National College of Ireland

Configuration Manual

MSc Research Project
MSc in Artificial Intelligence

Muhammet Gumus
Student ID: 23330562

School of Computing
National College of Ireland

Supervisor: Prof. Paul Stynes

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Muhammet Gumus
Student ID:	23330562
Programme:	MSc in Artificial Intelligence **Year:** 2024/2025
Module:	Practicum Part 2
Lecturer:	Prof. Paul Stynes
Submission Due Date:	11/08/2025
Project Title:	Transformer Framework for Language Translation in Low Resource Languages
Word Count:	833 **Page Count:** 7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Muhammet Gumus

Date: 10/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Muhammet Gumus
Student ID: 23330562

1 System Requirements and Environment Setup

This project was developed and executed using Python 3.10 with all code implemented in Jupyter Notebook environments on Google Colab Pro Platform totally. The primary IDEs used were Google Colab and Visual Studio Code on a MacOS system. The local development were used extremely less because of the easiness and powerfulness of Google Colab and Google Drive provides. Additionally using the GPU for the training part make that to be chosen that environment first. In addition to them, to enable the execution of large language models and training processes, a GPU runtime was used on Google Colab with NVIDIA T4 or A100. Due to GPU constraints and cost limitations, fine-tuning was restricted to smaller transformer models (e.g. mT5-small, mBART). All experiments requiring high compute resources in this study were executed in Colab Pro account, which offered better GPU availability than the regular free version.

Required System Setup:
Operating System: MacOS / Windows
Python Version: 3.10+
Google Colab Pro - (Mainly used in the development and preffered for GPU usage)
Jupyter Notebook
Minimum 16 GB RAM recommended for local testing / (Time to time High Ram Option selected on the Colab)

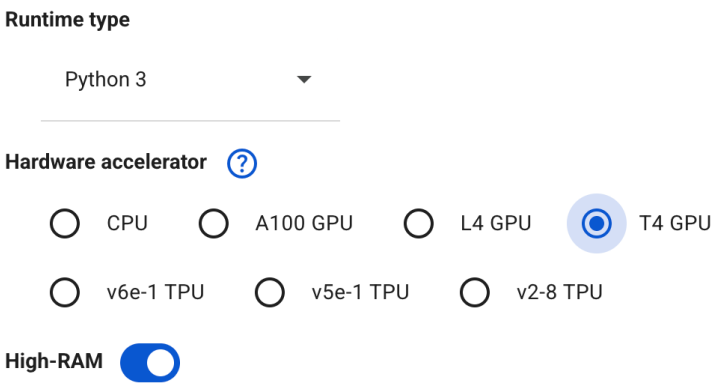

Figure 1. Runtime Type

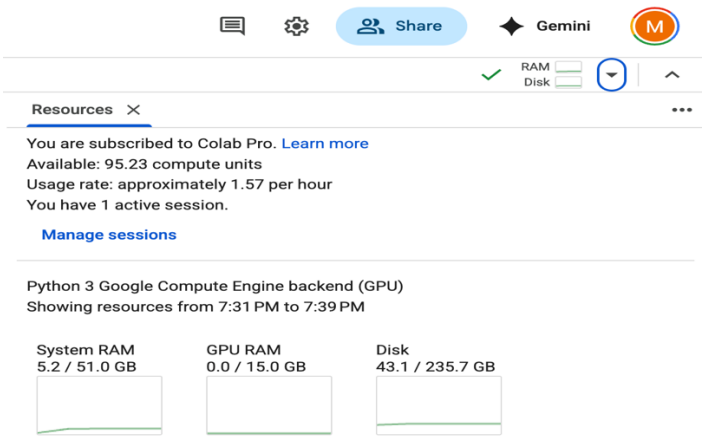

Figure 2. System Specifications

2 Data and Folder Specifications

The datasets used in this study consist of parallel sentence pairs stored in separate files for each language. In this study English and Turkish sentences are stored in files with the extensions ".en" and ".tr" respectively. In here, each line representing a matching translation pair. It is really critical for data consistency that the number of lines in both files must be equal.

The working environment was firstly set up by mounting Google Drive to the system. The projects main directory is located at /content/drive/MyDrive/MT_Project and the original data files are stored in the "/content/drive/MyDrive/MT_Project/data" directory. During the file processing phase, sentence pairs extracted from these original data are selected based on specific criteria and the processed files are created in the /content/drive/MyDrive/MT_Project/data-test directory. Also under the data folder each dataset has its own subfolders. Datasets are publicly available at OPUS-100 , Tatoeba , TED2020 . They can be downloaded and then used under the folders like the following images.

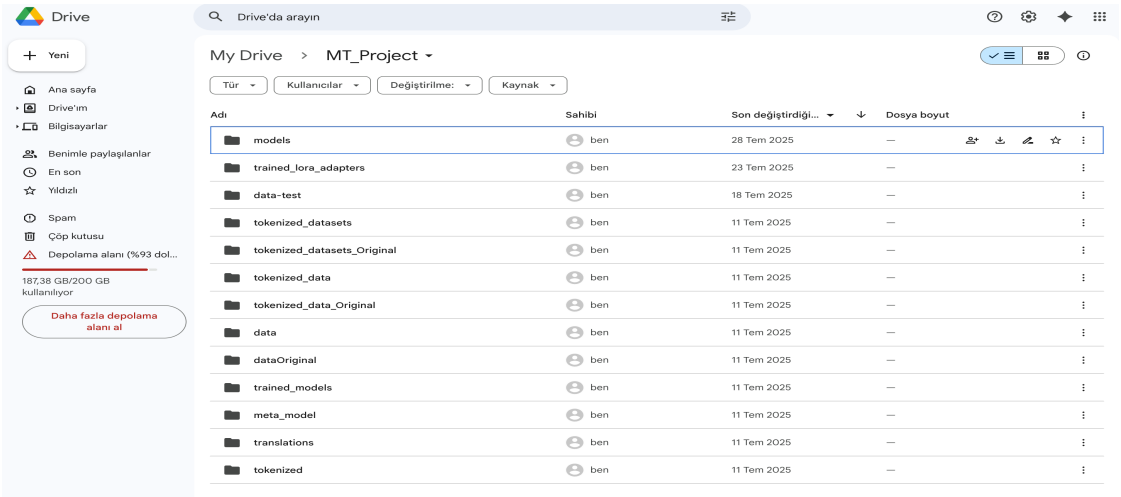

Figure 3. Base Project Folder

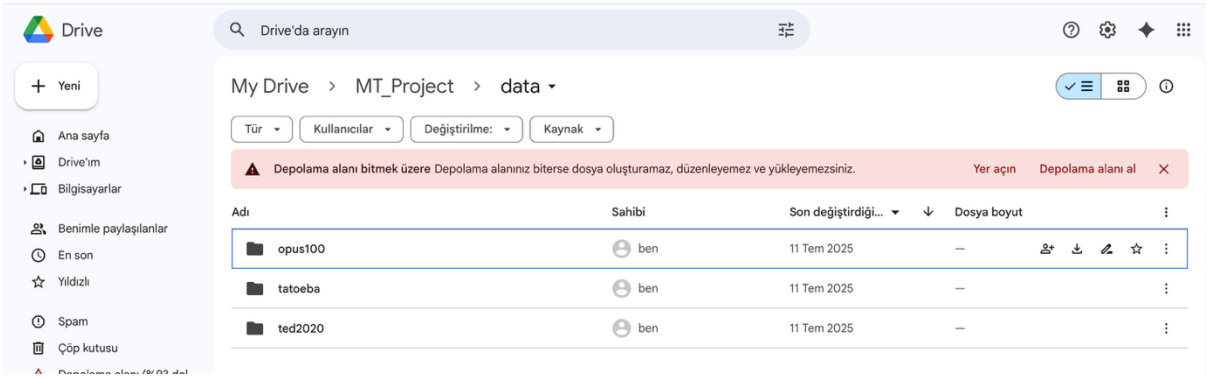

Figure 4. Base Data Folder

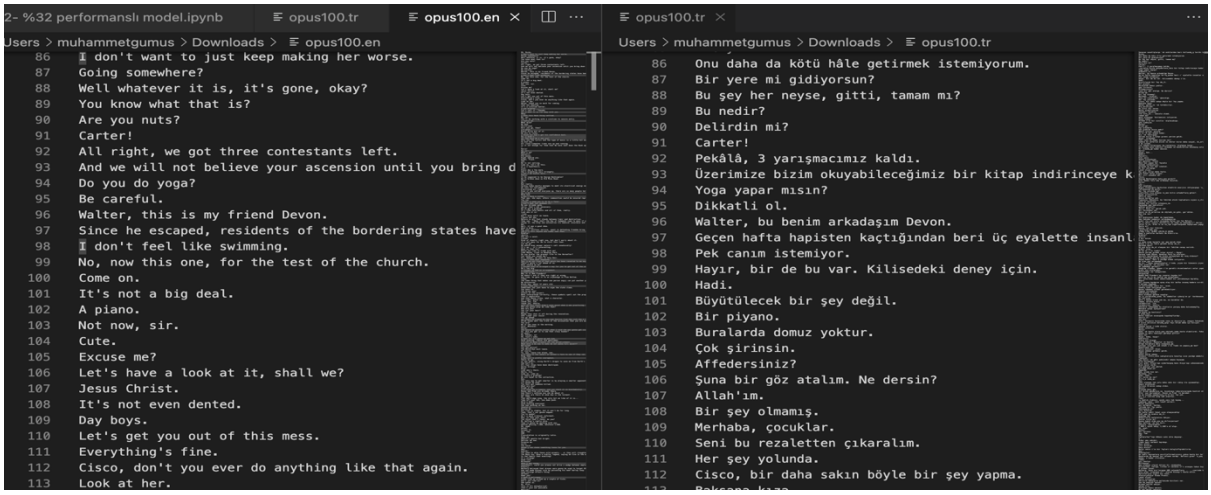

Figure 5. Dataset File Examples

3 Project Development

After collecting and putting the data under the suitable folders then Colab Notebook can be executed. All codes are stored in .ipynb file with the previous code outputs and results. To be able to run the code without any problem at this point, firstly data should be under the correct folders. Then, the required packages and libraries needs to be installed on the code environment. After that Google Drive must be mounted.

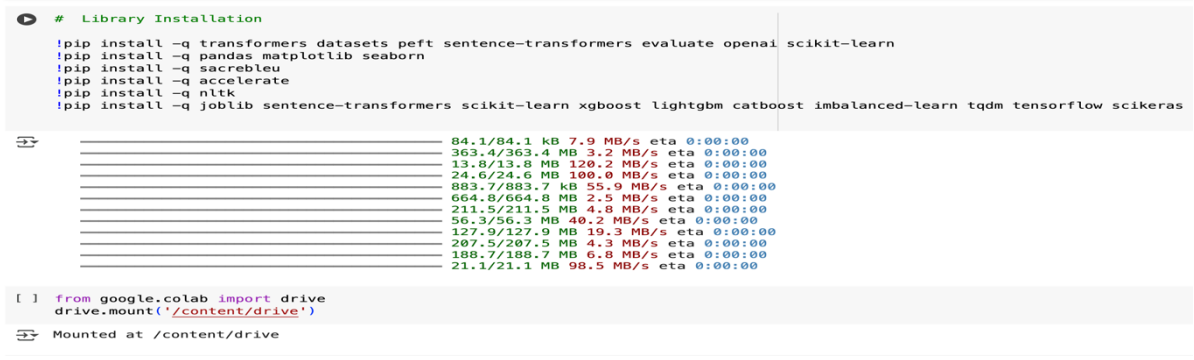

Figure 6. Library Installation and Drive Mount

As a second main structural thing, is can be the folders creation after installing the packages and libraries that needs to be required. In here, users do not need to create necessary folders by themselves manually because they are created by python scripts automatically.

```python
#  Folder Structure Creation

import os

BASE_PATH = "/content/drive/MyDrive/MT_Project/"
DATA_PATH = f"{BASE_PATH}/data"
TOKENIZED_PATH = f"{BASE_PATH}/tokenized_datasets"
MODEL_SAVE_PATH = f"{BASE_PATH}/trained_models"


TOKENIZED_DATASETS_DIR = os.path.join(BASE_PATH, "tokenized")
TRANSLATION_OUTPUTS_DIR = os.path.join(BASE_PATH, "translations")
META_MODEL_DIR = os.path.join(BASE_PATH, "meta_model")
DATASETS_DIR = os.path.join(BASE_PATH, "data")


os.makedirs(TOKENIZED_DATASETS_DIR, exist_ok=True)
os.makedirs(TRANSLATION_OUTPUTS_DIR, exist_ok=True)
os.makedirs(META_MODEL_DIR, exist_ok=True)
os.makedirs(DATASETS_DIR, exist_ok=True)
```

Figure 7. Folder Structure Creation

After creation the folders we can start the dataset loading and start to process it by the next code block. Mainly the datasets notified and according to their paths they can be readable and processable.

```python
### DATASET LOADING
dataset_names = ["ted2020", "tatoeba", "opus100"]
dataset_dicts = {}

for name in dataset_names:
    en_path = f"{BASE_PATH}data/{name}/{name}.en"
    tr_path = f"{BASE_PATH}data/{name}/{name}.tr"

    full_dataset = load_dataset_from_files(en_path, tr_path)

    print(f" Dataset yüklendi: {name} (Train: {len(full_dataset['train'])}, Test: {len(full_dataset['test'])})")
    dataset_dicts[name] = {
        "train": full_dataset["train"],
        "test": full_dataset["test"]
    }
```

Figure 8. Dataset Loading

Another crucial concept is also the tokenization. It is an obligatory step to tokenize the raw data. After this step our datasets will become processable by the LLM models like mt5-small nllb and mBART. In the following code tokenizers are preparing with using transformers library. In the end of the preparing and using of these tokenizers, raw texts are converted to units that are called tokens.

```python
### TOKENIZER SETTINGS
###
###

import os
import torch
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from peft import LoraConfig, get_peft_model, TaskType, PeftModel


lora_output_base_dir = "/content/drive/MyDrive/MT_Project/trained_lora_adapters"

#  Kullanacağımız modeller ve LoRA konfigürasyonları
model_configs = {
    "mt5-small": {
        "name": "google/mt5-small",
        "lora_config": LoraConfig(
            r=8,
            lora_alpha=32,
            target_modules=["q", "v"],
            lora_dropout=0.05,
            bias="none",
            task_type=TaskType.SEQ_2_SEQ_LM
        )
    },
    "nllb-200M": {
        "name": "facebook/nllb-200-distilled-600M",
        "lora_config": LoraConfig(
            r=8,
            lora_alpha=32,
            target_modules=["q_proj", "v_proj"],
            lora_dropout=0.05,
            bias="none",
            task_type=TaskType.SEQ_2_SEQ_LM
        )
    },
    "mbart": {
        "name": "facebook/mbart-large-50-many-to-many-mmt",
```

Figure 9. Tokenizer Settings

After tokenization, model finetunings are made on the datasets. End of this step, (model number * llm number) amount finetuned models are obtained. As a result we can get translations from these models too. Also to prevent and hardness of the train LLMs LoRA adapter were used to make it shorter and less costful.

```python
### FINETUNE of MODELS with LORA ADAPTERS
###

from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, Seq2SeqTrainer, Seq2SeqTrainingArguments
from datasets import load_from_disk
from peft import get_peft_model, LoraConfig, TaskType
import os

# === AYARLAR ===
BASE_DIR = "/content/drive/MyDrive/MT_Project"
TOKENIZED_DIR = f"{BASE_DIR}/tokenized_data"
MODEL_DIR = f"{BASE_DIR}/trained_models"

model_name_map = {
    "mt5-small": "google/mt5-small",
    "nllb-200M": "facebook/nllb-200-distilled-600M",
    "mbart": "facebook/mbart-large-50-many-to-many-mmt"
}

model_type_map = {
    "mt5-small": "mt5",
    "nllb-200M": "nllb",
    "mbart": "mbart"
}

tokenizers = {
    "mt5-small": AutoTokenizer.from_pretrained("google/mt5-small"),
    "nllb-200M": AutoTokenizer.from_pretrained("facebook/nllb-200-distilled-600M", src_lang="eng_Latn", tgt_lang="tur_Latn"),
    "mbart": AutoTokenizer.from_pretrained("facebook/mbart-large-50-many-to-many-mmt", src_lang="en_XX", tgt_lang="tr_TR")
}

# === TRAIN / LOAD FUNCTION ===
def train_or_load_model(model_key, model_name, tokenizer, train_dataset, output_dir, use_lora=True):
    if os.path.exists(output_dir):
        checkpoints = [ckpt for ckpt in os.listdir(output_dir) if ckpt.startswith("checkpoint")]
        if checkpoints:
            print(f" Zaten eğitilmiş: {output_dir} -> Yükleniyor...")
```

Figure 10. Finetuning with LoRA Adapters

As a next step we can get the translations predictions from the finetuned models with the following code snippets.Also in this step if it is required also prefixes are added for the specific transformers models.

```python
def generate_translation(model_key, tokenizer, model, input_text):
    input_text = input_text.strip()
    if not input_text:
        return ""

    # Ortak generation parametreleri
    generation_kwargs = {
        "max_new_tokens": 128,
        "num_beams": 5,
        "early_stopping": True,
    }

    if model_key == "mt5-small":
        # mT5 için prompt kullan
        input_text_processed = f"translate English to Turkish: {input_text}"
        inputs = tokenizer(input_text_processed, return_tensors="pt", padding=True, truncation=True).to(model.device)
        # generation_kwargs burada kullanılıyor
        outputs = model.generate(**inputs, **generation_kwargs)

    elif model_key == "mbart":
        tokenizer.src_lang = "en_XX"
        tokenizer.tgt_lang = "tr_TR"
        input_text_processed = input_text # mBART için prompta gerek yok
        inputs = tokenizer(input_text_processed, return_tensors="pt", padding=True, truncation=True).to(model.device)

        # forced_bos_token_id için güvenlik kontrolü eklendi
        forced_bos_token_id = tokenizer.lang_code_to_id.get("tr_TR")
        if forced_bos_token_id is None:
            print(f"Uyarı: mBART için 'tr_TR' dil kodu bulunamadı. Lütfen tokenizer'ı kontrol edin.")
            # Bulunamaması durumunda model.generate'ın varsayılan davranışına bırakılabilir veya hata fırlatılabilir.
            # Şimdilik uyarı verip devam ediyoruz.
        else:
            generation_kwargs['forced_bos_token_id'] = forced_bos_token_id

        # generation_kwargs burada kullanılıyor
        outputs = model.generate(**inputs, **generation_kwargs)
```

Figure 11. Translation Predictions Step

After getting the first results of the predictions and statistical scores then it is needed to also evaluate GPT based scores too. To achieve this firstly in the related code cell users have to change with their OpenAI Api key. After that when users send the request they will get successful results. Then, these scores will be saved to the .csv file in the upcoming part of the gpt scores logic.

```python
# GPT skorlarını çekmek için fonksiyon (hata yönetimi ve retry ile)
def get_gpt_score_with_retry(translation, reference, source, max_retries=5, delay=5):
    system_prompt = (
        "You are an expert in evaluating machine translation quality. "
        "Rate the provided translation for fluency (grammar, readability) "
        "and adequacy (meaning, completeness) on a scale of 1-5. "
        "1=poor, 5=excellent. "
        "Respond only with a JSON object: {\"fluency_score\": [1-5], \"adequacy_score\": [1-5]}. "
        "Do not add any extra text."
    )
    user_prompt = (
        f"Source: {source}\n"
        f"Reference: {reference}\n"
        f"Translation: {translation}"
    )

    for attempt in range(max_retries):
        try:
            # OpenAI'ın yeni istemci kullanımına göre güncellendi
            client = openai.OpenAI(api_key=openai.api_key)
            response = client.chat.completions.create(
                model="gpt-3.5-turbo-0125", # Kullanılacak model
                messages=[
                    {"role": "system", "content": system_prompt},
                    {"role": "user", "content": user_prompt}
                ],
                response_format={"type": "json_object"}, # JSON yanıtı almak için önemli
                temperature=0
            )
            content = response.choices[0].message.content
            scores = json.loads(content)
            # API'den gelen skorlar int veya float olmayabilir, bu yüzden güvenli dönüşüm yapalım
            fluency = float(scores.get("fluency_score", np.nan))
            adequacy = float(scores.get("adequacy_score", np.nan))
            return fluency, adequacy
        except openai.APIError as e:
            print(f"API Hatası (Deneme {attempt + 1}/{max_retries}): {e}")
            if attempt < max_retries - 1:
```

Figure 12. GPT Score Evaluation

Finally, after these steps we would have all the things for the experiments needed like it is shown in the picture below.

```python
### EXPERIMENT 1 ###

###
### META LEARNER EDUCATION STAGE (BEST MODEL CLASSIFICATION - VERİ SIZINTISI DÜZELTİLDİ VE XGBOOST EKLENDİ)


!pip install -q joblib sentence-transformers scikit-learn xgboost

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import os
import joblib
from sentence_transformers import SentenceTransformer
import xgboost as xgb # XGBoost import edildi
from sklearn.preprocessing import LabelEncoder # LabelEncoder eklendi

print("Meta-Learner (Best Model Seçimi) Model Eğitimi Başlatılıyor...")

# Önceki hücreden oluşturulan ve kaydedilen DataFrame'i yükle
# Lütfen bu dosya yolunun doğru olduğundan emin olun:
output_csv_path_final_for_meta = "/content/drive/MyDrive/MT_Project/data-test/unified_meta_learner_tset_with_all_scores_and_labels.csv"
try:
    df_meta_learner_data = pd.read_csv(output_csv_path_final_for_meta)
    print(f"DataFrame '{output_csv_path_final_for_meta}' yüklendi. Satır sayısı: {len(df_meta_learner_data)}")
except FileNotFoundError:
    print(f"Hata: '{output_csv_path_final_for_meta}' bulunamadı. Lütfen 'GPT SCORE ADIMI' hücresini çalıştırdığınızdan emin olun.")
    raise # Hata oluşursa programı durdur

# HATA AYIKLAMA: Sütunları kontrol et
print("\nDataFrame sütunları yükledikten hemen sonra:")
print(df_meta_learner_data.columns.tolist())
# 'dataset' sütununu da required_cols'a ekledik
```

Figure 13. Example Experiment