**CA#6 Part 2**

**Natural Language Understanding**

**Abstraction**

**Intent detection** and **slot filling** are the main tasks to solve when approaching the  
problem of **Natural Language Understanding (NLU)** in a **conversational system**. The two tasks are used to obtain a structured representation of the meaning of the utterance, so that it can be processed by a computer. Intent detection deals with identifying the **overall meaning** of the sentence. It is modeled as a **classification** problem, in which we receive an input utterance and we have to classify it as having one intent from a group of known intents. The available intents correspond to the actions that the conversational model can perform, such as adjusting the temperature, controlling the media center or turning the lights on/off in the case of a home assistant. On the other hand, slot filling is modeled as a **sequence labelling problem**, whose purpose is to take the utterance and determine which words **indicate relevant information for the intent**. These slots contain supplementary information about the action and correspond to the parameters of the action.

The task-oriented dialogue system is the basis of virtual assistants like Alexa, Siri, Cortana, and Portal has been increasingly used in modern society; users interact with them across different domains to complete diverse tasks and achieve their specific goals. Key component of these task-oriented dialogue systems is Natural Language Understanding (NLU) which aims to derive the intent of users and fill the value for the slots of the utterance.

**Dataset**

**MASSIVE** is a parallel dataset of > 1M utterances across **51 languages** with annotations for the Natural Language Understanding tasks of intent prediction and slot annotation. Utterances span **60 intents** and include **55 slot** types. MASSIVE was created by localizing the [SLURP](https://github.com/pswietojanski/slurp) dataset, composed of general Intelligent Voice Assistant single-shot interactions.

In this assignment, we are to use Farsi Dataset (fa-IR.jsonl). This Json file contains train, dev and test sets. Now will study the details of this dataset:

Text

Description automatically generated

Figure 1 Input Jsonl file format details

In this assignment the “judgments” and “worker\_id” are not considered.

There are 60 different intents and 56 different slots in this dataset.

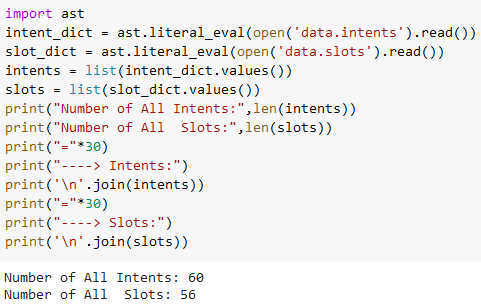


Figure 2 Number of slots and intents

Now we take a look at intents list:

----> Intents:

recommendation\_locations

play\_music

iot\_cleaning

email\_addcontact

datetime\_convert

transport\_ticket

qa\_stock

lists\_query

email\_query

datetime\_query

calendar\_remove

iot\_wemo\_off

recommendation\_events

email\_sendemail

qa\_maths

general\_quirky

calendar\_query

iot\_hue\_lighton

audio\_volume\_other

takeaway\_order

transport\_query

weather\_query

alarm\_set

qa\_factoid

play\_radio

lists\_remove

qa\_currency

news\_query

lists\_createoradd

general\_greet

social\_query

iot\_hue\_lightchange

iot\_hue\_lightdim

calendar\_set

iot\_hue\_lightup

recommendation\_movies

play\_audiobook

alarm\_query

audio\_volume\_up

cooking\_recipe

iot\_wemo\_on

social\_post

qa\_definition

audio\_volume\_mute

general\_joke

iot\_hue\_lightoff

music\_dislikeness

transport\_traffic

takeaway\_query

play\_podcasts

iot\_coffee

audio\_volume\_down

play\_game

transport\_taxi

email\_querycontact

music\_query

cooking\_query

music\_likeness

music\_settings

alarm\_remove

Now we take a look at slots list:

----> Slots:

transport\_agency

playlist\_name

house\_place

media\_type

time\_zone

time

device\_type

business\_name

music\_album

artist\_name

podcast\_descriptor

personal\_info

email\_folder

news\_topic

order\_type

podcast\_name

food\_type

transport\_type

game\_type

general\_frequency

list\_name

app\_name

audiobook\_name

sport\_type

alarm\_type

song\_name

place\_name

game\_name

music\_genre

person

change\_amount

date

ingredient

radio\_name

email\_address

meal\_type

movie\_name

definition\_word

transport\_name

coffee\_type

relation

event\_name

currency\_name

business\_type

music\_descriptor

weather\_descriptor

timeofday

movie\_type

transport\_descriptor

joke\_type

Other

color\_type

player\_setting

audiobook\_author

cooking\_type

drink\_type

Here we can compare frequencies of **domains**, **intents** and **slots** count in each partision:

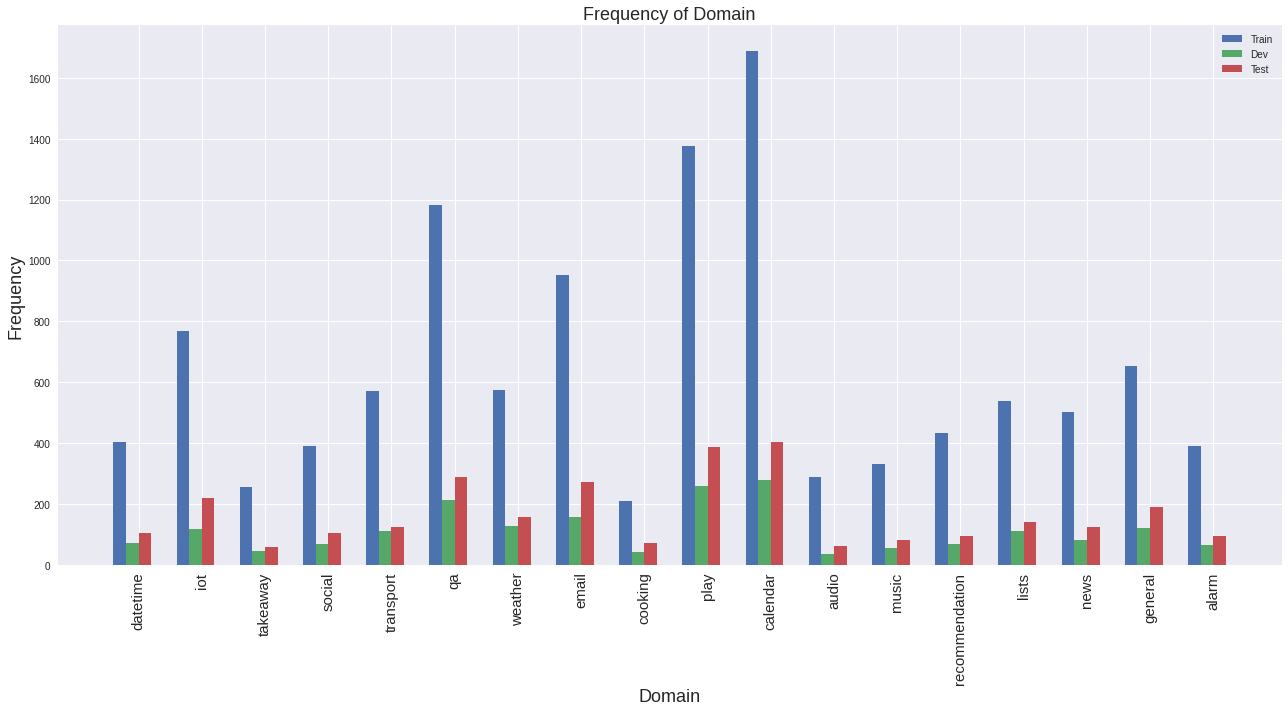


Figure 3 Domain Frequency Bar Plot

As we can see above, the **calendar** domain is the most frequent domain.

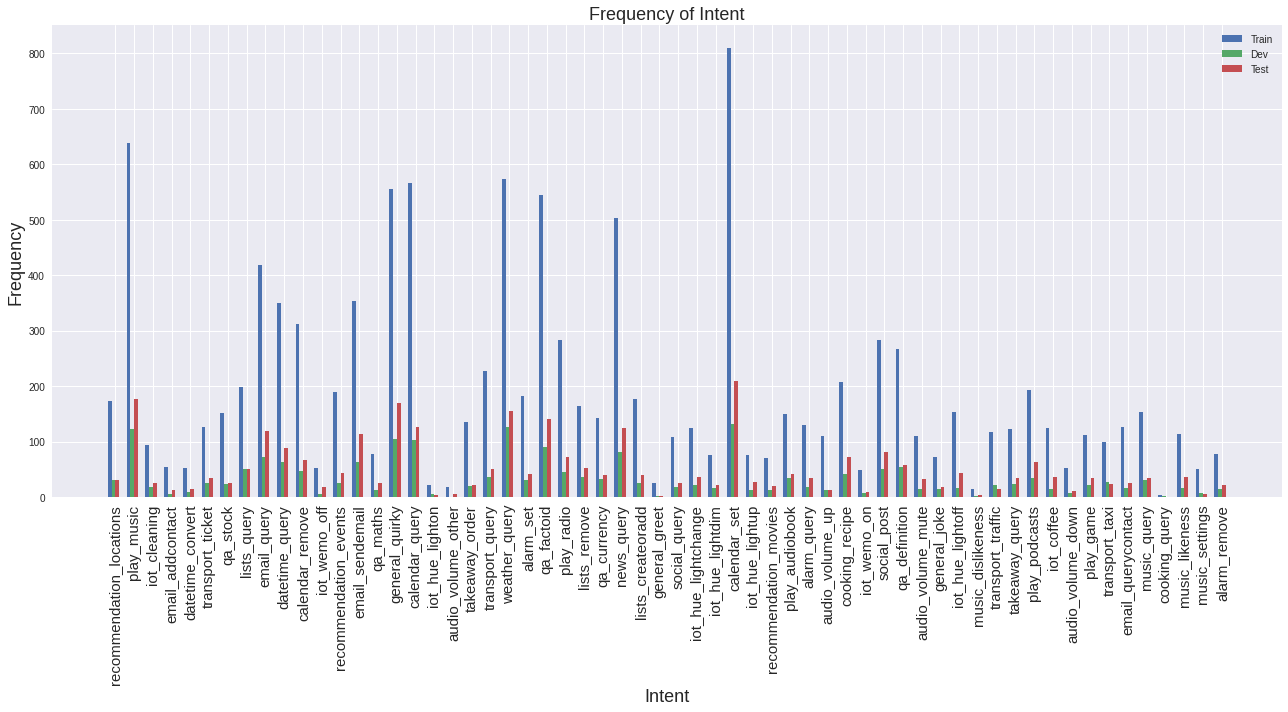


Figure 4 Intent Frequency Bar Plot

As we can see above, the **calendar\_set** intent is the most frequent and cooking\_query is the least frequent intent.

Chart

Description automatically generated

Figure 5 Slot Frequency Bar Plot

As we can see above, the **date** slot is the most frequent slot.

Here are some more details of counts:

|  |  |  |
| --- | --- | --- |
| Figure 6 Domain frequencies | Figure 7 Intent frequencies | Figure 8 slot frequencies |

**Preparing Dataset**

I put the fa-IR-jsonl file in “Dataset” folder and passed it to create\_hf\_dataset.py.

****

Figure 9 Creating Dataset

The create\_hf\_dataset.py python file has a class which prepare the dataset and parse Jason file. This class if for creating four dataset splits, in the Huggingface Datasets Apache Arrow format from the MASSIVE dataset.

Each dataset split has the following **columns**:

"id", "locale", "utt", "annot\_utt", "domain", "intent\_str", "intent\_num", "slots\_str", "slots\_num"

**Methods:**

* create\_datasets(data\_path): Creates the dataset splits using the data\_path of the MASSIVE set
* add\_numeric\_labels(): Create integer versions of intents and slot for modeling
* investigate\_datasets(): Prints out the seventh example from each dataset split as gut check
* save\_label\_dicts(prefix): Saves the mappings to the integer versions of the labels
* save\_datasets(out\_prefix): Saves the datasets to out\_prefix

In this implementation each intent and slot is mapped to a number in order to make it possible to process and use them in training.

**Training**

I cloned the MASSIVE github to get the needed python files.

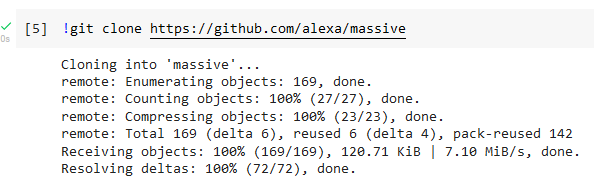


Figure 10 Cloning Massive

Here I used the xlm-roberta-base model for training. For training we use the following script and pass a config file to set the parameters of the model. This file is included in the folder uploaded named “train\_config.yml”. Overally, in train.py the following procedure flows:

* parsing the args
* creating the massive.configuration master config object
* Seting up logging
* Getting all inputs to the trainer
* Getting the right trainer

As I mentioned before, we need to pass a configuration file. We need to especify **pretrained\_weights** and **vocab\_file** in this file. For these to fields I used pytorch\_model.bin and sentencepiece.bpe.model in xlm-roberta-base of huggingface:



Figure 11 Getting xlm-roberta-base files

I trained the model with batch\_size of 128 and 45 epochs. Configuration file can be fully observed in the uploaded file but I will explain some of the parameters here.

The model and tokenizer parameters are:

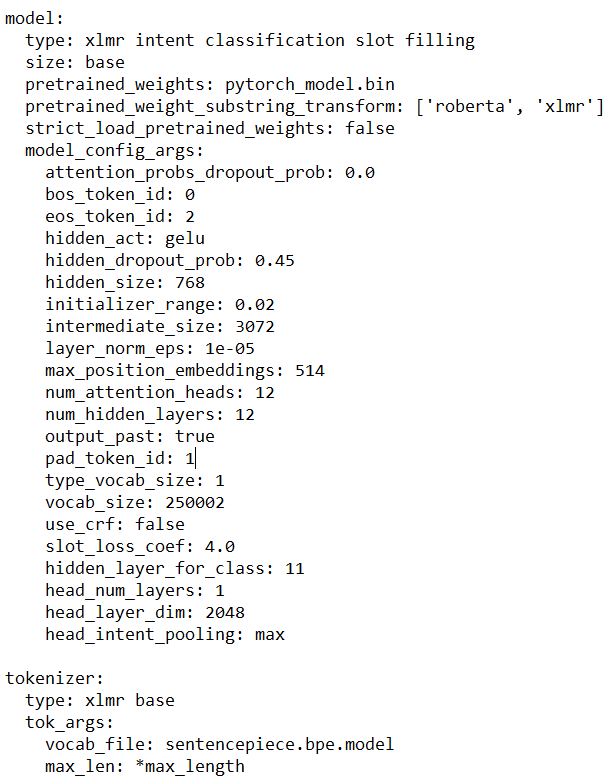


Figure 12 Parameters

As we can see we use the pytorch\_model.bin for pretrained\_weights and sentencepiece.bpe.model for vocab\_file in xlm-roberta-base. The head layer dimension is 2048 and and number of hidden layers are 12. By setting these parameters we get the following architecture:

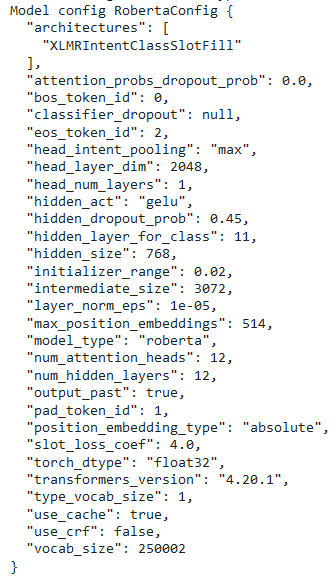


Figure 13 Model architecture details

Train and validation parameters are:

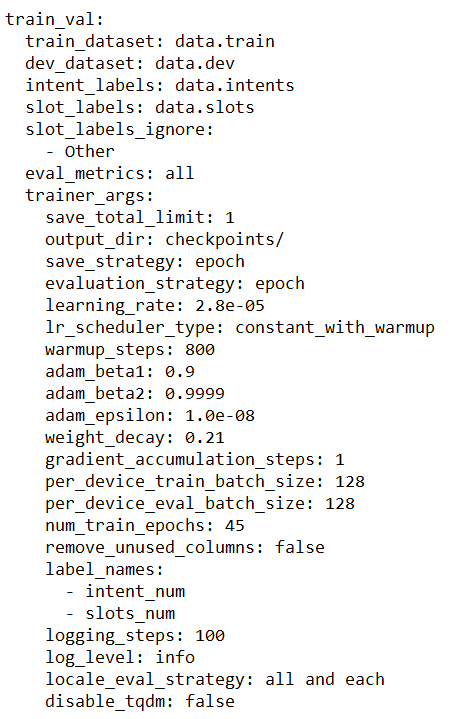


Figure 14 Parameters

As we can see the batch size is 128, learning rate is initially set to 0.000028. we used the adam optimizer and the epochs are 45. Pay attention since in this implementation there is a “other” slot which corresponds to no slots defined we ignore it.

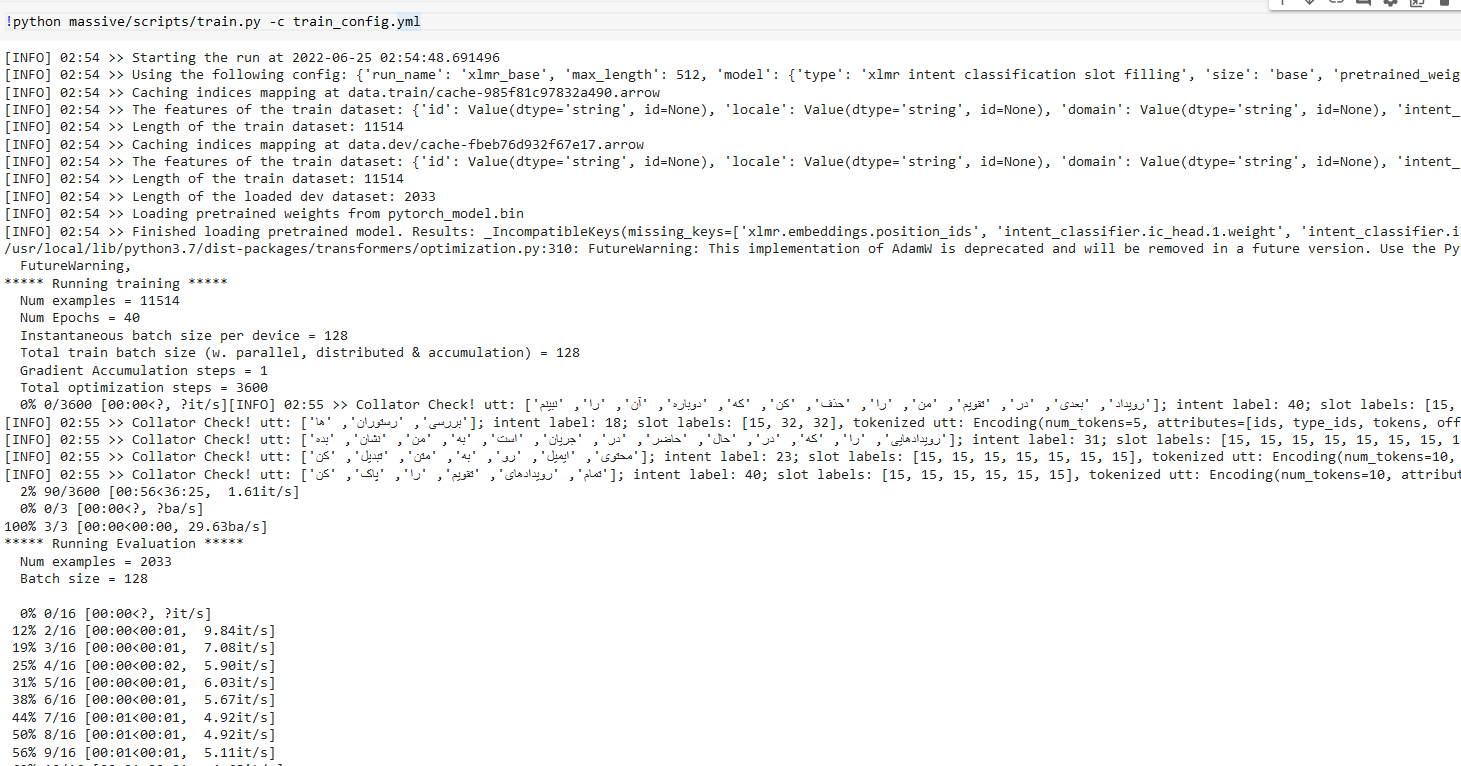


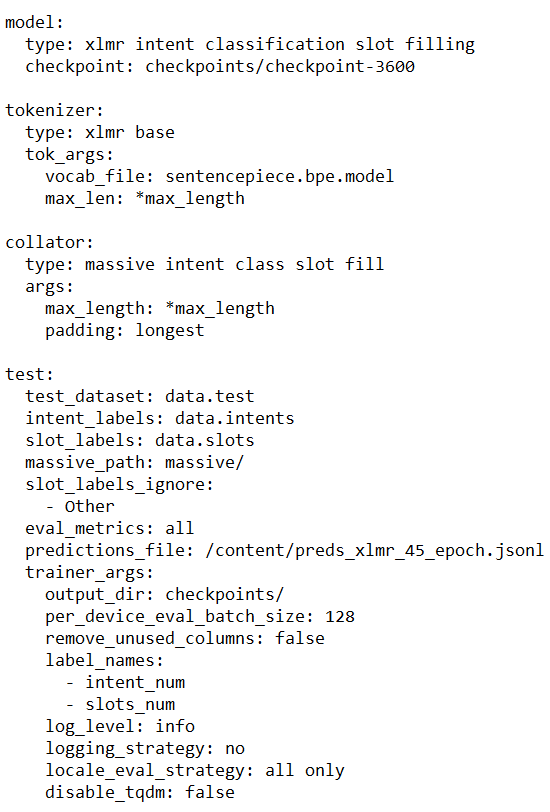
Figure 15 Training

After training by 45 epoches this is the **results**:

{'training\_global\_step': 3600, **'training\_epoch': 45.0**, 'eval\_fa-IR\_loss': 1.8478409051895142, **'eval\_fa-IR\_intent\_acc': 0.8494835218888342,** 'eval\_fa-IR\_intent\_acc\_stderr': 0.007930504466678288, **'eval\_fa-IR\_slot\_micro\_f1': 0.7392102846648301,** 'eval\_fa-IR\_slot\_micro\_f1\_stderr': 0.0015396802730499027, 'eval\_fa-IR\_ex\_match\_acc': 0.6148548942449582, 'eval\_fa-IR\_ex\_match\_acc\_stderr': 0.010792692898620112, 'eval\_fa-IR\_runtime': 3.6738, 'eval\_fa-IR\_samples\_per\_second': 553.373, 'eval\_fa-IR\_steps\_per\_second': 4.355, **'eval\_all\_loss': 1.8478409051895142**, **'eval\_all\_intent\_acc': 0.8494835218888342**, 'eval\_all\_intent\_acc\_stderr': 0.007930504466678288, **'eval\_all\_slot\_micro\_f1': 0.7392102846648301**, 'eval\_all\_slot\_micro\_f1\_stderr': 0.0015396802730499027, **'eval\_all\_ex\_match\_acc': 0.6148548942449582**, 'eval\_all\_ex\_match\_acc\_stderr': 0.010792692898620112, 'eval\_all\_runtime': 3.669, 'eval\_all\_samples\_per\_second': 554.109, 'eval\_all\_steps\_per\_second': 4.361, 'eval\_highest-locale\_intent\_acc': 'fa-IR', 'eval\_highest-locale-val\_intent\_acc': 0.8494835218888342, 'eval\_lowest-locale\_intent\_acc': 'fa-IR', 'eval\_lowest-locale-val\_intent\_acc': 0.8494835218888342, 'eval\_highest-locale\_slot\_micro\_f1': 'fa-IR', 'eval\_highest-locale-val\_slot\_micro\_f1': 0.7392102846648301, 'eval\_lowest-locale\_slot\_micro\_f1': 'fa-IR', 'eval\_lowest-locale-val\_slot\_micro\_f1': 0.7392102846648301, 'eval\_highest-locale\_ex\_match\_acc': 'fa-IR', 'eval\_highest-locale-val\_ex\_match\_acc': 0.6148548942449582, 'eval\_lowest-locale\_ex\_match\_acc': 'fa-IR', 'eval\_lowest-locale-val\_ex\_match\_acc': 0.6148548942449582, 'epoch': 45.0}

**Testing**

For testing we use the following script and pass a config file to set the parameters of the model. This file is included in the folder uploaded named “test\_config.yml”. These parameters are as follows:



The batch size is 128 and I save the predictions in preds\_xlmr\_45\_epoch.jsonl file.

****

Figure 16 Testing

**Results** of testing are:

{ **'test\_all\_ex\_match\_acc': 0.6146603900470746,**

'test\_all\_ex\_match\_acc\_stderr': 0.008924193086447684,

**'test\_all\_intent\_acc': 0.8490248823133827,**

'test\_all\_intent\_acc\_stderr': 0.006565116171480395,

'test\_all\_loss': 1.9511113166809082,

'test\_all\_runtime': 6.1897,

'test\_all\_samples\_per\_second': 480.475,

**'test\_all\_slot\_micro\_f1': 0.7281713344316308,**

'test\_all\_slot\_micro\_f1\_stderr': 0.0009614521731479325,

'test\_all\_steps\_per\_second': 3.877

}

As we can see the results of the model are very good and close to the accuracies mentioned in the [Massive paper.](https://arxiv.org/pdf/2204.08582.pdf)

The results reported in the paper are:

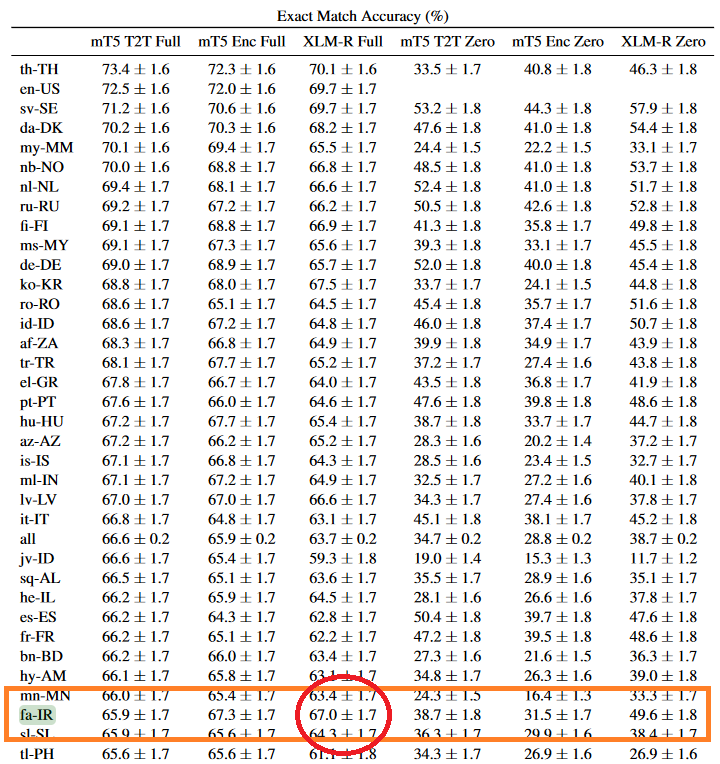


Figure 17 Exact Match Accuracy in the paper

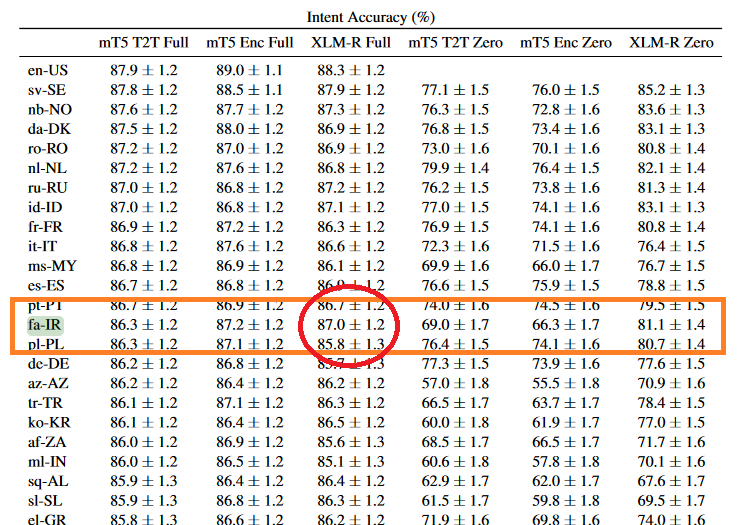


Figure 18 Intent Accuracy in the paper

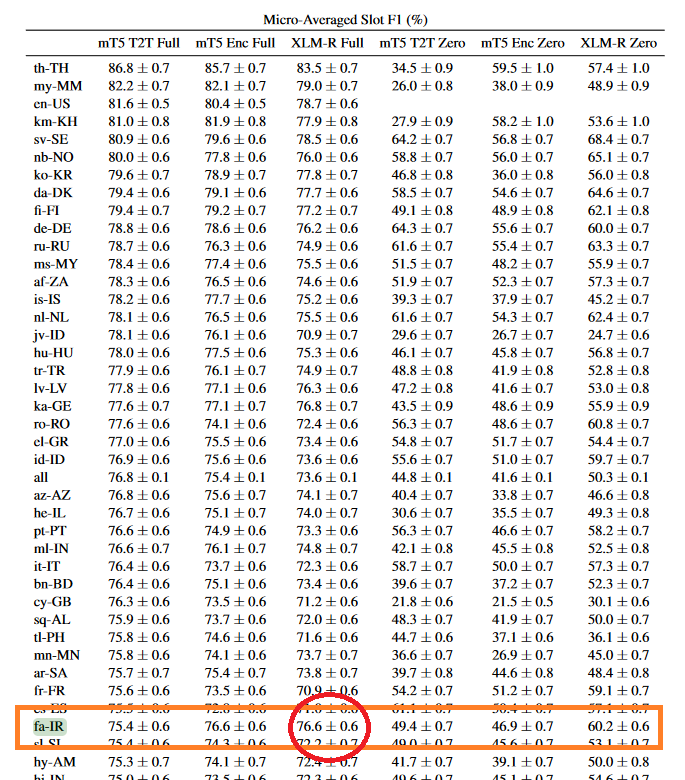


Figure 19 Micro-Averaged Slot F1 in the paper

Now, lets look at some prediction by more details:



Figure 20 Id 6725 - Actual

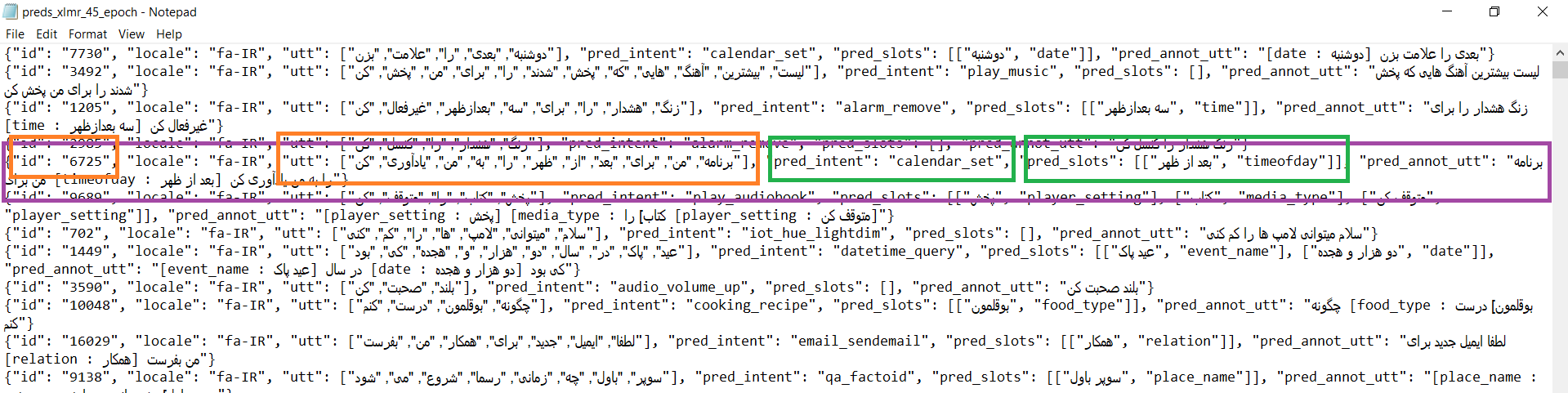


Figure 21 Id 6725 - Predicted

As we can see the prediction is correct an this confirms the results scores I showed above.

I also trained the model on mt5\_enc\_base. The results are:

'test\_all\_ex\_match\_acc': 0.594371217215870881,

'test\_all\_ex\_match\_acc\_stderr': 0.0012097048826613399,

'test\_all\_intent\_acc': 0.699394754539341,

'test\_all\_intent\_acc\_stderr': 0.008407927933758254,

'test\_all\_samples\_per\_second': 518.046,

'test\_all\_slot\_micro\_f1': 0.70055148853099072,

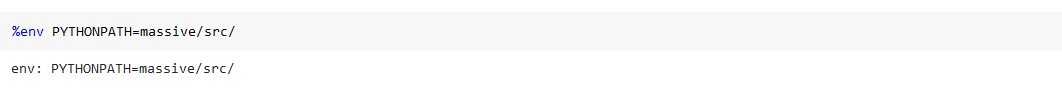
'test\_all\_steps\_per\_second': 8.187

}

Which we can see the results with xlm-roberta-base were better.

**Some problems I encountered during completing this part:**

Since creatigng the environment given in massive github was not easy I tried to run the codes without that environment. So I installed packages and dependencies manually and by a req.txt file wich is included in the folder uploaded. Without the given environment I got some errors during running the train.py and test.py so, I needded to change some parts of the code especially the importing parts. After trying to handle these errors I realized by using this command all the erros will be gone.



Another problem was the disk capacity of google colab which leaded to errors. In order to handle this I used an filed named save\_total\_limit = 1 in the training config file which will save just the last checkpoint not all the checkpoints.

\*\*\* for running the note book file make sure you have the “Dataset” folder which included the “fa-IR.jsonl” file, the req.txt file and train\_config.yml and test\_config.yml.

**References:**

https://arxiv.org/pdf/2204.08582.pdf

<https://arxiv.org/pdf/2010.11934.pdf>

<https://arxiv.org/abs/1911.02116>

<https://mdpi-res.com/d_attachment/sensors/sensors-21-01230/article_deploy/sensors-21-01230-v3.pdf?version=1614215625>