

# Claude 4 Opus chat on: Stage 1.2 Stanza and LatinCy

User:  
NEReus /

---

**User:**  
metadata: BB

Hi Claude.

First, here are some messages from past runs:

**## Summary: Latin Inscription NER Research - Stage 1 Data Preparation**

**### Project Context**

We're working on research around **\*\*named entity recognition (NER) in Latin inscriptions\*\***. The ultimate goal is to evaluate how various part-of-speech (POS) taggers perform with Latin inscriptional texts.

**### What We Accomplished in Stage 1**

We created a data extraction pipeline to prepare test data for POS tagger evaluation:

**1. \*\*Analyzed the Dataset Structure\*\***

- Working with a large Latin inscription dataset (LIST\_v1-2.parquet) containing 500k+ inscriptions
- Dataset has 66 columns including multiple text versions, dating, geographic, and metadata fields
- Key text fields include conservative (close to preserved state) and interpretive (with restorations/expansions) versions

**2. \*\*Defined Extraction Requirements\*\***

- Need 20 random inscriptions from the **\*\*top 10% longest texts\*\*** (using ``clean_text_interpretive_word`` length as the metric)
- Selected fields: LIST-ID, conservative text, two interpretive text versions, dating info (not\_before/not\_after), and geographic data (geometry, ancient findspot, urban context)
- Output format: JSON for easy processing in downstream POS tagging tools

**3. \*\*Built the Extraction Script\*\***

- Python script that reads parquet, calculates text lengths (handling nulls), filters for top decile, randomly samples 20 inscriptions
- Includes reproducibility (random seed), detailed statistics, and missing data reporting
- Output: ``POS-LIST-test1.json`` with structured data ready for POS tagger evaluation

#### ### Key Decisions Made

- Use interpretive word text for length calculation (most complete version)
- Focus on lengthy inscriptions (top 10%) for more substantial POS tagging evaluation
- Include both conservative and interpretive texts to compare POS performance on different text preparations
- Include temporal and geographic context which may be relevant for understanding linguistic variations

#### ### Next Steps (Implied)

The extracted JSON file will be used to test various POS taggers on Latin inscriptional text, likely comparing their performance on conservative vs. interpretive text versions.

### ## Project Summary: Latin Inscription NER - POS Tagging Validation Stage

#### ### Overall Project Goal

We're working on named entity recognition (NER) in Latin inscriptions. The broader research objective is to evaluate how various part-of-speech (POS) taggers perform on epigraphic Latin texts.

#### ### Current Stage: POS Tagging Gold Standard Creation

We're in Step 1 of the validation process - creating a gold standard dataset for POS tagging evaluation.

#### ### What We've Accomplished

- 1. \*\*Received input data\*\*:** A JSON file (POS-LIST-test1.json) containing Latin inscriptions with multiple text representations, including:
  - `text\_conservative`: Raw/diplomatic transcription
  - `text\_interpretive\_word`: Expanded/interpreted version (our focus)
  - Metadata: dating, geography, LIST-ID identifiers
- 2. \*\*Designed validation workflow\*\*:** Created a Python script that:
  - Extracts the `text\_interpretive\_word` field from 20 inscriptions
  - Splits each inscription into individual tokens
  - Generates an Excel workbook with one inscription per sheet
  - Implements constrained vocabulary using Universal Dependencies (UD) v2 tagset (17 tags)
  - Provides dropdown validation for consistent annotation
  - Includes columns for: Word, POS, and Notes
- 3. \*\*Key decisions made\*\*:**
  - Using UD tagset for standardization and compatibility

- Excel format for ease of use and data validation features
- Minimal column structure (Word, POS, Notes) for focused annotation

### ### The Data

The inscriptions vary significantly in length (from ~136 to 3561 characters) and include various Latin text types - epitaphs, military diplomas, dedicatory inscriptions, etc. They span different time periods and geographic locations across the Roman Empire.

### ### Next Implied Steps

1. Manual annotation of these 20 inscriptions to create the gold standard
2. Running various POS taggers on the same inscriptions
3. Comparing tagger outputs against the gold standard
4. Analyzing performance metrics to understand how well modern NLP tools handle epigraphic Latin

This validation spreadsheet is the foundation for systematic evaluation of POS tagging tools on Latin inscriptional evidence.

Here, we're going to be working on the non-LLM based POS tagging libraries.

Functionally decompose the task at hand, and then ask us one question at a time until you think you know enough to make this script.

LatinCy example:

```
# pip install https://huggingface.co/latincy/la_core_web_lg/resolve/main/la_core_web_lg-any-py3-
none-any.whl
import spacy
nlp = spacy.load('la_core_web_lg')
doc = nlp('Haec narratur a poetis de Perseo')

print(f'{doc[0].text}, {doc[0].norm_}, {doc[0].lemma_}, {doc[0].pos_}')

# > Haec, haec, hic, DET
```

and treetaggerwrapper.

My first intention is to make a proof of concept that has them all parsing the inscription

Dis Manibus sacrum Marisa Frontonis filia pia vixit annos LX hic sita est Dis Manibus sacrum  
Marhulus Luci filius pius vixit annis LXV hic est -- mostly to make sure all the various bits and  
bobs are installed correctly.

---

Assistant:

Looking at this task, I need to understand the full scope of what we're building. Let me break down the functional components:

## Functional Decomposition

### 1. POS Tagger Integration Layer

- Initialize multiple taggers (LatinCy, TreeTagger, Lamon?, others?)
- Handle different input/output formats
- Manage dependencies and models

### 2. Data Processing Pipeline

- Load inscriptions from JSON
- Preprocess text for each tagger's requirements
- Handle Latin-specific features (abbreviations, numerals, etc.)

### 3. Output Standardization

- Map different tagsets to common format (UD?)
- Align tokenization differences
- Structure results for comparison

### 4. Evaluation Framework

- Compare against gold standard (when available)
- Compute metrics
- Generate reports

Let me start with my first question:

#### Question 1: Which POS taggers should we include in this evaluation?

From what I can see, you've mentioned:

- LatinCy (spaCy-based) - already shown example
- TreeTaggerWrapper - mentioned
- Lamon (from the paste.txt documentation)

Are there other Latin POS taggers you want to include? For example:

- CLTK (Classical Language Toolkit) - has multiple Latin taggers
- Stanza (Stanford NLP) - has Latin models
- Any other specialized Latin NLP tools?

Also, should we prioritize getting a basic proof-of-concept working with just 2-3 taggers first, or aim for comprehensive coverage from the start?

---

**User:**

**metadata: BB**

**latincy is CLTK's model. Added Stanza**

**However, right now, let's just add each tagger one at a time to an example, to demonstrate that they're working.**

---

**Assistant:**

Perfect! Let's build this incrementally. Before I create the proof-of-concept script, I need to understand the setup requirements for each tagger.

## Question 2: What are the installation/setup requirements for each tagger?

From what I can see:

1. **LatinCy** - You've shown the pip install command for the model
2. **TreeTagger** - I see you have a treetagger/README file, suggesting you have TreeTagger installed locally.  
Do you have:
  - The TreeTagger binary installed?
  - The Latin parameter file downloaded?
  - The treetaggerwrapper Python package installed?
3. **Lamon** - From paste.txt, it looks like `pip install lamonpy` should work
4. **Stanza** - Usually requires:
  - `pip install stanza`
  - Downloading the Latin model with `stanza.download('la')`

Can you confirm:

- Do you have TreeTagger set up already? If so, where is it installed?
- Have you tried installing Lamon and Stanza yet?
- Are there any specific versions or compatibility issues I should be aware of?

This will help me write a script that handles any missing dependencies gracefully and provides helpful setup instructions where needed.

---

**User:**

**metadata: BB**

**stanza added. give me la as a oneliner for the large file downloads, please? I have treetagger ... downloaded. But I don't have the executable tested at all. That was put off until... now, really....**

**Let's just take this one tiny step at a time.**

---

**Assistant:**

Got it! Let's start with the Stanza model download oneliner:

---

**User:**

bash

```
python -c "import stanza; stanza.download('la')"
```

Now, let's create a simple proof-of-concept script that tests each tagger **one at a time** on your example inscription. We'll start with just LatinCy since you already have it working:

---

**User:**

python

```
#!/usr/bin/env python3
"""
Proof of concept: Test Latin POS taggers one by one
"""

# Test inscription
TEST_TEXT = "Dis Manibus sacrum Marisa Frontonis filia pia vixit annos LX hic sita est Dis Manibus sacrur

print("="*60)
print("Testing Latin POS Taggers")
print("="*60)
print(f"\nTest text: {TEST_TEXT[:50]}...")
print("="*60)

# Test 1: LatinCy (spaCy-based)
print("\n1. Testing LatinCy...")
try:
    import spacy
    nlp = spacy.load('la_core_web_lg')
    doc = nlp(TEST_TEXT)

    print("✓ LatinCy loaded successfully")
    print("\nFirst 5 tokens:")
    for i, token in enumerate(doc[:5]):
        print(f" {token.text:10} -> POS: {token.pos_:6} Lemma: {token.lemma_}")
    print(f" ... ({len(doc)} tokens total)")

except Exception as e:
    print(f"✗ LatinCy failed: {e}")

print("\n" + "="*60)
```

Run this first to make sure LatinCy works. Once that's confirmed, should we add Stanza next?

## User:

metadata: BB

I added from rich import print at the top, just so we can have some nice pretty printing with markdown.

LatinCy passed.

Assistant:

Great! Let's add Stanza as the next tagger:

User:

python

```
#!/usr/bin/env python3
"""
Proof of concept: Test Latin POS taggers one by one
"""
from rich import print

# Test inscription
TEST_TEXT = "Dis Manibus sacrum Marisa Frontonis filia pia vixit annos LX hic sita est Dis Manibus sacrur

print("="*60)
print("[bold blue]Testing Latin POS Taggers[/bold blue]")
print("="*60)
print(f"\n[dim]Test text: {TEST_TEXT[:50]} ...[/dim]")
print("="*60)

# Test 1: LatinCy (spaCy-based)
print("\n[bold]1. Testing LatinCy...[/bold]")
try:
    import spacy
    nlp = spacy.load('la_core_web_lg')
    doc = nlp(TEST_TEXT)

    print("[green]✓ LatinCy loaded successfully[/green]")
    print("\nFirst 5 tokens:")
    for i, token in enumerate(doc[:5]):
        print(f" {token.text:10} -> POS: {token.pos_:6} Lemma: {token.lemma_}")
    print(f" [dim]... ({len(doc)} tokens total)[/dim]")

except Exception as e:
    print(f"[red]✗ LatinCy failed: {e}[/red]")

# Test 2: Stanza
```

```

try:
    import stanza
    # Initialize with silent mode to reduce verbosity
    nlp_stanza = stanza.Pipeline('la', processors='tokenize,pos,lemma', verbose=False)
    doc_stanza = nlp_stanza(TEST_TEXT)

    print("[green]✓ Stanza loaded successfully[/green]")
    print("\nFirst 5 tokens:")
    token_count = 0
    for sent in doc_stanza.sentences:
        for token in sent.words[:5]:
            print(f" {token.text:10} -> POS: {token.upos:6} Lemma: {token.lemma}")
            token_count += 1
            if token_count >= 5:
                break
        if token_count >= 5:
            break

    total_tokens = sum(len(sent.words) for sent in doc_stanza.sentences)
    print(f" [dim]... ({total_tokens} tokens total)[/dim]")

except Exception as e:
    print(f"[red]✗ Stanza failed: {e}[/red]")

print("\n" + "="*60)

```

Run this and let me know if Stanza works! Then we'll add TreeTagger next.

---

**User:**

**metadata: BB**

**huh. Stanza has the names as nouns. And beefing it up was poor. But hey, research!**

**Next!**

**We're out of context. Give me a message for your future self, please.**

**3 / 3**

---

**Assistant:**

**Context for Next Session**



## Current Task

Building a proof-of-concept script to test multiple Latin POS taggers on inscriptional text. Testing one tagger at a time on this inscription:

---

## User:

"Dis Manibus sacrum Marisa Frontonis filia pia vixit annos LX hic sita est Dis Manibus sacrum Marhulus Luci filius pius vixit annis LXV hic est"

## Progress

- ✓ **LatinCy** - Working (uses PROPN for names)
- ✓ **Stanza** - Working (tags names as NOUN, not PROPN - interesting research finding!)
- 🕒 **TreeTagger** - Downloaded but needs setup/testing
- 🕒 **Lamon** - Not tested yet

## Next Steps

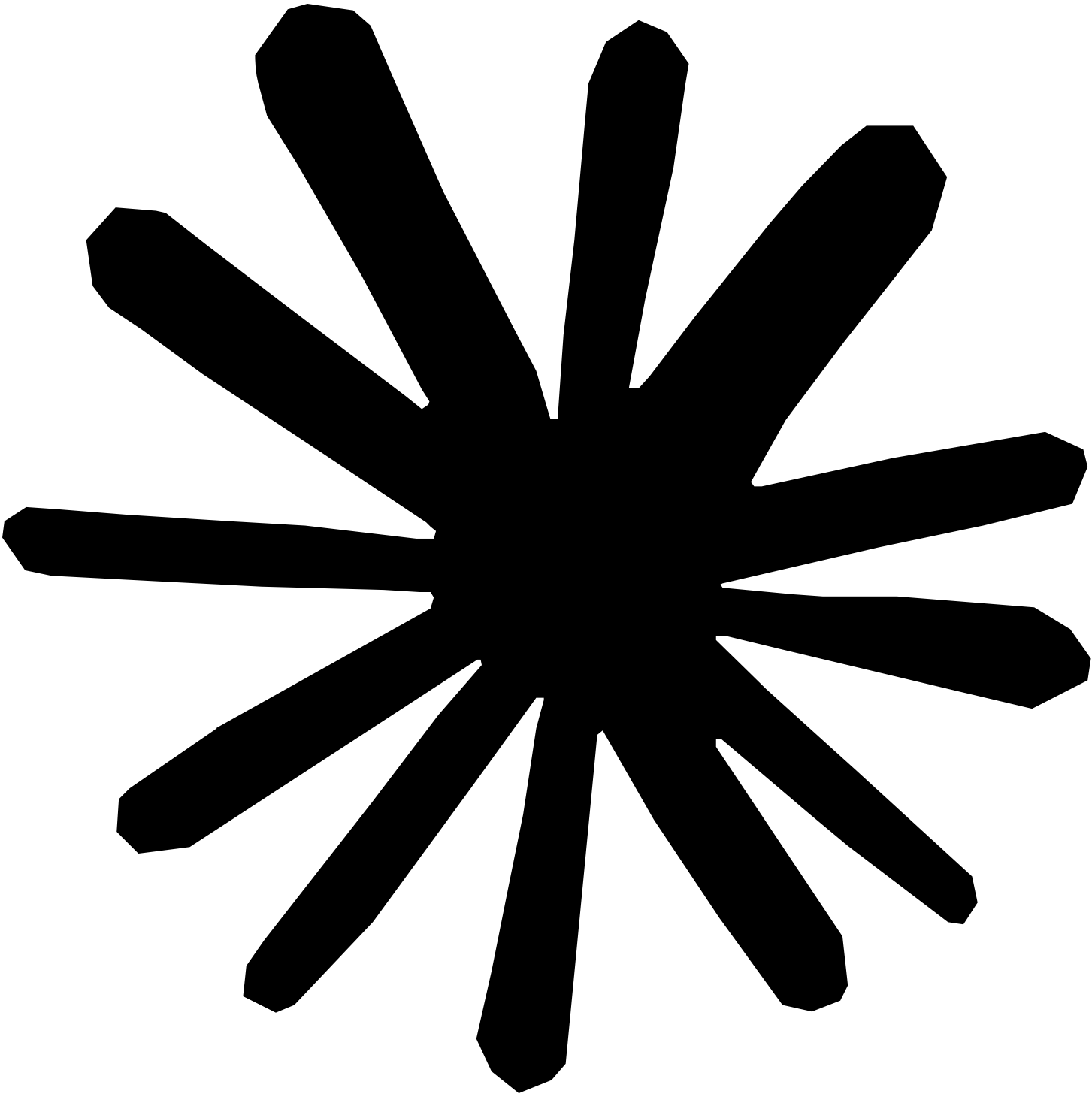
1. **TreeTagger**: User has it downloaded but hasn't tested the executable. Need to:
  - Set up TreeTagger binary path
  - Test with treetaggerwrapper
  - Handle Latin parameter file
2. **Lamon**: Install via `pip install lamonpy` and test

## Key Files

- POS-LIST-test1.json - 20 inscriptions for full evaluation
- Using rich for pretty printing
- Building toward comparing taggers against gold standard

## Research Note

Already finding interesting differences: Stanza treats proper names differently than LatinCy. This variation is exactly what the research aims to document!



[Claude can make mistakes.](#)  
[Please double-check responses.](#)

Choose Files

No file chosen

metadata:

Chat controls

User: Claude

**Opus 4.1**

Powerful, large model for complex challenges [Learn more](#)

**Project content**

 [NEReus](#)  
Created by You

**petrifiedvoices/NEReus**

master



GITHUB

**Content**