# Apache OpenNLP and LLMs - Where does OpenNLP fit in?

Jeff Zemerick

# Hi!

I’m Jeff.

- OpenNLP user since ~2010
- PMC chair since ~2020
- Independent consultant doing cloud / data / NLP stuff
- Pittsburgh, PA, USA

# Introduction

With large language models (LLMs), NLP has exploded in the forefront of almost everything.

So where does this leave Apache OpenNLP? What role does it have in today's LLM-dominated NLP landscape?

Does anything else have names more fun than LLMs?

# What is Apache OpenNLP?

- A machine learning based toolkit for the processing of natural language text. https://opennlp.apache.org/

- Can do common NLP tasks:
  - Tokenization
  - Sentence segmentation
  - Named-entity extraction
  - Chunking
  - Language Detection
  - Parts-of-speech tagging
  - Document classification (sentiment)

- Joined ASF incubator in 2010.
- Top-level project in 2012.
- Current version is 2.3.0 released in July 2023.

Pre-ASF OpenNLP files on SourceForge go back to 2003.

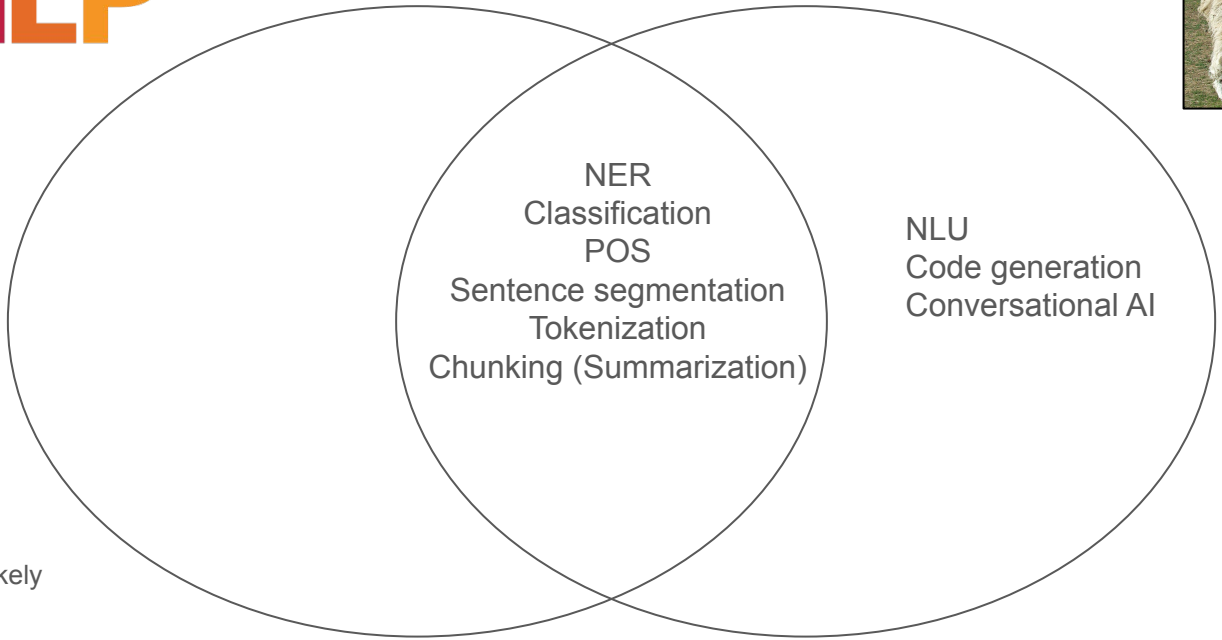
The team and a thanks to everyone listed there and everyone who has ever made a contribution to the project.

# What are Large Language Models (LLMs)?

- A language model characterized by its large size. 
  https://en.wikipedia.org/wiki/Large_language_model
- Successor to word n-gram language models.
- Uses:
  - Sentiment analysis
  - Named Entity Recognition (NER)
  - Text generation and summarization
  - Natural language understanding (NLU)
  - Code generation
  - Conversational AI
- BERT, GPT-4, ChatGPT, PALM, LLaMa, etc.

Term “LLM” seems to have started after GPT-2.

# Overlap


LLMs

* Not meant to be an exhaustive list and will likely be out of date fast…

# Today

We see LLMs + Python ecosystem can do everything Apache OpenNLP can, plus some more.


So why consider using Apache OpenNLP?

# Let’s go back in time

- Python was the language of choice for the explosion of NLP in the late 2010s.
- But what happened? Its popularity has led to the belief that the “P” in NLP stands for Python. ***/s***


# Let’s Compare

Named-entity recognition

Disclaimer - An unscientific test.

But should be sufficient for highlighting capabilities.

Just one experiment and one use-case!

# Training Data - the hardest part, right?

- Used the multiNERD dataset - https://huggingface.co/datasets/Babelscape/multinerd
- Has multiple languages and entities, but just used English text and person entities in my training set
- This subset was converted to OpenNLP's training format - https://github.com/jzonthemtn/opennlp-formats

```
<START:person> John Smith <END> is a person.
```

# Notable Training Parameters

**Apache OpenNLP**

10 iterations (passes of the training data)
1 cutoff (each token must be seen once)

(Default parameters for first time training a model.)

**SpanMarker (Python)**

1 epoch
Learning rate 0.00005

(Default parameters from the git repository - no extra tuning was done.)

This is ***not*** an evaluation or critique of the SpanMarker library. It's an excellent Python NLP library with LLM support and a great choice for your NLP Python app.

# Training NER Model using OpenNLP and Python SpanMarker

https://github.com/jzonthemtn/opennlp-formats/blob/main/src/main/java/com/github/jzonthemtn/opennlp/TrainTokenNameFinder.java

```
ObjectStream<NameSample> sampleStream = new
NameSampleDataStream(new
PlainTextByLineStream(in,
StandardCharsets.UTF_8));

TrainingParameters params = new
TrainingParameters();

params.put(TrainingParameters.ITERATIONS_PARAM,
10);
params.put(TrainingParameters.CUTOFF_PARAM, 1);

TokenNameFinderModel nameFinderModel =
NameFinderME.train("en", null, sampleStream,
params, TokenNameFinderFactory.create(null,
null, Collections.emptyMap(), new BioCodec()));
```

https://huggingface.co/tomaarsen/span-marker-mbert-base-multinerd/blob/main/train.py

```
trainer = Trainer(
       model=model,
       args=args,
       train_dataset=train_dataset,
       eval_dataset=eval_dataset,
)

trainer.train()
trainer.save_model("models/span_marker_mbert_base_multinerd/checkpoint-final")
test_dataset = load_dataset(dataset, split="test")

# Compute & save the metrics on the test set
metrics = trainer.evaluate(test_dataset,
metric_key_prefix="test")
trainer.save_metrics("test", metrics)
trainer.create_model_card(language="multilingual",
license="apache-2.0")
```

# Named-entity Extraction Results Comparison

**OpenNLP**

184 seconds

Model training took 184794 ms

[main] INFO opennlp.tools.ml.perceptron.PerceptronModelWriter - Compressed 3163084 parameters to 108347
[main] INFO opennlp.tools.ml.AbstractMLModelWriter - 4 outcome patterns
Model saved to /home/ubuntu/ner-multinerd.bin

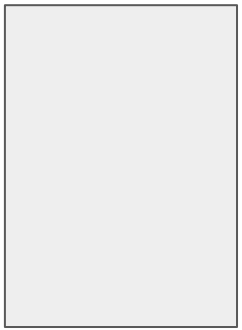
**BERT-based Model**

Language

Precision

Recall

F1



Training time **1.86 hours**

# Named-entity Extraction Results Comparison

**OpenNLP**

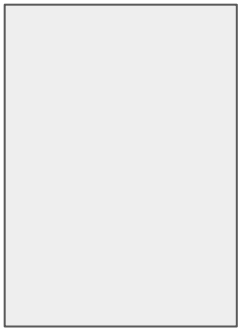
Model training took 184794 ms

[main] INFO opennlp.tools.ml.perceptron.PerceptronModelWriter - Compressed 3163084 parameters to 108347
[main] INFO opennlp.tools.ml.AbstractMLModelWriter - 4 outcome patterns
Model saved to /home/ubuntu/ner-multinerd.bin

Precision: 0.9388020833333334

**BERT-based Model**

| | |
|---|---|
| Language | en |
| Precision | **99.46** |
| Recall | |
| F1 | |
| Training time | |



# Named-entity Extraction Results Comparison

**OpenNLP**

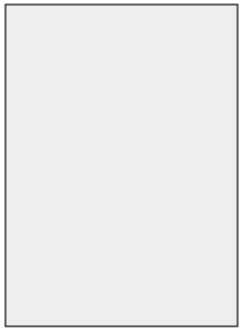
Model training took 184794 ms

[main] INFO opennlp.tools.ml.perceptron.PerceptronModelWriter - Compressed 3163084 parameters to 108347
[main] INFO opennlp.tools.ml.AbstractMLModelWriter - 4 outcome patterns
Model saved to /home/ubuntu/ner-multinerd.bin

Precision: 0.9388020833333334
Recall: 0.9585944919278253

**BERT-based Model**

| | |
|---|---|
| Language | en |
| Precision | 99.46 |
| Recall | **99.52** |
| F1 | |
| Training time | |



# Named-entity Extraction Results Comparison

**OpenNLP**

Model training took 184794 ms

[main] INFO opennlp.tools.ml.perceptron.PerceptronModelWriter - Compressed 3163084 parameters to 108347
[main] INFO opennlp.tools.ml.AbstractMLModelWriter - 4 outcome patterns
Model saved to /home/ubuntu/ner-multinerd.bin

Precision: 0.9388020833333334
Recall: 0.9585944919278253
F-Measure: 0.9485950568555587

**BERT-based Model**

| | |
|---|---|
| Language | en |
| Precision | 99.46 |
| Recall | 99.52 |
| F1 | **99.46** |
| Training time | 1.86 hours |

# Comparisons

| | OpenNLP | BERT-based Model |
|---|---|---|
| **Training Time** | **185 seconds** | 1.86 hours |
| **Precision** | 0.9388 | **0.9946** |
| **Recall** | 0.9586 | **0.9952** |
| **F1** | 0.9486 | **0.9946** |
| **Cost** | AWS t4g.large @ $0.0672/hr (2 vCPU / 8 GB RAM)<br><br>= **$0.0034** | AWS g5.xlarge (NVIDIA A10G) @ $1.19/hour<br><br>$1.212 * 1.86 = $2.25 |

**36x more time**

**+~5%**

**662x more expensive**

* Again, not a rigorous scientific test. Some things could probably be optimized.
** AWS EC2 minimum billing is 60 seconds.

# Comparisons

| | OpenNLP | BERT-based Model |
|---|---|---|
| **Training Time** | **185 seconds** | 1.86 hours |
| **Precision** | 0.9388 | **0.9946** |
| **Recall** | 0.9586 | **0.9952** |
| **F1** | 0.9486 | **0.9946** |
| **Cost** | AWS t4g.large @ $0.0672/hr (2 vCPU / 8 GB RAM)<br><br>= **$0.0034** | AWS G5.xlarge (NVIDIA A10G) @ $1.19/hour<br><br>$1.212 * 1.86 = $2.25 |

The purpose is ***not*** to say one is always better than the other.

The goal is to highlight Apache OpenNLP's role in today's LLM-dominated NLP world.

# So, what does it mean?

- What's more important to you? Training/eval time? Cost? Precision?
- Is the ~0.4 increase in precision worth 662x the cost? 36x the time?
- What's your current stack?
- Current architecture? Future plans?


Is $2.25 significant?
At scale, it may be.

# Is the cost significant?

Maybe.

- Does your model use a larger training data set and take longer to train?
- Do you need to retrain the model frequently due to model degradation?
- Do you need multiple models?
  - Separate models per language?
  - Separated by entity types?

The cost difference may become significant.


Is $2.25 significant?
At scale, it may be.

# Model Inference Times

- Both have low inference timing.
- Apache OpenNLP does not need a GPU.
- Apache OpenNLP needs fewer resources, in general.
  - Model file sizes are much smaller. A few KB vs. hundreds of MBs.
  - Smaller CPU, memory requirements.
- Same takeaways - is the cost significant? Maybe, depends on your needs.

# But what about us Java devs and LLMs?

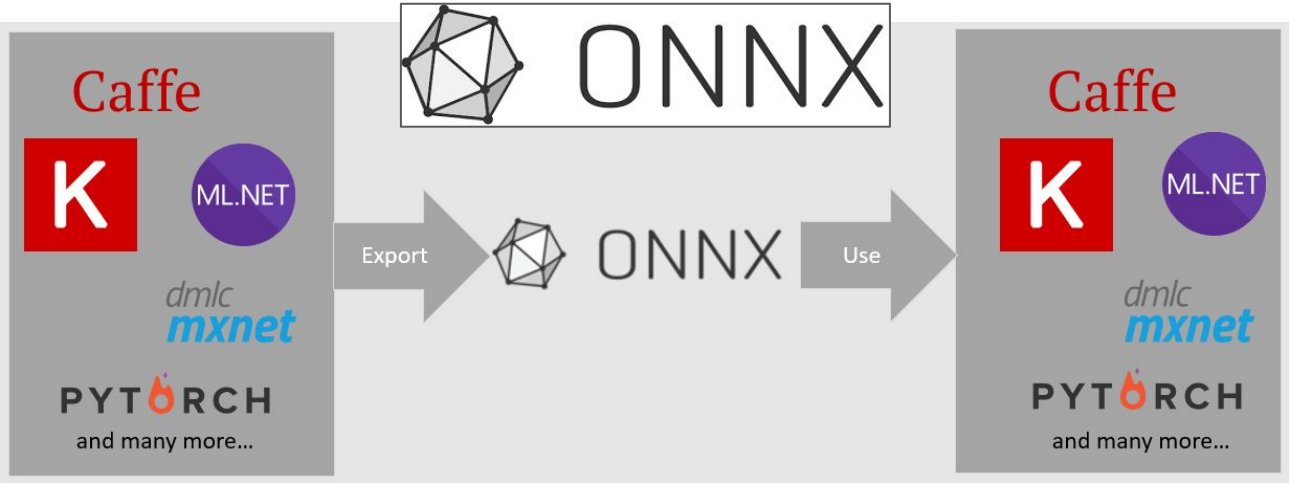
- You might have a valid reason to use LLMs (fine-tuning, etc.).
- But you want to do inference from a JVM app.
- What can we do?

# OpenNLP 2.0 and ONNX Runtime

- OpenNLP 2.0 introduced support for ONNX Runtime.
- Can train a model in Python, convert it to ONNX, and do inference using OpenNLP. Python folks can stay in Python, Java folks can stay in Java, and the model can be served directly from Java - no new services required.
- OpenNLP’s support for ONNX Runtime can use NER, document classification, and sentence embedding generation models.

Currently a work-in-progress to support other NLP tasks via ONNX Runtime. Want to help? :)

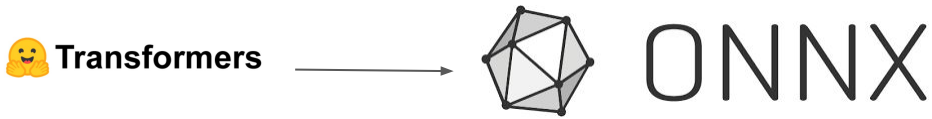
# ONNX and the ONNX Runtime




Convert sequence classification model to ONNX:

```
# python3 -m transformers.onnx -m nlptown/bert-base-multilingual-uncased-sentiment –feature sequence-classification exported
```

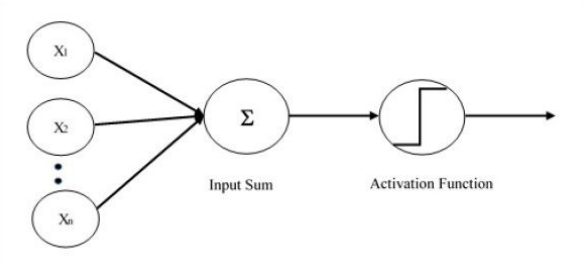
https://huggingface.co/docs/transformers/serialization



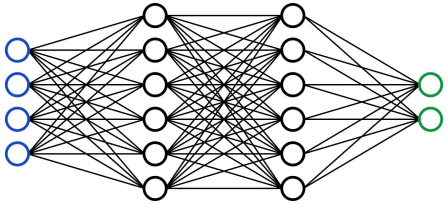
# Rock[et] Solid

OPEN NLP™

Java



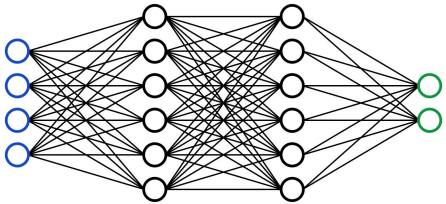

Transformers



# Using a PyTorch Model from OpenNLP




Transformers



[U.S. Navy photo by Photographer's Mate 2nd Class Anthony Koch.]

# Model Inputs

```
final Map<String, OnnxTensor> inputs = new HashMap<>();
inputs.put("input_ids", OnnxTensor.createTensor(env, LongBuffer.wrap(tokens.getIds()), new long[]{1, tokens.getIds().length}));
inputs.put("attention_mask", OnnxTensor.createTensor(env, LongBuffer.wrap(tokens.getMask()), new long[]{1, tokens.getMask().length}));
inputs.put("token_type_ids", OnnxTensor.createTensor(env, LongBuffer.wrap(tokens.getTypes()), new long[]{1, tokens.getTypes().length}));
```

# Model Outputs

```
final float[][][] vectors = (float[][][]) session.run(inputs).get(0).getValue();
```

Now, just go through the 3d array to find the highest scoring label for each token!

*George Washington was president*

**B-PER I-PER O O**


opennlp.tools.namefind

## Interface TokenNameFinder

**All Known Implementing Classes:**

DictionaryNameFinder, NameFinderME, RegexNameFinder

---

public interface TokenNameFinder

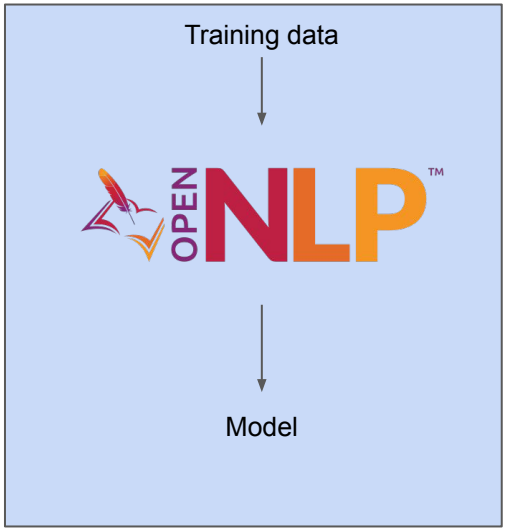
The interface for name finders which provide name tags for a sequence of t

```
public class NameFinderDL implements TokenNameFinder {

    final TokenNameFinderInference inference;

    public NameFinderDL(File model, File vocab, boolean doLowerCase, Map<Integer, String> ids2Labels

        inference = new TokenNameFinderInference(model, vocab, doLowerCase, ids2Labels);

    }

    @Override
    public Span[] find(String[] tokens) {

        try {

            final double[][] v1 = inference.infer(String.join(" ", tokens));

            //System.out.println(Arrays.toString(v1));
            //return results;

        } catch (Exception ex) {
            System.err.println(ex.getMessage());
        }

        System.out.println("Returned null - something wrong");
        return null;

    }
```

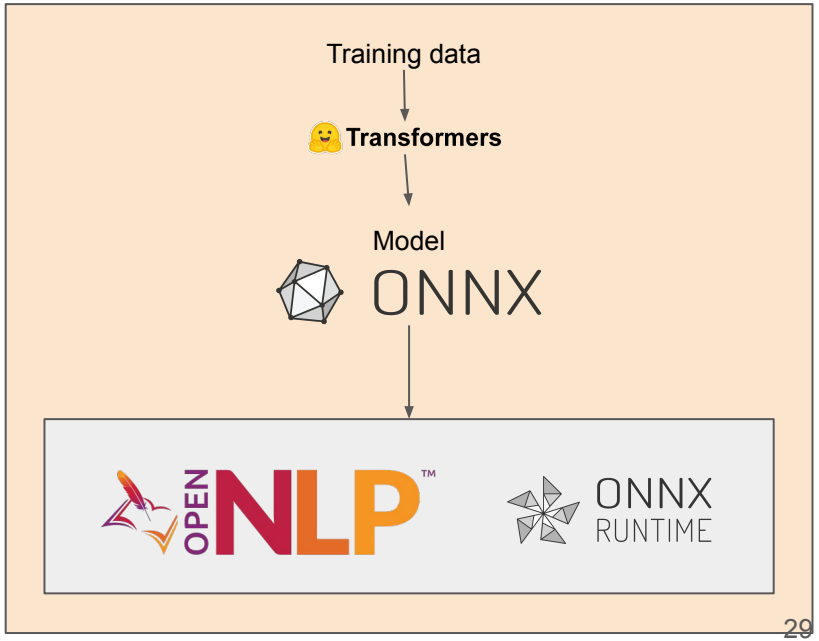
**Implements the existing OpenNLP interfaces!**

# Summary - Two Choices - How OpenNLP Fits in with LLMS

1 Apache OpenNLP is still a solid choice for NLP tasks. Model training is fast, doesn't require a GPU, and can be done with minimal cost.




2 Can use LLMs with OpenNLP via ONNX Runtime for some NLP tasks.



# Thanks!

Jeff Zemerick
jzemerick@apache.org
https://jeffzemerick.dev

If you want to get involved, the Apache OpenNLP team would love to have you!
https://opennlp.apache.org/get-involved.html