JSP3: Report for task **WSD-M**

Adam Hurdálek (Hurdys)
hurdad00@upol.cz

January 31, 2026

1 Introduction

- Main goal:

  The primary objective of this project is to develop and evaluate a robust Word Sense Disambiguation (WSD) pipeline for the Czech language. Vaguely said, have a system that will extract the definition of each word for the text that user will input.

- My mission:

  Inside this project I have a special task. To test different models. That is why I customized my environment just to my purposes. I have to run some text through a tagger powered by LLM to extract tags from wordnet which belongs to each words meaning. However, to achieve meaningful results, I had to go beyond simple testing. I was quite unsure with the assignment, so I made what made sense.

2 Background

The project relies on several key resources and existing methodologies:

- **Czech WordNet (WN-ntumc-ces):** The lexical database providing synsets and relations for Czech.
- **English WordNet (OMW 1.4/3.1):** Used as the source of textual definitions through the Interlingual Index (ILI), a method often referred to in literature as the **"English Bridge."** I had to use it because absence of the definitions in Czech wordnet.
- **Simplemma:** A lightweight library used for morphological analysis and lemmatization of the input Czech text.
- **Ollama API:** A local inference engine allowing the deployment of open-weight models on consumer-grade hardware.
- **Tagger from NTUMC:** Natural Text Understanding Multilingual Corpus made by Francis Bond and contributors. I used some logic from his tagger.
- **GPT 5.2 Free:** Served as an architecture consultant, providing guidance through the Linux terminal, WordNet syntax, and error interpretation. Additionally, it assisted with prompt optimization and the development of the evaluation logic, including the automated `.xlsx` output.

Resources

- *Pala, Karel; et al., 2011, Czech WordNet 1.9 PDT, LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), http://hdl.handle.net/11858/00-097C-0000-0001-4880-3.*
- *John P. McCrae, Alexandre Rademaker, Francis Bond, Ewa Rudnicka and Christiane Fellbaum. 2019. English WordNet 2019 – An Open-Source WordNet for English. In Proceedings of the 10th Global WordNet Conference (GWC 2019), Wrocław, Poland. Global Wordnet Association.*
- *Adrien Barbaresi. 2021. Simplemma: A simple multilingual lemmatizer for Python. Zenodo. https://doi.org/10.5281/zenodo.4673264*
- *Bond Lab. 2024. NTUMC Tagger: tag-llm.py (refactored-tagdb branch). GitHub repository, bond-lab/NTUMC. Available at: https://github.com/bond-lab/NTUMC/blob/refactored-tagdb/ntumc/taggers/tag-llm.py*

  Liling Tan and Francis Bond. 2012. Building and annotating the linguistically diverse NTUMC (NTU-multilingual corpus). International Journal of Asian Language Processing 22(4): 161–174.
- *OpenAI. (2026). ChatGPT (GPT-5.2) [Large language model]. https://chat.openai.com/*

3 Approach

3.1 Implementation and early Technical Decisions

I started creating a notebook on Google Colab and was trying to get the tagger running. I have downloaded Ollama and started working. From the start I have faced numerous complications. The tagger was tough to implement for me, so I made radical decision and developed my own pipeline.

The primary reason for developing a custom pipeline from scratch instead of using the provided `tag_llm.py` script was:

- **Architectural Complexity:** The original tool has strict dependencies and is deeply integrated into the NTUMC repository. Successfully running it would require cloning the entire library and setting up a specific database environment (NTUMC database format), which was unfeasible for an available cloud environment.

- **Data Format Hurdles:** The script is designed to work with structured database records rather than raw text files. Transitioning from a simple `.txt` input to a compatible database format presented an unnecessary technical hurdle that shifted focus away from the linguistic core of the task.

- **Customization and Control:** By writing a custom script, I gained full control over the prompt engineering and context windowing strategies. This allowed for the implementation of the "English Bridge" (ILI mapping), which was not straightforward to integrate into the existing `tag_llm.py` logic.

- **Skill Development:** It also allowed me to do some tweaks while working and it elevated my coding skills, regex... Especially, building the pipeline from scratch helped me understand importance of a version control system. *(finally...)*

3.2 Hardware and Infrastructure

Using Colab took me ages. That is why I paid some money for a GPU rental. I have ended with Colab and switched to **Vast.ai** — a site where you can rent a machine for your projects.

This was a perfect decision, even crucial for my project, may I say. Since I wanted to test multiple models and it got really time-consuming while really getting nowhere. I was able to access the Jupyter app with the explorer, etc. Grind finally seemed to be improving.

Component	Specification
GPU	NVIDIA RTX 3060 12GB
CPU	AMD Ryzen 5 5600X (6-Core)
RAM	32 GB DDR4
Storage	32 GB NVME SSD
Mainboard	B550-A PRO (PCIe 3.0)

Table 1: Hardware specifications of the Vast.ai instance.

So I remained in this environment and started playing with different models. My code went through a bunch of versions. Early ones were really straightforward and I was rewriting code a lot.

```
class Tagger:
    def __init__(self, model_name):
        self.model = model_name

    def tag_word(self, sentence, target_word):
        synsets = wn.synsets(target_word, lang='cs')
        if not synsets:
            return None

        options = ""
        for s in synsets:
            options += f"- ID: s.id | Význam: s.definition()"

        prompt = f"""Úkol: Vyber správné ID významu pro slovo 'target_word' ve větě.
Věta: "sentence"
Možnosti z WordNetu:
options
Odpověz POUZE kódem ID (např. cswn-06307680-n). Nepiš žádné vysvětlení."""

        response = ollama.generate(model=self.model, prompt=prompt)
        response_text = response['response']

        import re
        cleaned = re.sub(r'<think>.*?</think>', '', response_text, flags=re.DOTALL).strip()
        return cleaned
```

Snippet 1: First ever version of my tagger.

In next versions I pretty much copied the whole logic from `tag_llm.py` except from parsing because I didn't understand the code (skill issue - atleast I've used regex)

3.3 Evolution of the Logic and Optimization

The early notebook versions performances were a significant bottleneck: chewing 7 sentences with 5 models was running for 40 minutes. Consequently, I realized I must do some optimization steps:

- **Feature Removal:** I removed the sentiment analysis since it is not my task and I feel there must be better way to measure than just ask a model.
- **Efficiency Gain:** Most importantly, I have added a **Czech stopwords filter**, which saved me countless hours of redundant processing.
- **Definitions addition:** Czech WordNet does not contain definitions *(glosses)*. In order to boost performance, I added English definitions via the Interlingual Index (*"English Bridge"*). By implementing this, the accuracy of the models skyrocketed by approximately 15% `!` Which is a huge improvement that I am particularly proud of.

3.4 Model Selection and Evaluation Strategy

Until the two latest versions, I was picking models based on what I have read about them on the Ollama website, personal preferences, and size. This choosing process was based solely on qualitative observations, which were not 100% scientific *(Few tags were just hard to rate, e.g.: same synonyms but different tag)*

To transition to a more data-driven approach, I wrote a short benchmark text, added a time counter, and tried to implement the EVAL group's code. However:

- **Evaluation Challenges:** I have ultimately failed with the EVAL code, probably because of a lack of documentation and my unfamiliarity with `JSON`.
- **Practical Solution:** Since evaluation is not my main goal, I found it fair to hardcode correct tags and let ChatGPT to code an automated output which I was then reviewing. This approach was my last resort, but it served ideally for my own testing so I think it is okay.

3.5 Technical Pipeline Summarized:

1. **Preprocessing:** Lemmatization via `simplemma` and filtering using a custom Czech stop-words list.
2. **Knowledge Retrieval:** Fetching Czech synsets from WordNet and mapping them to English definitions via ILI (The "English Bridge").
3. **Context Formatting:** Implementation of a sliding window ($w = 2$) and target word highlighting (`***word***`) to guide the LLM's attention.
4. **Inference:** Querying the local Ollama instance with a structured prompt and a temperature of 0 for reproducibility.
5. **Post-processing:** Robust parsing of the LLM output using Regular Expressions to extract the final tag.

The pipeline was finally fitting for my purposes, allowing me to shift my focus entirely to testing different models (which took more time than I thought).

Link to code:

JSP3_WSD-M_robuss0.5.ipynb — benchmark_text.txt

4 Results

The primary goal of the experiment was to benchmark various LLM architectures to identify the optimal balance between Czech language understanding, inference speed, and output stability.

4.1 Experimental Constraints

Testing was conducted on a **Vast.ai** instance with a 32GB storage limit. This constraint prevented simultaneous storage of all models; therefore, the evaluation process was iterative (downloading, testing, and deleting models sequentially), which significantly extended the processing time and also I was unable to test many 14B models at once.

4.2 Tested Models and Qualitative Observations

A total of 10 models were evaluated during the development of the pipeline. The following table summarizes their performance characteristics and the reasons for their inclusion or exclusion in the final benchmark.

4.3 Example Output Analysis

The superiority of `mistral-nemo` was evident in handling polysemous words. For the target word *"panák"* (meaning *shot/drink* in the context), smaller models often defaulted to the primary sense *(dummy/figure)*, whereas Mistral correctly utilized the English definition bridge to select `cswn-07826283-n` (shot/liquor).

4.4 Analysis of Model Selection

The selection of the top four models was not arbitrary; each represented a distinct approach to the problem that was crucial to compare against the baseline.

- **Mistral-Nemo (12B):** Selected to test the limit of the hardware. With 12 billion parameters, it sits right on the edge of the RTX 3060's VRAM capacity. The goal was to determine if a "larger but still local" model would outperform standard 8B models.
- **Llama 3.1 (8B):** Included as the industry standard. No benchmark would be complete without comparing against Llama, which served as the reference point for reliability.

- **Gemma 2 (9B):** Chosen for its **unique architecture**. Thanks to this design, it consistently performed better than other models in the ∼8B category, offering a higher density of knowledge per parameter.
- **Qwen 2.5 (14B):** Included as a hardware "stress test." I wanted to verify if sacrificing speed for a massive parameter count (14B) would yield proportionally better semantic understanding.

4.5 Inference Speed Anomalies

Contrary to expectations, the larger **Mistral-Nemo (12B)** was the fastest model on the RTX 3060, outperforming even smaller 8B models. This is likely attributed to its specific tokenizer (*Tekken*), which is highly efficient for non-English languages, compressing Czech text into fewer tokens than Llama or Gemma.

4.6 Reasons for Exclusion of Other Models

Several models were tested but failed to make the final cut for specific reasons:

- **DeepSeek-R1:** This model suffered from frequent output format errors. While it was slower than Qwen3, its semantic accuracy was actually very high — when it managed to produce a valid tag. However, its instability made it unusable for automation.
- **Aya (8B):** I had high expectations for Aya as the *"Dark Horse"* of the experiment due to its strong focus on multilingualism. Unfortunately, it failed to adhere to the strict output format, being too "chatty" for the pipeline.
- **Qwen 3 / Qwen Variants:** Despite the documentation claiming superior knowledge of **Slavic languages**, these models did not demonstrate a significant advantage over Llama 3.1 in this specific WSD task and were overshadowed by the larger Qwen 2.5 context handling.
- **LukasPlevac/Qwen3-CZ:** Fine-tuned qwen3:8b on Czech data by some homelabber. Excluded primarily due to extreme latency issues on the test hardware.

4.7 Results (Final Benchmark - Summarized)

.

Model	Key Observation / Verdict
`mistral-nemo:12b`	**The Champion.** Highest accuracy. Excellent context understanding and perfect instruction following.
`llama3.1:8b`	Reliable runner-up. Fast and stable, slightly less accurate on nuanced synonyms than Mistral.
`qwen2.5:14b`	Disappointing. Slowest despite largest size, no accuracy advantage over 12B Mistral.
`gemma2:9b`	Decent but prone to hallucinations. Struggled with strict ”ID-only” output format.
`aya:8b`	Multilingual model (∼60% accuracy). Added conversational filler, breaking parser.
`lukasplevac/Qwen3-CZ`	**Failed.** Czech fine-tune but extremely slow (> 10s/word) and unstable format.
`deepseek-r1:8b`	**Unusable.** Sometimes outputted Chain-of-Thought instead of tags, impossible to parse.
`qwen3:8b`	Generic. Overshadowed by superior Llama 3.1 in same size category.
`qwen2.5:3b`	**Too small.** Fast but lacked semantic depth. Frequent FORMAT_ERRORs.
`granite4:3b`	Like Qwen 3B. Insufficient parameters for complex Czech WSD tasks.

Table 1: Qualitative evaluation of tested models.

Model	Accuracy (%)	Avg Time (s/word)
mistral-nemo:12b	**95.65%**	**0.75**
llama3.1:8b	91.30%	1.02
gemma2:9b	86.96%	1.00
qwen2.5:14b	86.96%	1.33

Table 2: Final performance comparison.

Link to one of the output files for example:

wsd_final_20260130_1949.xlsx

Sorry for not including graphs. I don't have these and unfortunately I have not much time till deadline. These data are from my output reports

5 Discussion

5.1 Analysis of Results

The results unequivocally demonstrate that the **"English Bridge"** (mapping Czech synsets to English definitions via ILI) is the critical component for a functional Czech WSD pipeline. Without textual definitions, LLMs operate on mere guesswork; with them, the system achieved over 95% accuracy using `mistral-nemo`. I hope that ILI can be trusted.

Furthermore, the experiment revealed a hardware "sweet spot" on the RTX 3060. While standard 8B models (Llama 3.1) are reliable, the 12B architecture of Mistral-Nemo provided a significant jump in semantic understanding without exceeding VRAM limits or sacrificing inference speed, thanks to efficient tokenization.

5.2 Challenges and Limitations

Throughout the project, several technical and linguistic hurdles were encountered:

- **Technical Dependencies:** The architectural complexity of the original `tag_llm.py` necessitated the creation of a custom, lightweight pipeline.
- **Output Instability:** "Chatty" models (e.g., Aya) refused the strict "ID-only" format, requiring a robust RegEx parser to extract tags.
- **Synonym Granularity:** WordNet ambiguity made distinguishing between nearly identical synsets subjective, making automated evaluation extremely difficult.
- **Lemmatization Errors:** `simplemma` occasionally misidentified polysemous words, confusing the downstream LLM.
- **Hardware Constraints:** The 12GB VRAM limit on the RTX 3060 restricted the testing of larger models (> 14B) and slowed down the benchmarking process significantly.
- **Team Coordination:** Due to fragmentation in team communication, the majority of the testing and development had to be made independently. (But this is solely our fault.)

5.3 Future Work

To transition this project from a prototype to a usable tool, the following improvements are proposed:

- **Pipeline Universalization:** Refactor the static Jupyter Notebook into a standard application usable on any text file.
- **Better Lemmatization:** Replace the simple dictionary method with a context-aware analyzer (e.g., MorphoDiTa) to fix preprocessing errors.
- **Context Caching:** Implement caching for long books. If a word appears in the same context twice, the model shouldn't need to process it again.
- **Multi-language Support:** Modify the code so it is easy to switch between different languages (loading different stop-words and WordNets) without rewriting the script.
- **Expanded Evaluation:** Create a larger, professionally annotated dataset to perform testing that is statistically more significant.

  (! This one is important !)
- **Few-Shot Prompting:** Providing the model with 1–2 examples in the prompt could stabilize the output for smaller models.
- **And much more...** During the development I was having many ideas how to do some little things better. I was bottlenecked by my own lack of skill. I was doing a lot of research on things that I am unable to do and I haven't had time to learn those. My curiosity was almost frightening — always slowing me down, getting me out of focus.

6 Conclusions

This project successfully demonstrated the viability of using open-weight Large Language Models for Word Sense Disambiguation in the Czech language. I effectively overcame deficiency of the Czech WordNet—the lack of textual definitions—through the implementation of the **"English Bridge"**.

My benchmark of ten different models on an NVIDIA RTX 3060 revealed that **Mistral-Nemo (12B)** is the optimal choice for this task. But I wish I've done more and bigger testing with much more models and much better hardware.

In conclusion, this work proves that somehow accurate WSD can be done without massive enterprise resources.

Last words:

Firstly, sorry for not writing academically all the time. I tend to story-tell what was happening even when trying not to. Hope it's not a big deal.

This semestral project was truly exceptional. I struggled a lot during the semester, but during exam season I really surpassed myself. At the start, I couldn't find my grip and was unsure about the assignment. I may have bent some rules a little—by not fully adapting your code and not adapting other WSD code (We have bad communication in class). Probably that was my turning point—when I started doing it *my way*. (I really hope I've done the assingment.)

I'm genuinely proud of my work. It took roughly 70 hours. The Vast.ai instance ran nearly two weeks while I restlessly developed, debugged, studied different strategies, and changed approaches multiple times. Those were moments when I truly learned—by myself, powered by my own desire.

Throughout the process, I felt a constant urge to make it better and better. Which was a little annoying, since I'm just a beginner.

I would love to test on better hardware, with improved methodologies, and more classmate collaboration.

Thanks alot, Hurdálek