

Master in Artificial Intelligence

Advanced Human Language Technologies

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona

FIB

Outline

1 LLM NERC

2 General Structure

3 Detailed Structure

- Few shot learning
- Fine-tuning
- Auxiliary classes

4 Core task

5 Goals & Deliverables

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Task 1.3 - NERC using large language models

Assignment

Write a python program that parses the given XML and recognizes and classifies drug names. The program must use a LLM.

```
$ python3 ./llm-NERC.py devel.xml result.out  
DDI-DrugBank.d278.s0|0-9|Enoxaparin|drug  
DDI-DrugBank.d278.s0|93-108|pharmacokinetics|group  
DDI-DrugBank.d278.s0|113-124|eftifibatide|drug  
DDI-MedLine.d88.s0|15-30|chlordiazepoxide|drug  
DDI-MedLine.d88.s0|33-43|amphetamine|drug  
DDI-MedLine.d88.s0|49-55|cocaine|drug  
DDI-MedLine.d88.s1|82-95|benzodiazepine|drug  
...
```

LLM NERC

General Structure

Detailed Structure

Core task

Goals & Deliverables

Outline

1 LLM NERC

2 General Structure

3 Detailed Structure

- Few shot learning
- Fine-tuning
- Auxiliary classes

4 Core task

5 Goals & Deliverables

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

General Structure

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

- B-I-O schema is no longer valid, since LLMs use sub-word tokenization.
 - Also, LLMs are good with language, but not counting character positions, token spans, offsets, etc.
 - So, we need a task representation suitable for LLMs. They are good handling marking languages (XML, markdown, etc).
For instance, for the sentence:
Antimicrobial activity of Ganoderma lucidum extract alone and in combination with some antibiotics.
- We will ask the LLM to produce the output:

Antimicrobial activity of <drug_n>Ganoderma lucidum extract</drug_n> alone and in combination with some <group>antibiotics</group>.

General Structure

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

- We will need to convert the dataset to/from the pseudo-XML representation.
- We will need to instruct the LLM with a suitable prompt describing the task, followed by each target sentence to annotate.
- We will be using two different strategies:
 - **Few-shot learning:** Add some solved examples in the prompt, so the model understands better what is expected.
 - **Fine-tuning:** Tune the LLM weights using several (dozens, hundreds, ...) solved examples, so it learns the task and does not need the few-show examples each time.

Few-shot learning

- The LLM is asked to perform an specific task in a prompt, which contains:
 - The task description. It must be precise and well explained.
 - Some pairs of (input, expected output) examples.
 - The input for the example to solve
- Adjustable elements:
 - The **prompt**: How the target task is described. Includes specific orders to produce output in certain ways/formats, and NOT to produce undesired output.
 - The **number** of few-shot examples.
 - The **variety** of used few-shot examples (e.g. making sure all drug types appear in some example).

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Fine-Tuning

- The LLM is presented a collection of prompts, and the model weights are iteratively updated. Each prompt includes:
 - The task description.
 - One pair of (input, expected output) example.
- Adjustable elements:
 - The **prompt**: How required task is described. Specific orders to produce output in certain ways, and NOT to produce undesired output. Not as crucial as in few-shot
 - The **amount** of fine tuning examples
 - The **distribution** of used examples (e.g. making sure all drug types are represented enough, maybe balancing too frequent or too scarce classes).

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Outline

1 LLM NERC

2 General Structure

3 Detailed Structure

- Few shot learning
- Fine-tuning
- Auxiliary classes

4 Core task

5 Goals & Deliverables

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Outline

1 LLM NERC

2 General Structure

3 Detailed Structure

- Few shot learning
- Fine-tuning
- Auxiliary classes

4 Core task

5 Goals & Deliverables

LLM NERC

General
Structure

Detailed
Structure

Few shot learning

Core task

Goals &
Deliverables

Few shot - Annotator

LLM NERC
General Structure
Detailed Structure
Few shot learning
Core task
Goals & Deliverables

```
1 # load training data (FS examples)
2 trainfile = os.path.join(paths.DATA,traindata+".xml")
3 fs_examples = Examples(trainfile, "NER").select_examples(num_few_shot)
4 # load prompts, create few-shot prompt
5 prompts = Prompts(promptfile, fs_examples)
6
7 # load test data
8 testfile = os.path.join(paths.DATA,testdata+".xml")
9 test = Examples(testfile, "NER")
10
11 # load model and tokenizer
12 if ollama:
13     engine = Inference(model, ollama=True)
14 else :
15     MODEL_PATH = f"/scratch/nas/1/PDI/mml0/models/{model}"
16     engine = Inference(MODEL_PATH, quantized=quantized)
17
18 # annotate each example in testdata
19 annotated = []
20 for i,ex in enumerate(test.select_examples()):
21     # create prompt for this example, adding it to FS prompt
22     messages = prompts.prepare_messages(ex['input'])
23     # call model to generate response
24     gen_text = engine.generate(messages)
25     # store responses
26     ex['predicted'] = gen_text
27     ex['evaluator'] = test.eval_format(ex,gen_text)
28     annotated.append(ex)
29
30 # print results
31 for e in annotated: print("\n".join(e["evaluator"]))
```

Outline

1 LLM NERC

2 General Structure

3 Detailed Structure

- Few shot learning
- **Fine-tuning**
- Auxiliary classes

4 Core task

5 Goals & Deliverables

LLM NERC

General
Structure

Detailed
Structure

Fine-tuning

Core task

Goals &
Deliverables

Fine tuning - Trainer

```
1 # load prompts
2 prompts = Prompts(promptfile)
3
4 # load model and tokenizer
5 MODEL_PATH = f"/scratch/nas/1/PDI/mml0/models/{model}"
6 engine = FineTuning(MODEL_PATH, quantized=quantized)
7
8 # load and tokenize datasets
9 trainfile = os.path.join(paths.DATA,traindata+".xml")
10 train_examples = Examples(trainfile, "NER").select_examples()
11 train_dataset = engine.tokenize_dataset(train_examples, prompts)
12
13 valfile = os.path.join(paths.DATA,valdata+".xml")
14 val_examples = Examples(valfile, "NER").select_examples()
15 val_dataset = engine.tokenize_dataset(val_examples, prompts)
16
17 # Fine-tune the model and save results
18 engine.train(train_dataset,
19               val_dataset,
20               outputdir)
```

LLM NERC

General
Structure

Detailed
Structure

Fine-tuning

Core task

Goals &
Deliverables

Fine tuning - Annotator

LLM NERC

General
Structure

Detailed
Structure

Fine-tuning

Core task

Goals &
Deliverables

```
1 # load prompts
2 prompts = Prompts(promptfile)
3
4 # load test/devel dataset
5 testfile = os.path.join(paths.DATA,testdata+".xml")
6 test = Examples(testfile, "NER")
7
8 # load model and tokenizer
9 MODEL_PATH = f"/scratch/nas/1/PDI/mml0/models/{model}"
10 engine = Inference(MODEL_PATH, quantized=quantized, peft=weightdir)
11
12 # analyze each example
13 annotated = []
14 for i,ex in enumerate(test.select_examples()):
15     # prepare sequence of messages for this example
16     messages = prompts.prepare_messages(ex['input'])
17     # call model to generate response
18     gen_text = engine.generate(messages)
19     # extract json from response
20     ex["predicted"] = gen_text
21     ex['evaluator'] = test.eval_format(ex,gen_text)
22     annotated.append(ex)
23
24 # save output
25 for e in annotated: print("\n".join(e["evaluator"]))
```

Outline

1 LLM NERC

2 General Structure

3 Detailed Structure

- Few shot learning
- Fine-tuning
- Auxiliary classes

4 Core task

5 Goals & Deliverables

LLM NERC

General
Structure

Detailed
Structure

Auxiliary classes

Core task

Goals &
Deliverables

Auxiliary classes - Examples

```
1 class Examples:  
2     ## constructor: parses data XML file, and converts sentences into  
3     ## the format expected by the LLM for the given task (NER/DDI).  
4     def __init__(self, xmlfile, task)  
5  
6     ## get 'numFS' examples (or all dataset if omitted)  
7     ## balance classes if requested  
8     def select_examples(self, numFS=-1, balanced=False)  
9  
10    ## convert LLM output into the format expected by the  
11    ## evaluator for the task this instance is loaded for  
12    def eval_format(self, ex, text)
```

- Class `Examples` takes care of loading the dataset XML files and converting the examples into the appropriate representation for the LLM.
- Class `Examples` also converts LLM responses into the output format expected by the evaluator.
- For NERC, `balanced` parameter is not suitable, since the output is not a single class per sentence.
- Method `select_examples` can be replaced with a custom selection strategy.

LLM NERC

General Structure

Detailed Structure

Auxiliary classes

Core task

Goals & Deliverables

Auxiliary classes - Prompts

```
1 class Prompts:  
2     # Constructor: loads prompts from json file (sysprompt and usrprompt)  
3     # If a list of fs_examples is provided, they are added to the prompt  
4     def __init__(self, promptfile, fs_examples=[])  
5  
6         # Build prompt for a particular example, concatenating the  
7         # base prompt (sysprompt + usrprompt + fs_xamples --if any)  
8         # with the given user request 'question', plus the expected  
9         # answer --if provided  
10        def prepare_messages(self, question, answer="")
```

- The constructor stores a prompt that contains the system prompt and the user prompt. For few-shot, it also contains the provided solved examples.
- Method `prepare_messages` is used in few-shot to add a request (question) to the prompt (which already has some solved examples), without the corresponding answer.
- Method `prepare_messages` is used in fine-tuning training to add a complete solved example (request plus expected answer). In this scenario, the base prompt does not contain few-shot examples.

LLM NERC

General Structure

Detailed Structure

Auxiliary classes

Core task

Goals & Deliverables

Auxiliary classes - model.Inference

```
1 class Inference:  
2     # load given model in inference mode  
3     def __init__(self, model_path, quantized=False, peft=None, ollama=False)  
4  
5     # Generate completion for given messages  
6     def generate(self, messages)  
7
```

- Constructor loads requested model (either using ollama or HF transformers) in inference mode. The model can be loaded in full, or quantized to 4 bits. If `peft` is given it is expected to be the path to fine-tuned LoRa adapters for the given model, which are also loaded.
- Method `generate` uses the loaded LLM to generate a completion for given prompt (typically, the answer to the last request in it).

LLM NERC
General Structure

Detailed Structure
Auxiliary classes

Core task

Goals & Deliverables

Auxiliary classes - model.FineTuning

```
1 class Inference:  
2     # load given model in LoRa fine tuning mode.  
3     def __init__(self, model_path, quantized=False)  
4  
5         # Tokenizes given dataset with the model appropriate tokenizer.  
6         # Returns a "datasets.Dataset" object suitable for HF trainer  
7         def tokenize_dataset(self, dataset, prompts)  
8  
9             # Fine tunes model with given train and validation data.  
10            # Saves tuned weights in outputdir  
11            def train(self, train_dataset, val_dataset, outputdir)  
12
```

- The constructor loads requested model (quantized if required) and adds LoRa adapters to be tuned.
- Method `tokenize_datasets` uses model tokenizer to preprocess the training/validation data and convert them into batched `datasets.Dataset`, suitable for the trainer.
- Method `train` performs the fine-tuning using training data and computes loss on validation data at each epoch.
- Method `train` can be adjusted to change learning parameters if needed.

LLM NERC

General Structure

Detailed Structure

Auxiliary classes

Core task

Goals & Deliverables

Outline

1 LLM NERC

2 General Structure

3 Detailed Structure

- Few shot learning
- Fine-tuning
- Auxiliary classes

4 Core task

5 Goals & Deliverables

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Build a good LLM-based drug NERC

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Strategy: Experiment with different **few-shot** configurations:

- Prompts:

- Explain better/more clearly the task.
- If the model produces wrong output, insist on what is the expected output or its right format.
- If the model misses some expected output, insist on its importance.
- If the model produces unrequested output, explicitly request it not to do so.

- Few-shot examples

- Experiment with different number of few-shot examples.
- Experiment with better example selection methods (even hand-picking most informative examples from training set). Modify `select_examples` method to achieve this.

- Model

- Use different models (llama, qwen, gemma, mistral...).
- Experiment with/without quantization

Build a good LLM-based drug NERC

Strategy: Experiment with different **fine-tuning** configurations:

- Prompts:
 - Improve prompts as in few-shot (although it has much less effect here).
- Training examples
 - Experiment with different number of training examples.
 - Experiment with better example selection methods.
Modify `select_examples` method to achieve this.
- Model
 - Use different models (llama, qwen, gemma, mistral...) and sizes.
 - Experiment with/without quantization
- Learning hyperparameters
 - Learning rate
 - Epochs
 - batch size (or gradient accumulation steps)

LLM NERC
General Structure
Detailed Structure
Core task
Goals & Deliverables

Build a good LLM-based drug NERC

Few-shot caveats:

- **Model context size:** Too many few-shot examples might fill the model context window and make it forget the instructions.
- **Available GPU RAM:** Too many few-shot examples, even if they don't fill the context window, may not fit into GPU RAM.
- **Token offset errors:** The model might slightly alter the text, which will result in wrong offsets when converting the XML tags to the format expected by the evaluator.

Fine-tuning caveats:

- **Available GPU RAM:** Fine tuning a model takes about 3 times the RAM used in inference. Bigger GPUs are required. Smaller batch sizes may help. Quantization too.
- **Available time:** Fine tuning takes a lot of time, since the dataset needs to be run through the model, loses computed, weights updated. For each example, several epochs.

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Outline

1 LLM NERC

2 General Structure

3 Detailed Structure

- Few shot learning
- Fine-tuning
- Auxiliary classes

4 Core task

5 Goals & Deliverables

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Exercise Goals

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

What you should do:

- Experiment with few-shot and fine-tuning variations.
- Experiment with different prompts and few-shot examples.
- Experiment with different prompts and training examples.
- Experiment with different learning hyperparameters.
- Keep track of tried variants and parameter combinations.

What you should **NOT** do:

- Get insights about errors from `devel` dataset. You have `train` for that. `devel` is only used to evaluate the performance of a given configuration.
- Select architectures or hyperparameters based on system performance on `test` dataset. You have `devel` for that. `test` is only used to evaluate model generalization ability once the best configuration has been chosen.

Deliverables

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

At the end of the NERC task, you will need to deliver a single report on the work carried out on NERC-ML, NERC-NN, and NERC-LLM systems

So, during the development and experimentation on NERC-LLM:

- keep track of tried/discard architectures/hyperparameters
- keep track of tried/discard input information.
- Record obtained results in the different experiments, and compile the information you'll later need to elaborate the report.

Annex: Resources to use LLMs. Ollama

ollama can be used for few-shot experiments.

- Ollama runs LLMs with or without GPU (much slower without)
- Install ollama: <https://ollama.com/download>
- Install ollama python API: pip install ollama
- Download desired ollama models
 - ollama pull llama3.2:3b
 - ollama pull qwen3:8b
 - ollama pull ministral3:8b
- See <https://ollama.com/library> for a list
- Use provided `fewshot.py` with `-ollama` option.

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Annex: Resources to use LLMs. Huggingface

Huggingface Transformers can be used for fine-tuning experiments.

- If you have a GPU (with at least 8-10GB), you can install requirements.txt in a python virtual environment, and run the programs out-of-the box.
- Use HF model id (e.g. `meta-llama/Llama-3.2-3B-Instruct`) as MODEL_PATH when loading the model.
- If you don't have a GPU, you can use [google colab](#), or UPC [boada](#) cluster, where llama3.2-3B and qwen3-3B are already installed.

LLM NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables