Translating with LLMs

Multilingual NLP – Lab 5

Philippos Triantafyllou

January 15, 2026

Table of contents

1 Constructing a small contextual test set

We load `Helsinki-NLP/opus_tedtalks`: it is a Croatian-English parallel corpus of transcribed and translated TED talks, originally extracted from https://wit3.fbk.eu. This corpus is sentence aligned for both language pairs.

```python
from datasets import load_dataset

original_data = load_dataset("Helsinki-NLP/opus_tedtalks")
print(type(original_data))
print(original_data)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:104: UserWarning:
Error while fetching `HF_TOKEN` secret value from your vault: 'Requesting secret HF_TOKEN timed out. Secrets
↪  can only be fetched when running from the Colab UI.'.
You are not authenticated with the Hugging Face Hub in this notebook.
If the error persists, please let us know by opening an issue on GitHub
↪  (https://github.com/huggingface/huggingface_hub/issues/new).
  warnings.warn(
```

```
README.md: 0.00B [00:00, ?B/s]
```

```
en-hr/train-00000-of-00001.parquet:   0%|          | 0.00/9.93M [00:00<?, ?B/s]
```

```
Generating train split:   0%|          | 0/86348 [00:00<?, ? examples/s]
```

```
<class 'datasets.dataset_dict.DatasetDict'>
DatasetDict({
    train: Dataset({
        features: ['id', 'translation'],
        num_rows: 86348
    })
})
```

We define a context window of 3 sentences before and after the target sentence. We randomly sample 20 sentences from the training set, with its context window.

```python
import random
from datasets import Dataset

train: Dataset = original_data['train'] # type: ignore
translations = train["translation"]
n = len(translations)

valid_indices = list(range(3, n - 3))
random.seed(42)
sampled_indices = random.sample(valid_indices, k=20)
```

We a small test set of 20 examples structured as follows:

- source: the English sentence to be translated;
- translation: the Croatian translation of the source sentence, we will use this as reference to compute COMET scores later on;
- context: the context window of 3 sentences before and after the source sentence. We have both English and Croatian context.

We unfortunately have to reduce the size of the test set due to Ollama getting stuck at processing larger amounts of data. We found that beyond 20 examples, model's would simply stop working...

```python
examples = []

for i in sampled_indices:
    center = translations[i]

    window = {
        "source": center["en"],
        "reference": center["hr"],
        "context_before": [
            translations[i - 3],
            translations[i - 2],
            translations[i - 1],
        ],
        "context_after": [
            translations[i + 1],
            translations[i + 2],
            translations[i + 3],
        ],
    }

    examples.append(window)
```

```python
for i, ex in enumerate(examples[:3]):
    print(f"Example {i+1}:")
    print(f"Source: {ex['source']}")
    print(f"Reference: {ex['reference']}")
    print("Context before:")
    for ctx in ex["context_before"]:
        print(f" - {ctx['en']}")
    print("Context after:")
    for ctx in ex["context_after"]:
        print(f" - {ctx['en']}")
    print()

print(f"...and so on for a total of {len(examples)} examples.")
```

```
Example 1:
Source: (Laughter) Idiot, Robbins.
Reference: (Smijeh) Idiote, Robbins.
Context before:
 - The defining factor is never resources; it's resourcefulness.
 - And what I mean specifically, rather than just some phrase, is if you have emotion, human emotion,
 ↪  something that I experienced from you a day before yesterday at a level that is as profound as I've
 ↪  ever experienced, and if you'd communicated with that emotion I believe you would have beat his ass and
 ↪  won.
 - (Applause) But, how easy for me to tell him what he should do.
Context after:
 - But I know when we watched the debate at that time, there were emotions that blocked people's ability to
 ↪  get this man's intellect and capacity.
 - And the way that it came across to some people on that day -- because I know people that wanted to vote
 ↪  in your direction and didn't, and I was upset.
 - But there was emotion that was there.

Example 2:
Source: We all know everybody in this room makes mistakes.
Reference: Svi mi znamo da svatko od nas u ovoj sobi čini pogreške.
Context before:
 - This might strike you as a strange career move, but it actually has one great advantage: no job
 ↪  competition.
 - (Laughter) In fact, most of us do everything we can to avoid thinking about being wrong, or at least to
 ↪  avoid thinking about the possibility that we ourselves are wrong.
 - We get it in the abstract.
```

```
Context after:
 - The human species, in general, is fallible -- okay fine.
 - But when it comes down to me, right now, to all the beliefs I hold, here in the present tense, suddenly
  ↪  all of this abstract appreciation of fallibility goes out the window -- and I can't actually think of
  ↪  anything I'm wrong about.
 - And the thing is, the present tense is where we live.

Example 3:
Source: One man died after working a 36-hour shift.
Reference: Jedan je čovjek umro nakon rada u 36-osatnoj smjeni.
Context before:
 - This cell phone started its trajectory in an artisanal mine in the Eastern Congo.
 - It's mined by armed gangs using slaves, child slaves, what the U.N. Security Council calls "blood
  ↪  minerals," then traveled into some components and ended up in a factory in Shinjin in China.
 - That factory -- over a dozen people have committed suicide already this year.
Context after:
 - We all love chocolate.
 - We buy it for our kids.
 - Cote d'Ivoire, we have a huge problem of child slaves.

...and so on for a total of 20 examples.
```

There were no datasets online (that we found at least) that had timestamps, and many datasets did not have sentence level gold translations, so it was difficult to create proper examples. For example, we tried loading a English-Spanish dataset from TED talks, but the translations were only at the talk level, not sentence level. We tried segmenting the talks into sentences and then aligning them, but it was a failure. Although we do not understand Croatian, we have the correct setup to compute COMET scores later on, that we can interpret. Furthermore, the great advantage of this dataset is that the pairs are aligned at the talk level so constructing context windows is straightforward. Choosing 100 examples was a trade-off between having enough examples to compute meaningful COMET scores later on, and the time for one Ollama call. Since for each example we are trying many configurations, this can take a while. For that reason, we set up Ollama to run on GPU, considerably speeding up inference time.

2 Set up Ollama LLM environment

Everything is run on Google Colab with a A100 GPU.

```python
import threading
import subprocess
import time

def run_ollama_serve():
  subprocess.Popen(["ollama", "serve"])

thread = threading.Thread(target=run_ollama_serve)
thread.start()
time.sleep(5)
```

We try with two models: `llama3.2` and `mistral`.

```python
MODELS = ["llama3.2", "mistral"]
```

Here we simply set up our main Ollama call function, it is prompt and model agnostic, so we can reuse it later on.

```
SYSTEM_PROMPT = "You are a you are an expert translator that translates English text to Croatian."
USER_PROMPT = "Translate the following English text to Croatian: 'We all know everybody in this room makes
↪  mistakes.'"
```

Loads ollama.

We will use json schemas to structure the output of our LLM calls, this will be especially useful for the COT and n-shot experiments.

```
from pydantic import BaseModel

class TranslationSchema(BaseModel):
    translation: str
```

The function takes a model, a system prompt, a user prompt, and a schema to structure the output. It simply returns the parsed output according to the schema. We will wrap this function to create more specific outputs for evaluation. We set temperature to 0.

```
from typing import Type, TypeVar, Any
import ollama

T = TypeVar("T", bound=TranslationSchema)

def call_ollama(*, model: str, system_prompt: str, user_prompt: str, schema: Type[T]) -> T:
    # Convert schema to JSON Schema
    json_schema: dict[str, Any] = schema.model_json_schema()
    json_schema["additionalProperties"] = False

    # Ollama chat call with schema-constrained decoding
    response = ollama.chat(
        model=model,
        messages=[
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_prompt},
        ],
        format=json_schema,
        stream=False,
        options={
            "temperature": 0.0,
            "num_predict": 128,
        },
    )

    # Extract and validate
    content: str = response["message"]["content"]
    output = schema.model_validate_json(content)

    return output
```

One response takes around 3 seconds (on CPU it was around 15-20 seconds), so for 100 examples and many configurations this can take a while...

```
result = call_ollama(model=MODELS[0], system_prompt=SYSTEM_PROMPT, user_prompt=USER_PROMPT,
↪  schema=TranslationSchema)
print(result)
```

```
translation='Sve se zna da svaki u ovom salju napravljaju greške.'
```

Structured output is great, we can keep the prompts to task description and not worry about formatting.

Finally, we define a function that is our pipeline to run translations with different strategies (direct, COT, n-shot). It takes the examples, model, output path, strategy name, prompt builder function, and schema to structure the output. It writes the results to a file in JSONL format.

```python
import json
from tqdm import tqdm
from typing import Callable

def run_translation(
    *,
    examples: list[dict],
    model: str,
    output_path: str,
    strategy: str,
    prompt_builder: Callable[[dict, bool], str],
    schema: Type[T] = TranslationSchema,
) -> None:

    with open(output_path, "w", encoding="utf-8") as f:
        for example_id, ex in enumerate(tqdm(examples, desc=f"{strategy} translation")):
            translations = {}

            for use_context in (False, True):
                user_prompt = prompt_builder(ex, use_context)

                try:
                    parsed = call_ollama(
                        model=model,
                        system_prompt=SYSTEM_PROMPT,
                        user_prompt=user_prompt,
                        schema=schema,
                    )

                    key = "with_context" if use_context else "no_context"
                    translations[key] = parsed.translation

                except Exception:
                    key = "with_context" if use_context else "no_context"
                    translations[key] = "ERROR"

            record = {
                "example_id": example_id,
                "model": model,
                "strategy": strategy,
                "source": ex["source"],
                "translations": translations,
            }

            f.write(json.dumps(record, ensure_ascii=False) + "\n")
```

3 Experiments

Direct translation prompt

The direct translation prompt schema is simple, we just want the translation of the source sentence.

```python
class DirectTranslation(TranslationSchema):
    translation: str
```

We first design the prompt template. For each example we specify if we want context or not. Otherwise the prompt is straightforward.

```python
def build_direct_prompt(example: dict, use_context: bool = False) -> str:
    if use_context:
        context_lines = []

        for c in example.get("context_before", []):
            context_lines.append(f"- {c['en']}")

        for c in example.get("context_after", []):
            context_lines.append(f"- {c['en']}")

        context_block = "\n".join(context_lines)

        prompt = (
            "Translate the following sentence into Croatian.\n\n"
            "Context:\n"
            f"{context_block}\n\n"
            "Sentence to translate:\n"
            f"{example['source']}"
        )
    else:
        prompt = (
            "Translate the following sentence into Croatian.\n\n"
            f"{example['source']}"
        )

    return prompt
```

Let's try.

```python
run_translation(
    examples=examples,
    model=MODELS[0],
    output_path=f"direct_translations_{MODELS[0]}.jsonl",
    strategy="direct",
    prompt_builder=build_direct_prompt,
    schema=DirectTranslation,
)
```

```
direct translation: 100%|     | 20/20 [01:05<00:00,  3.27s/it]
```

```python
run_translation(
    examples=examples,
    model=MODELS[1],
    output_path=f"direct_translations_{MODELS[1]}.jsonl",
    strategy="direct",
    prompt_builder=build_direct_prompt,
    schema=DirectTranslation,
)
```

```
direct translation: 100%|     | 20/20 [00:36<00:00,  1.81s/it]
```

Chain-of-thought translation prompt

```python
class COTTranslation(TranslationSchema):
    translation: str
```

```python
def build_cot_prompt(example: dict, use_context: bool = False) -> str:
    if use_context:
        context_lines = []

        for c in example.get("context_before", []):
            context_lines.append(f"- {c['en']}")

        for c in example.get("context_after", []):
            context_lines.append(f"- {c['en']}")

        context_block = "\n".join(context_lines)

        prompt = (
            "Translate the following sentence into Croatian. Think step-by-step.\n\n"
            "Step 1: Identify potential translation difficulties.\n"
            "Step 2: Produce a context-aware translation.\n"
            "Step 3: Revise the translation to correct any errors.\n\n"
            "Context:\n"
            f"{context_block}\n\n"
            "Sentence to translate:\n"
            f"{example['source']}"
        )
    else:
        prompt = (
            "Translate the following sentence into Croatian. Think step-by-step.\n\n"
            "Step 1: Identify potential translation difficulties.\n"
            "Step 2: Produce a translation.\n"
            "Step 3: Revise the translation to correct any errors.\n\n"
            "Sentence to translate:\n"
            f"{example['source']}"
        )

    return prompt
```

```python
run_translation(
    examples=examples,
    model=MODELS[0],
    output_path=f"cot_translations_{MODELS[0]}.jsonl",
    strategy="cot",
    prompt_builder=build_cot_prompt,
    schema=COTTranslation,
)
```

```
cot translation: 100%|     | 20/20 [01:03<00:00,  3.17s/it]
```

```python
run_translation(
    examples=examples,
    model=MODELS[1],
    output_path=f"cot_translations_{MODELS[1]}.jsonl",
    strategy="cot",
    prompt_builder=build_cot_prompt,
    schema=COTTranslation,
)
```

```
cot translation: 100%|     | 20/20 [00:43<00:00,  2.15s/it]
```

n-best generation prompt

4 Evaluation with COMET