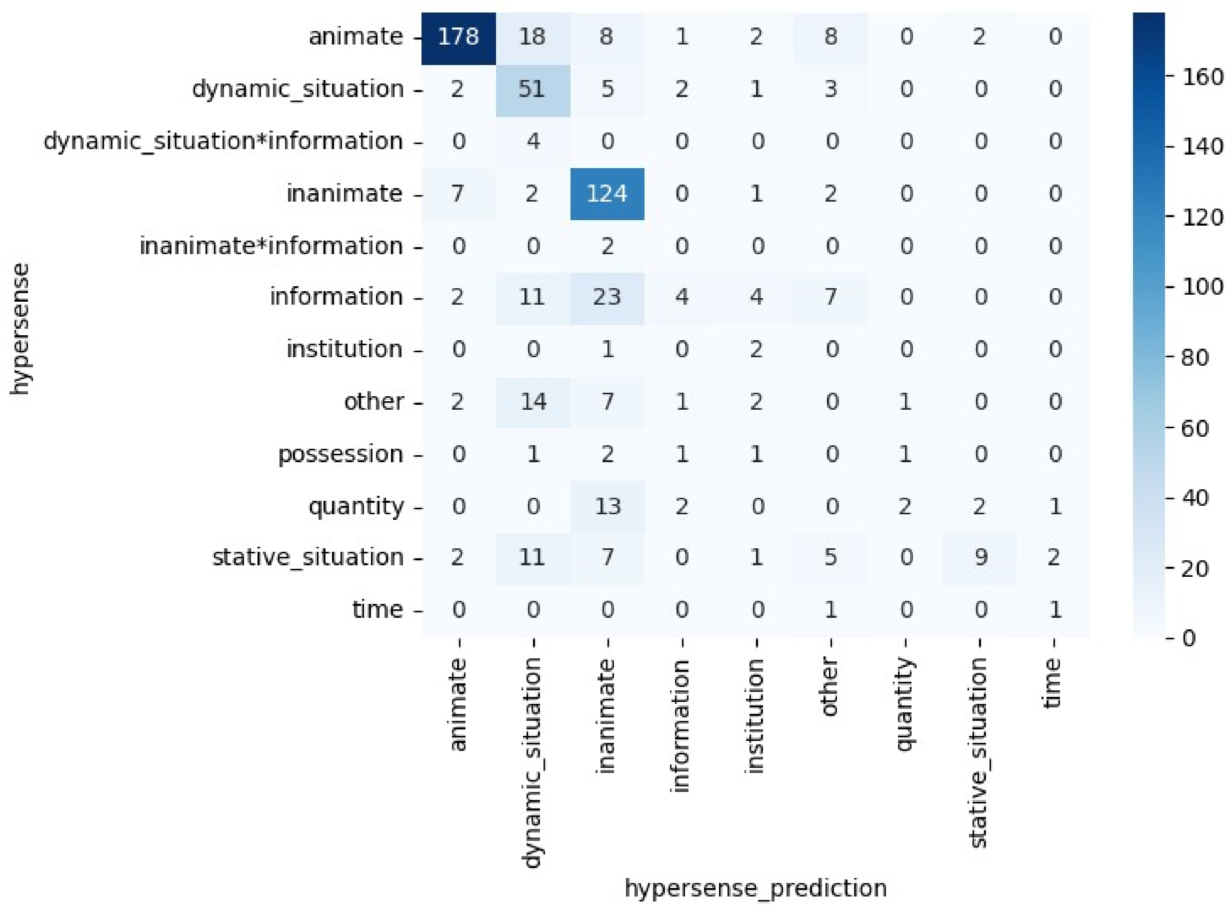
```
baseline_incorrect_df =
baseline_dev_df[~baseline_dev_df["pred_is_correct"]].reset_index()

# shows rows where the hypersense prediction is not one of the gold
hypersenses
baseline_incorrect_df[~baseline_incorrect_df["hypersense_prediction"].
isin(set(df["hypersense"].unique()))]
```

```
{"repr_error":"Out of range float values are not JSON compliant:
nan","type":"dataframe"}
```

The model always predicts one of the possible gold classes, so using a Verbalizer is not usefull.

Confusion Matrix

```
# pd.crosstab: computes a frequency table of the gold and predicted
hypersenses
confusion_matrix = pd.crosstab(baseline_dev_df['hypersense'],
baseline_dev_df['hypersense_prediction'])

# plot the frequency table using seaborn, with annotation in the form
of integers with the Blues color scheme
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues')

<Axes: xlabel='hypersense_prediction', ylabel='hypersense'>
```



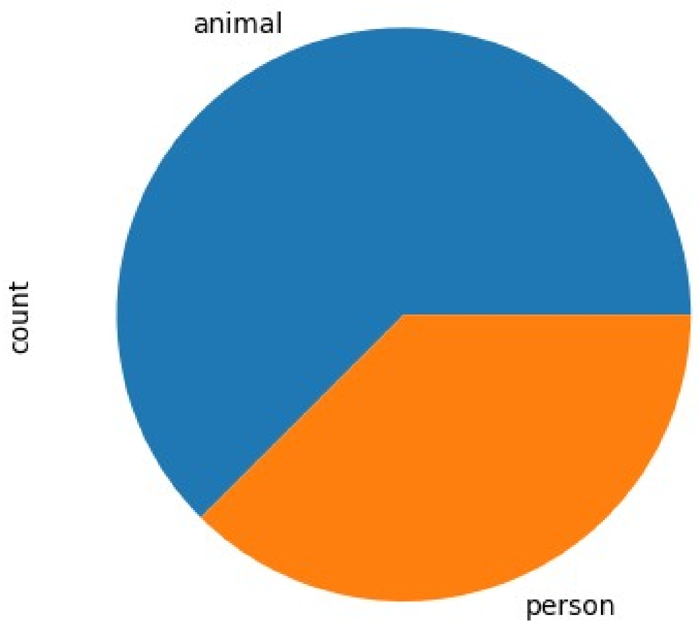
hypersense	animate	dynamic_situation	inanimate	information	institution	other	quantity	stative_situation	time
animate	178	18	8	1	2	8	0	2	0
dynamic_situation	2	51	5	2	1	3	0	0	0
dynamic_situation*information	0	4	0	0	0	0	0	0	0
inanimate	7	2	124	0	1	2	0	0	0
inanimate*information	0	0	2	0	0	0	0	0	0
information	2	11	23	4	4	7	0	0	0
institution	0	0	1	0	2	0	0	0	0
other	2	14	7	1	2	0	1	0	0
possession	0	1	2	1	1	0	1	0	0
quantity	0	0	13	2	0	0	2	2	1
stative_situation	2	11	7	0	1	5	0	9	2
time	0	0	0	0	0	1	0	0	1

- The model predicted badly the minority classes
- The model has a clear preference to predict inanimate and dynamic_situation

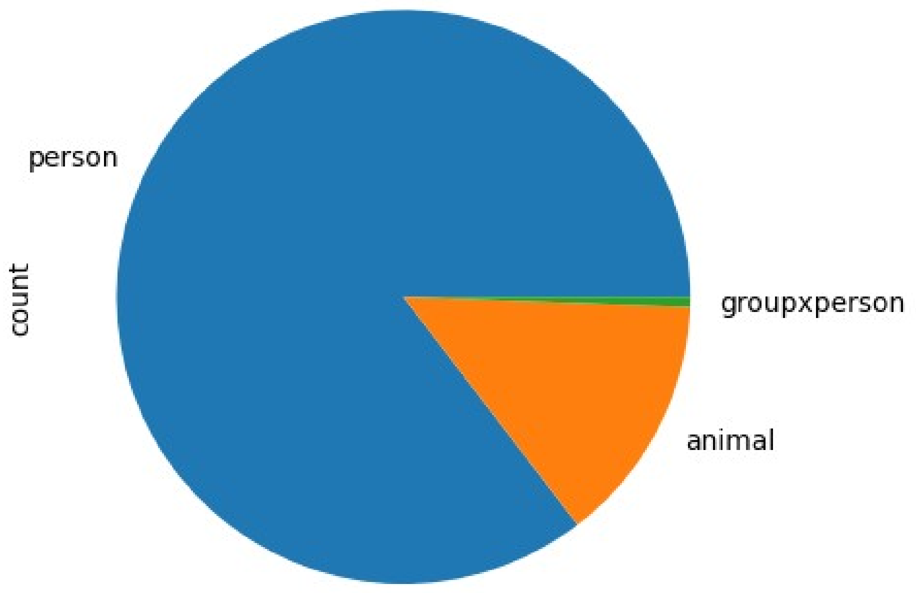
Checking Examples of Errors

Animate predicted as Inanimate

```
animate_pred_inanimate_df =
baseline_incorrect_df[(baseline_incorrect_df["hypersense"] ==
"animate") & (baseline_incorrect_df["hypersense_prediction"] ==
"inanimate")]

animate_pred_inanimate_df["supersense"].value_counts().plot(kind="pie"
)

<Axes: ylabel='count'>
```



```
animate_correct_df =
baseline_dev_df[(baseline_dev_df["hypersense"].str.strip() ==
"animate") & (baseline_dev_df["hypersense_prediction"].str.strip() ==
"animate") ]

animate_correct_df["supersense"].value_counts().plot(kind="pie")

<Axes: ylabel='count'>
```



When the model predicted an animate as inanimate, the word to classify was more frequently related to an animal then a person. This may happen since the model was trained on English and English uses the pronoun "it" to refer to animals, which can be interprete as inanimate. The model may have thus a bias and deduce that animals should be considered as inanimates.

Possible Solutions

- Add more examples from more classes in the prompt to see if the model behaves better, especially for minority classes that were badly predicted (see (1) in messages)
- Add extra examples to pay attention to when the model makes a reasoning mistake. (see (2) in messages)

Improved method: making a better prompt

```
messages = [
    {"role": "system", "content": "You are an expert linguist of
Semantics."},
    {"role": "user", "content": f"Word:'webcaméra', Definition:
'Synonyme de webcam.', Hypersenses: {hypersenses}. From these
hypersenses, which one 'webcaméra' belongs to ? Answer only the
hypersence without additional information"},
    {"role": "assistant", "content": "inanimate"},
    {"role": "user", "content": f"Word:'agroécologie', Definition:
'Pratique ou mode de production agricole appliquant les principes de
l'agroécologie.', Hypersenses: {hypersenses}. From these hypersenses,
which one 'agroécologie' belongs to ? Answer only the hypersence
without additional information"},
    {"role": "assistant", "content": "dynamic_situation"},
    {"role": "user", "content": f"Word:'Reguinois', Definition:
'Habitant de Réguiny, commune française située dans le département du
Morbihan.', Hypersenses: {hypersenses}. From these hypersenses, which
one 'Reguinois' belongs to ? Answer only the hypersence without
additional information"},
    {"role": "assistant", "content": "animate"},
    {"role": "user", "content": f"Word:'oisillon', Definition: 'Petit
oiseau.', Hypersenses: {hypersenses}. From these hypersenses, which
one 'oisillon' belongs to, keeping in mind that animals are alive ?
Answer only the hypersence without additional information"},
    {"role": "assistant", "content": "animate"}, # (2) added animal
example and explained that animals should be considered as animates
    {"role": "user", "content": f"Word:'héritage', Definition: 'Ce qui
se transmet, au sein d’une famille de génération en génération, ou,
plus généralement, de personne à personne.', Hypersenses:
{hypersenses}. From these hypersenses, which one 'héritage' belongs to
? Answer only the hypersence without additional information"},
    {"role": "assistant", "content": "information"}, # (1)
    {"role": "user", "content": f"Word:'proportion', Definition:
'Rapport et convenance des parties entre elles et avec leur tout.',
Hypersenses: {hypersenses}. From these hypersenses, which one
'proportion' belongs to ? Answer only the hypersence without
additional information"},
    {"role": "assistant", "content": "quantity"}, # (1)
    {"role": "user", "content": f"Word:'journée', Definition: 'Période
de rotation d’une planète sur elle-même, en particulier de la Terre.',
Hypersenses: {hypersenses}. From these hypersenses, which one
'journée' belongs to ? Answer only the hypersence without additional
information"},
    {"role": "assistant", "content": "time"} # (1)
]

improved_test_df = test_df.copy()

# make sure the examples used in the prompt are not in the test
dataframe
improved_test_df[improved_test_df["lemma"].isin({"webcaméra","héritage
","proportion","journée","agroécologie","Reguinois", "oisillon"})]

{"repr_error":"Out of range float values are not JSON compliant:
nan","type":"dataframe"}
```

the test df does not contain the examples used as few-shot prompting

---

```
improved_test_df["hypersense_prediction"] =
improved_test_df.apply(lambda row: predict_hypersense(messages,
row["lemma"], row["definition"]), axis=1)
print(f"Improved prompt accuracy:
{compute_accuracy(improved_test_df)}")
```

```
Improved prompt accuracy: 0.6671597633136095
```

```
print(f"Improved prompt f-score: {compute_f_score(improved_test_df)}")
```

```
Improved prompt f-score: 0.6718995552014357
```

```
improved_dev_df = dev_df.copy()

# make sure the examples used in the prompt are not in the dev
dataframe
improved_dev_df[improved_dev_df["lemma"].isin({"webcaméra","héritage",
"proportion","journée","agroécologie","Reguinois", "oisillon"})]
```

{"repr_error":"Out of range float values are not JSON compliant:
nan","type":"dataframe"}

```
improved_dev_df["hypersense_prediction"] =
improved_dev_df.apply(lambda row: predict_hypersense(messages,
row["lemma"], row["definition"]), axis=1)

# save results
improved_test_df.to_csv("improved_test_df.csv", index=False)
improved_dev_df.to_csv("improved_dev_df.csv", index=False)
```

Accuracy for each class

```
baseline_test_df.groupby("hypersense")
["pred_is_correct"].mean().reset_index()
```

{"summary":"{\n \"name\": \"baseline_test_df\",\n \"rows\": 12,\n
\"fields\": [\n   {\n   \"column\": \"hypersense\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 12,\n   \"samples\": [\n
\"stative_situation\",\n   \"quantity\",\n
\"animate\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"pred_is_correct\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.4184913114701726,\n   \"min\":
0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 9,\n
\"samples\": [\n   0.2765957446808511,\n
0.7894736842105263,\n   0.7142857142857143\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
improved_test_df.groupby("hypersense")
["pred_is_correct"].mean().reset_index()
```

```
{"summary":"{\n  \"name\": \"improved_test_df\",\n  \"rows\": 12,\n
\"fields\": [\n    {\n      \"column\": \"hypersense\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 12,\n        \"samples\": [\n
\"stative_situation\",\n          \"quantity\",\n
\"animate\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"pred_is_correct\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.3637491021927218,\n        \"min\":
0.0,\n        \"max\": 0.8732394366197183,\n
\"num_unique_values\": 9,\n        \"samples\": [\n
0.21428571428571427,\n          0.5526315789473685,\n
0.8571428571428571\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

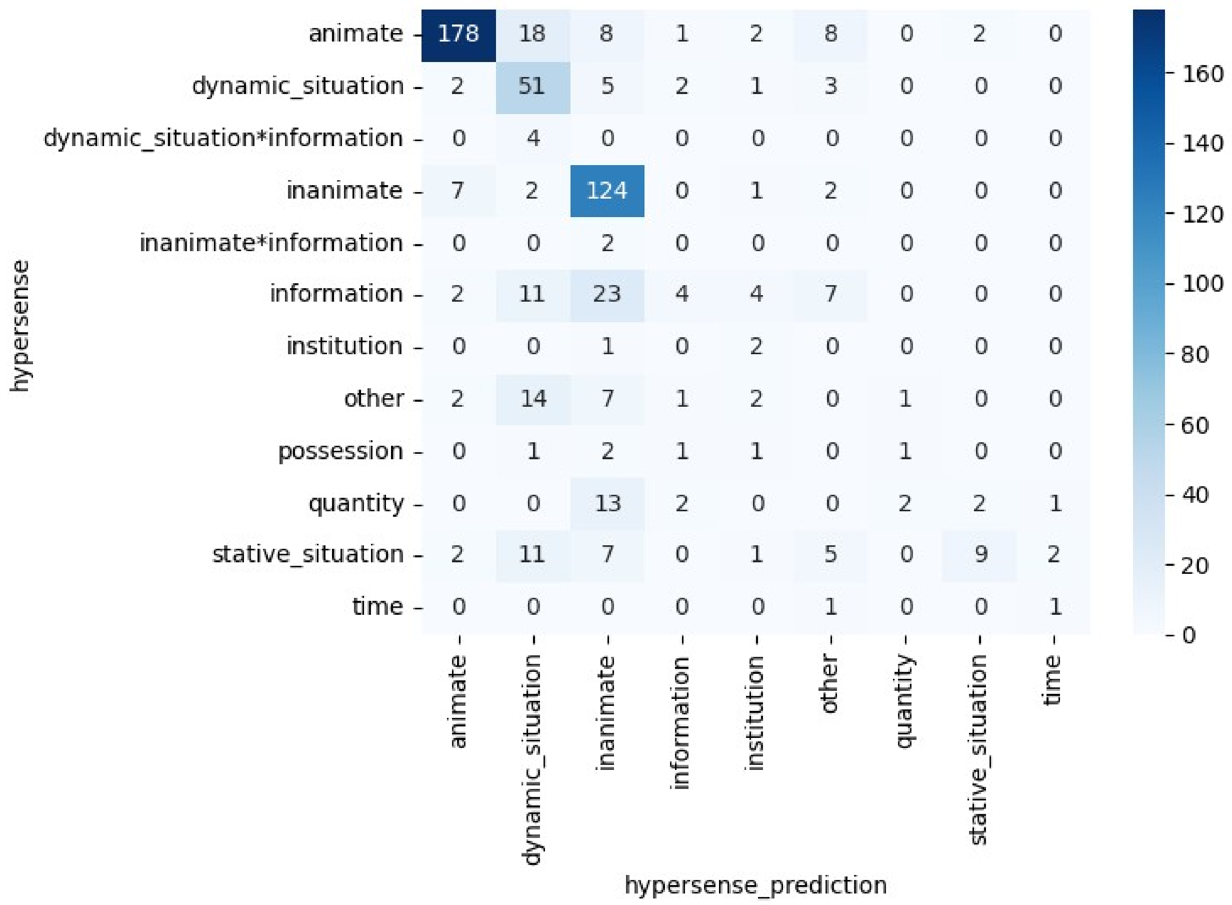
The overall accuracy is worse mainly because the accuracy of the most frequent classes, animate and inanimate, are worse. However, since more examples of minority classes were added in the prompt, the accuracy of the minority classes is better. (quantity, information)

Confusion Matrix (used the dev set for error analysis)

Baseline

```
confusion_matrix = pd.crosstab(baseline_dev_df['hypersense'],
baseline_dev_df['hypersense_prediction'])
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues')
```

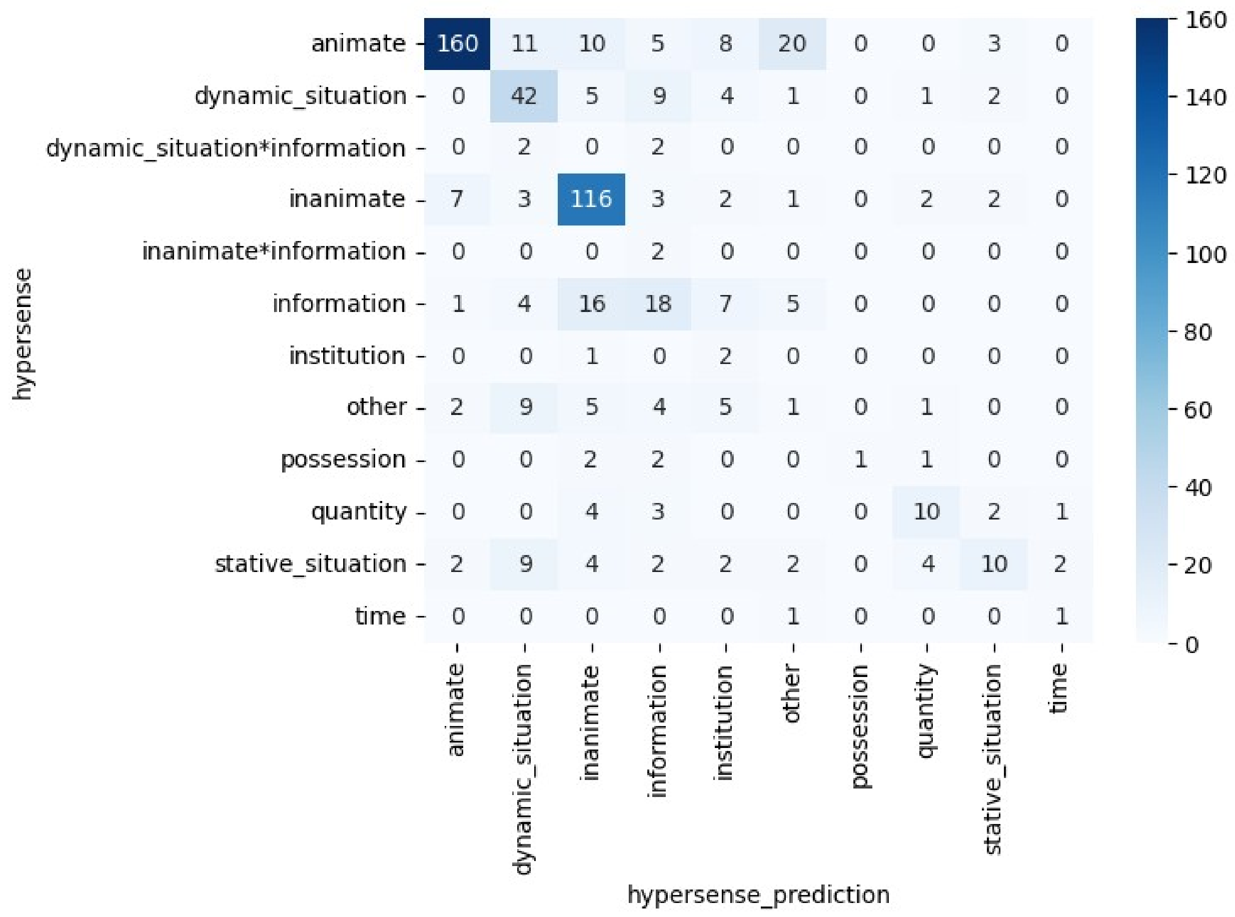
```
<Axes: xlabel='hypersense_prediction', ylabel='hypersense'>
```



hypersense	animate	dynamic_situation	inanimate	information	institution	other	quantity	stative_situation	time
animate	178	18	8	1	2	8	0	2	0
dynamic_situation	2	51	5	2	1	3	0	0	0
dynamic_situation*information	0	4	0	0	0	0	0	0	0
inanimate	7	2	124	0	1	2	0	0	0
inanimate*information	0	0	2	0	0	0	0	0	0
information	2	11	23	4	4	7	0	0	0
institution	0	0	1	0	2	0	0	0	0
other	2	14	7	1	2	0	1	0	0
possession	0	1	2	1	1	0	1	0	0
quantity	0	0	13	2	0	0	2	2	1
stative_situation	2	11	7	0	1	5	0	9	2
time	0	0	0	0	0	1	0	0	1

Improved Prompt

```
confusion_matrix = pd.crosstab(improved_dev_df['hypersense'],
improved_dev_df['hypersense_prediction'])
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues')

<Axes: xlabel='hypersense_prediction', ylabel='hypersense'>
```



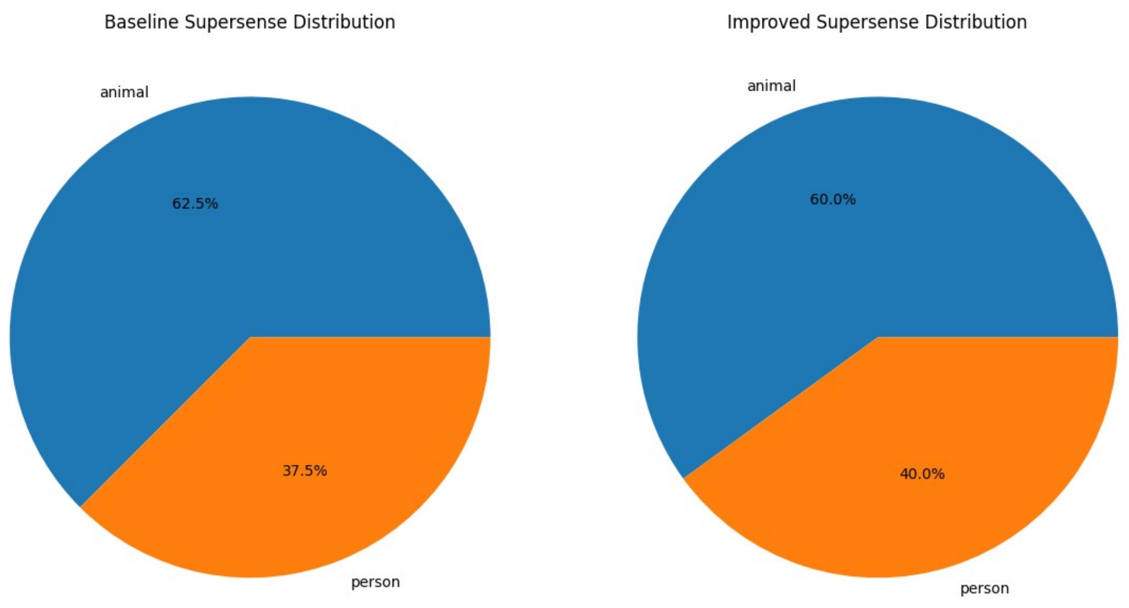
hypersense	animate	dynamic_situation	inanimate	information	institution	other	possession	quantity	stative_situation	time
animate	160	11	10	5	8	20	0	0	3	0
dynamic_situation	0	42	5	9	4	1	0	1	2	0
dynamic_situation*information	0	2	0	2	0	0	0	0	0	0
inanimate	7	3	116	3	2	1	0	2	2	0
inanimate*information	0	0	0	2	0	0	0	0	0	0
information	1	4	16	18	7	5	0	0	0	0
institution	0	0	1	0	2	0	0	0	0	0
other	2	9	5	4	5	1	0	1	0	0
possession	0	0	2	2	0	0	1	1	0	0
quantity	0	0	4	3	0	0	0	10	2	1
stative_situation	2	9	4	2	2	2	0	4	10	2
time	0	0	0	0	0	1	0	0	0	1

we confirm what we saw in the previous graph, the minority classes were predicted better but the most frequent classes were more badly predicted (the correct predictions for animate, inanimate and dynamic_situation are worse)

Animate predicted as Inanimate

```
baseline_dev_df = pd.read_csv("baseline_dev_df.csv")
improved_dev_df = pd.read_csv("improved_dev_df.csv")

baseline_acc = compute_accuracy(baseline_dev_df) # to get
pred_is_correct column
improved_acc = compute_accuracy(improved_dev_df) # to get
pred_is_correct column

improved_incorrect_df =
improved_dev_df[~improved_dev_df["pred_is_correct"]].reset_index()
baseline_incorrect_df =
baseline_dev_df[~baseline_dev_df["pred_is_correct"]].reset_index()

baseline_animate_pred_inanimate_df =
baseline_incorrect_df[(baseline_incorrect_df["hypersense"] ==
"animate")
  & (baseline_incorrect_df["hypersense_prediction"] == "inanimate")]
```

```
improved_animate_pred_inanimate_df =
improved_incorrect_df[(improved_incorrect_df["hypersense"] ==
"animate")
  & (improved_incorrect_df["hypersense_prediction"] == "inanimate")]

fig, axes = plt.subplots(1, 2, figsize=(12, 6))  # 1 row, 2 columns

# Plot the baseline pie chart
baseline_animate_pred_inanimate_df["supersense"].value_counts()\
  .plot(kind="pie", ax=axes[0], autopct='%1.1f%%')
axes[0].set_title("Baseline Supersense Distribution")
axes[0].set_ylabel("")  # Remove y-axis label for a cleaner look

# Plot the improved prompt pie chart
improved_animate_pred_inanimate_df["supersense"].value_counts()\
  .plot(kind="pie", ax=axes[1], autopct='%1.1f%%')
axes[1].set_title("Improved Supersense Distribution")
axes[1].set_ylabel("")  # Remove y-axis label for a cleaner look

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



With the improved prompt, among the examples that were animate but were incorrectly predicted as inanimate, a lower proportion of predictions had the supersense animal. However, this difference is very small (2.5%).

As a consequence, adding an example explaining that animals are living beings and not inanimate helped the model to better predict animals as animates, but only a little bit.

F_score

Baseline F Score:

```
f1_score(baseline_test_df['hypersense'],
baseline_test_df['hypersense_prediction'], average="weighted") #
weighted to account for class imbalance

0.6795089955192883
```

Improved Prompt:

```
f1_score(improved_test_df['hypersense'],
improved_test_df['hypersense_prediction'], average="weighted") #
weighted to account for class imbalance

0.6718995552014357
```

The baseline is better since the f-score of the baseline is better than the f-score of the improved prompt.

We added in the prompt an example for the incorrectly predicted minority classes "quantity" "time" and "information". This resulted in these classes being better predited (although for the class time it is hard to see an effect since there is only 2 examples in rand-dev with the gold class time).

However, this lowered the capacity of the model to correctly predict the most frequent classes like animate, inanimate and dynamic-situation.

A better prompt with more representative examples, or including chain-of-thought prompting, might improve the results of improved prompt.

PEFT

The main difficulty from this section was to train a model without getting a CUDA error. Indeed, many models crashed during the training part. Therefore, the solution was to use a quantized version of a small model (flan-t5-small), instead of using the same Phi-3 model. Also, we had to run this part locally in order to train the model quicker.

```
os.environ["WANDB_DISABLED"] = "true" # Prevents wandb to load on
Colab.
```

Preprocessing the data

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# Creating datasets from df's set column.
```

```
train_df = df[df['set'] == 'train']
dev_df = df[df['set'] == 'rand-dev']
test_df = df[df['set'] == 'rand-test']

# Remove useless columns

train_dataset =
Dataset.from_pandas(train_df).remove_columns(["sense_id", "entry_id",
"nb_occ_lemma_frsemcor", "supersense", "set"])
dev_dataset = Dataset.from_pandas(dev_df).remove_columns(["sense_id",
"entry_id", "nb_occ_lemma_frsemcor", "supersense", "set"])
test_dataset =
Dataset.from_pandas(test_df).remove_columns(["sense_id", "entry_id",
"nb_occ_lemma_frsemcor", "supersense", "set"])

print(train_dataset)
```

```
Dataset({
    features: ['lemma', 'definition', 'hypersense',
'__index_level_0__'],
    num_rows: 10604
})
```

```
labels = df['hypersense'].unique() # List of labels (hypersenses)
id2label = {i: label for i, label in enumerate(labels)}
label2id = {label: i for i, label in enumerate(labels)}
num_labels = len(labels)
print(id2label)
print(label2id)
```

```
{0: 'inanimate', 1: 'dynamic_situation', 2: 'animate', 3:
'information', 4: 'other', 5: 'stative_situation', 6: 'time', 7:
'possession', 8: 'quantity', 9: 'institution', 10:
'dynamic_situation*information', 11: 'inanimate*information'}
{'inanimate': 0, 'dynamic_situation': 1, 'animate': 2, 'information':
3, 'other': 4, 'stative_situation': 5, 'time': 6, 'possession': 7,
'quantity': 8, 'institution': 9, 'dynamic_situation*information': 10,
'inanimate*information': 11}
```

```
# Function to map the labels
def encode_labels(batch):
    batch["label"] = label2id[batch["hypersense"]]
    return batch

train_dataset = train_dataset.map(encode_labels)
dev_dataset = dev_dataset.map(encode_labels)
test_dataset = test_dataset.map(encode_labels)
```

```
Map: 100%|██████████| 10604/10604 [00:00<00:00, 27921.34 examples/s]
Map: 100%|██████████| 570/570 [00:00<00:00, 28594.79 examples/s]
Map: 100%|██████████| 678/678 [00:00<00:00, 25662.73 examples/s]
```

```
tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-small")

def preprocess_function(batch):
    # Convert the 'definition' column to a list of strings before
tokenization
    definitions = batch["definition"]
    # Handle potential None values in definitions
    definitions = [d if d else "" for d in definitions]

    # Pad sequences to a fixed length
    tokenized_output = tokenizer(definitions, truncation=True,
padding="max_length", max_length=128)

    # Ensure that all input_ids have the same length
    max_length = max(len(ids) for ids in
tokenized_output["input_ids"])
    tokenized_output["input_ids"] = [
        ids + [tokenizer.pad_token_id] * (max_length - len(ids))
        for ids in tokenized_output["input_ids"]
    ]
    tokenized_output["attention_mask"] = [
        mask + [0] * (max_length - len(mask))
        for mask in tokenized_output["attention_mask"]
    ]

    return tokenized_output

train_dataset = train_dataset.map(preprocess_function, batched=True)
dev_dataset = dev_dataset.map(preprocess_function, batched=True)
test_dataset = test_dataset.map(preprocess_function, batched=True)
```

```
Map: 100%|██████████| 10604/10604 [00:00<00:00, 18531.39 examples/s]
Map: 100%|██████████| 570/570 [00:00<00:00, 17512.24 examples/s]
Map: 100%|██████████| 678/678 [00:00<00:00, 18385.01 examples/s]
```

```
# Change the format to use PyTorch tensors
train_dataset.set_format("torch", columns=["input_ids",
"attention_mask", "label"])
dev_dataset.set_format("torch", columns=["input_ids",
"attention_mask", "label"])
test_dataset.set_format("torch", columns=["input_ids",
"attention_mask", "label"])
```

Model configuration

Here, the choice of the hyperparameters is important. However, because we do not have enough time to find the best hyperparameters, some values were chosen arbitrarily or influenced by tutorials.

```
# Configuration for quantization
bnb_config = BitsAndBytesConfig(
    load_in_4bit= True, # Use 4-bit precision
    bnb_4bit_quant_type= 'nf4',
    bnb_4bit_compute_dtype= torch.bfloat16, # Reduced float size to 16
    bnb_4bit_use_double_quant= True,
)

# LoRA configuration for fine-tuning
lora_config = LoraConfig(
    lora_alpha=16, # Scaling factor
    lora_dropout=0.1, # Dropout rate (to prevent from overfitting)
    r=4, # Rank
    bias='none', # We want no bias modification
    task_type='SEQ_CLS', # Sequence classification
    target_modules=['q', 'k', 'v'] # The names of the modules query,
key and value
)

# Set the random seed to reproduce the same results.
torch.random.manual_seed(0)

# Base model with bnb configuration
base_model = AutoModelForSequenceClassification.from_pretrained(
    "google/flan-t5-small",
    num_labels=num_labels,
    id2label=id2label,
    label2id=label2id,
    quantization_config=bnb_config, # Load the quantization
    torch_dtype="auto", # Should be CUDA
    trust_remote_code=True,
)
```

```
`low_cpu_mem_usage` was None, now set to True since model is
quantized.
Some weights of T5ForSequenceClassification were not initialized from
the model checkpoint at google/flan-t5-small and are newly
initialized: ['classification_head.dense.bias',
'classification_head.dense.weight',
'classification_head.out_proj.bias',
'classification_head.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```
# Then we apply the LoRA configuration to our model and change the
device to GPU if not done before.
model = get_peft_model(base_model, lora_config).to(device)

metric = evaluate.load("accuracy")
```

```python
# Metric function
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits[0], axis=1)
    return metric.compute(predictions=predictions, references=labels)

# Training arguments
training_args = TrainingArguments(
    output_dir='./checkpoints',
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    num_train_epochs=30,
    weight_decay=0.01, # Regularization to avoid overfitting
    logging_steps=1000 # For better visualization (500 was too much)
)
```

```
Using the `WANDB_DISABLED` environment variable is deprecated and will
be removed in v5. Use the --report_to flag to control the integrations
used for logging result (for instance --report_to none).
```

```python
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=dev_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)

trainer.train()
```

```
  5%|▌         | 1000/19890 [02:58<56:53,  5.53it/s]

{'loss': 2.3521, 'grad_norm': 1.4178788661956787, 'learning_rate':
4.7486173956762195e-05, 'epoch': 1.51}

 10%|█         | 2000/19890 [06:08<52:44,  5.65it/s]

{'loss': 2.101, 'grad_norm': 1.725350260734558, 'learning_rate':
4.497234791352439e-05, 'epoch': 3.02}

 15%|█▌        | 3000/19890 [09:09<49:32,  5.68it/s]

{'loss': 2.0016, 'grad_norm': 2.085395097732544, 'learning_rate':
4.2458521870286574e-05, 'epoch': 4.52}

 20%|██        | 4000/19890 [12:08<45:07,  5.87it/s]

{'loss': 1.9218, 'grad_norm': 2.386924982070923, 'learning_rate':
3.9944695827048774e-05, 'epoch': 6.03}

 25%|██▌       | 5000/19890 [15:05<44:05,  5.63it/s]
```

```
{'loss': 1.8574, 'grad_norm': 2.2598865032196045, 'learning_rate':
3.743086978381096e-05, 'epoch': 7.54}

 30%|███       | 6000/19890 [18:00<38:06,  6.07it/s]

{'loss': 1.8225, 'grad_norm': 2.74214243888855, 'learning_rate':
3.491704374057315e-05, 'epoch': 9.05}

 35%|███▌      | 7000/19890 [21:08<37:58,  5.66it/s]

{'loss': 1.7902, 'grad_norm': 3.2459897994995117, 'learning_rate':
3.2403217697335346e-05, 'epoch': 10.56}

 40%|████      | 8000/19890 [24:10<34:30,  5.74it/s]

{'loss': 1.7546, 'grad_norm': 2.63429594039917, 'learning_rate':
2.988939165409754e-05, 'epoch': 12.07}

 45%|████▌     | 9000/19890 [27:02<30:14,  6.00it/s]

{'loss': 1.7314, 'grad_norm': 2.5767664909362793, 'learning_rate':
2.737556561085973e-05, 'epoch': 13.57}

 50%|█████     | 10000/19890 [30:04<30:30,  5.40it/s]

{'loss': 1.7199, 'grad_norm': 6.1550798416137695, 'learning_rate':
2.486173956762192e-05, 'epoch': 15.08}

 55%|█████▌    | 11000/19890 [33:06<26:15,  5.64it/s]

{'loss': 1.6908, 'grad_norm': 3.1944830417633057, 'learning_rate':
2.2347913524384114e-05, 'epoch': 16.59}

 60%|██████    | 12000/19890 [36:08<24:38,  5.34it/s]

{'loss': 1.6964, 'grad_norm': 4.019471645355225, 'learning_rate':
1.9834087481146303e-05, 'epoch': 18.1}

 65%|██████▌   | 13000/19890 [39:05<20:04,  5.72it/s]

{'loss': 1.6734, 'grad_norm': 2.212674856185913, 'learning_rate':
1.7320261437908496e-05, 'epoch': 19.61}

 70%|███████   | 14000/19890 [42:01<17:11,  5.71it/s]

{'loss': 1.6616, 'grad_norm': 2.5456154346466064, 'learning_rate':
1.4806435394670689e-05, 'epoch': 21.12}

 75%|███████▌  | 15000/19890 [44:58<14:31,  5.61it/s]

{'loss': 1.6556, 'grad_norm': 4.285328388214111, 'learning_rate':
1.229260935143288e-05, 'epoch': 22.62}

 80%|████████  | 16000/19890 [47:53<11:17,  5.74it/s]
```

```
{'loss': 1.6478, 'grad_norm': 4.304238796234131, 'learning_rate': 9.778783308195073e-06, 'epoch': 24.13}

 85%|████████▌ | 17000/19890 [50:47<07:56,  6.06it/s]

{'loss': 1.6384, 'grad_norm': 4.752474308013916, 'learning_rate': 7.264957264957266e-06, 'epoch': 25.64}

 90%|█████████ | 18000/19890 [53:44<05:09,  6.10it/s]

{'loss': 1.6434, 'grad_norm': 3.9599428176879883, 'learning_rate': 4.751131221719457e-06, 'epoch': 27.15}

 96%|█████████▌| 19000/19890 [56:32<02:26,  6.09it/s]

{'loss': 1.6301, 'grad_norm': 2.6730852127075195, 'learning_rate': 2.2373051784816493e-06, 'epoch': 28.66}

100%|██████████| 19890/19890 [58:59<00:00,  5.62it/s]

{'train_runtime': 3539.1924, 'train_samples_per_second': 89.885, 'train_steps_per_second': 5.62, 'train_loss': 1.782144507191899, 'epoch': 30.0}



TrainOutput(global_step=19890, training_loss=1.782144507191899, metrics={'train_runtime': 3539.1924, 'train_samples_per_second': 89.885, 'train_steps_per_second': 5.62, 'total_flos': 1.089370478186496e+16, 'train_loss': 1.782144507191899, 'epoch': 30.0})
```

The training step lasted one hour on the full train dataset (over 10.000 rows) with 30 epochs. This computation is very quick as the base model is small (< 1B parameters), the LoRA config reduced the number of parameters, and the many batches sped up the training.

Results

```
predictions = trainer.predict(test_dataset)

100%|██████████| 85/85 [00:05<00:00, 15.09it/s]
```

Here, we want to see the accuracy of the test dataset for each label.

```
predicted_labels = np.argmax(predictions.predictions[0], axis=1)
predicted_labels = [id2label[pred] for pred in predicted_labels]

test_dataset = test_dataset.add_column("prediction", predicted_labels)
test_dataset = test_dataset.map(lambda x: {"correct": x["hypersense"] == x["prediction"]})
```

```python
pred_df = test_dataset.to_pandas()
print(
    pred_df.groupby("hypersense")["correct"]
    .mean()
    .reset_index()
    .rename(columns={"correct": "accuracy"})
)
```

```
Map: 100%|██████████| 678/678 [00:00<00:00, 2787.45 examples/s]
```

```
                        hypersense  accuracy
0                          animate  0.788991
1                dynamic_situation  0.545455
2    dynamic_situation*information  0.000000
3                        inanimate  0.845794
4            inanimate*information  0.000000
5                      information  0.222222
6                      institution  0.000000
7                            other  0.000000
8                       possession  0.000000
9                         quantity  0.000000
10               stative_situation  0.468085
11                            time  0.000000
```

```python
print(pred_df["hypersense"].value_counts())
```

```
hypersense
animate                          218
inanimate                        214
dynamic_situation                 77
stative_situation                 47
information                       45
other                             26
inanimate*information             17
quantity                          14
time                               7
institution                        7
possession                         3
dynamic_situation*information      3
Name: count, dtype: int64
```

With these results, we can see that the obtained accuracies for the hypersenses depend on the number of examples in the test dataset. "Animate" and "Inanimate" are around 80% accuracy, whereas labels with < 50 samples are close to 0% accuracy. These results can be explained easily, since the labels with < 50 examples are very underrepresented in the train/dev sets. We can therefore conclude that PEFT with LoRA can show good results quickly, if trained with enough data.