PII TOOLS

Detecting personal names in texts

Technical whitepaper

PII TOOLS

Detecting personal names in texts

Introduction

Detecting personal names is an integral part of PII discovery. Traditional techniques like regexps and keywords don't work, as the set of every possible name is too large. Therefore, a good Named Entity Recognizer (NER) is essential for detecting names, home addresses, passport scans, etc., for purposes of compliance or breach incident management. One could use a prepared list of names and surnames for such tasks, however, this approach obviously cannot be exhaustive and will fail on sentences such as:

"Calvin Klein founded Calvin Klein Company in 1968."

Humans can easily distinguish a person's name. For machines, the task is more difficult because they have trouble understanding the context. This leads to two well-known types of errors:

- **False positives:**
  words detected as personal names, typically because they're capitalized. Example:

  "Skin Games and Jesus Jones were two of the first western bands."

- **False negatives:**
  personal names missed by the detection process, typically because they're lowercased, foreign, or uncommon. Example:

  "bill carpenter was playing football."

In order to recognize personal names in text, it is necessary to know what names look like as well as what context they're used in. Another prerequisite is general domain knowledge.

Technical details of PII Tools' neural network

The creation of NER

NER is a well-studied task in academia. Naturally, we turned to open-source NER solutions first and evaluated the most popular ready-made software: Stanford NER and Stanza from Stanford University, FLAIR from Zalando Research, spaCy from Explosion AI.

None of these open-source tools were precise or fast enough for our purposes. While they work great on well-behaved data, such as news articles or Wikipedia, they fail when applied to the wild, messy documents of the real world. For this reason, we developed our own NER, with a special focus on **names as they appear in real company documents.**

This whitepaper compares our NER against other popular open-source options. Additionally, we'll benchmark a simple gazetteer-based NER that uses a predefined list of names to serve as a **baseline.**

Description of tested NERs

	version	type	language	source
list	-	list of names	multi-language	in-house
Stanford	4.1	CRF classifier	single	https://nlp.stanford.edu/software/CRF-NER.html
Stanza	1.1.1	neutral network	single	https://stanfordnlp.github.io/stanza/
Flair	0.6.1	neutral network	multi-language	https://github.com/flairNLP/flair
SpaCy	2.3.2	neutral network	multi-language	https://spacy.io/
PII Tools	3.8.0	neutral network	multi-language	https://pii-tools.com/

The requirements for our new NER:

- **Multi-language** – with a special focus on English, Spanish, German, and Brazilian Portuguese.
- **Accepting arbitrary document contexts** – texts from PDFs, Word documents, Excel, emails, database fields, OCR, etc.
- **Accuracy** – to minimize false positives and negatives.
- **Efficiency** – to process **large amounts of text quickly using a CPU** (no need for specialized GPU hardware) and with a low memory footprint.
- **Flexibility** – to be able to evolve the model behavior: retraining, adding new languages, and correcting detection errors.

To compare our NER, we evaluated its performance on standard benchmark datasets: CoNLL-2002 (Spanish), CoNLL-2003 (English and German), and LeNER-Br (Brazilian Portuguese).

We also included a manually annotated OpenWeb dataset to ensure we test on data that no NER system (including our own) has seen during its training. Text for this dataset was randomly sampled from the English OpenWebTextCorpus. We value OpenWeb results the most because, among these public datasets, OpenWeb most accurately reflects the real (messy) data found in actual documents.

How NOT to build a practical NER

The most effortless way to make an NER is to grab an existing pre-trained model, and just use that. If you've ever tried this, however, you most likely quickly found out that, in practice, this leads nowhere – the models are too brittle, trained on artificial datasets, with narrow domain-specific preprocessing. They fail when applied to real documents in the wild. This approach is useful mostly for leaderboards and academic articles, where people compare the accuracy of different model architectures on a **fixed, static dataset** (the so-called "state-of-the-art" or "SOTA dataset").

The second popular way to create a NER is to fetch an already prepared model and tune it to work with your own data. For the NER task of *"I give you text, you give me the positions of every personal name in the text"*, training data is scarce and usually less-than-diverse. The state-of-the-art models are transformers, a special kind of neural networks that is quite slow and memory intensive (although there is active research to make them faster). Manually labeling huge amounts of text is costly too. It's safe to say, this wasn't the right path for us.

Bootstrapping a NER

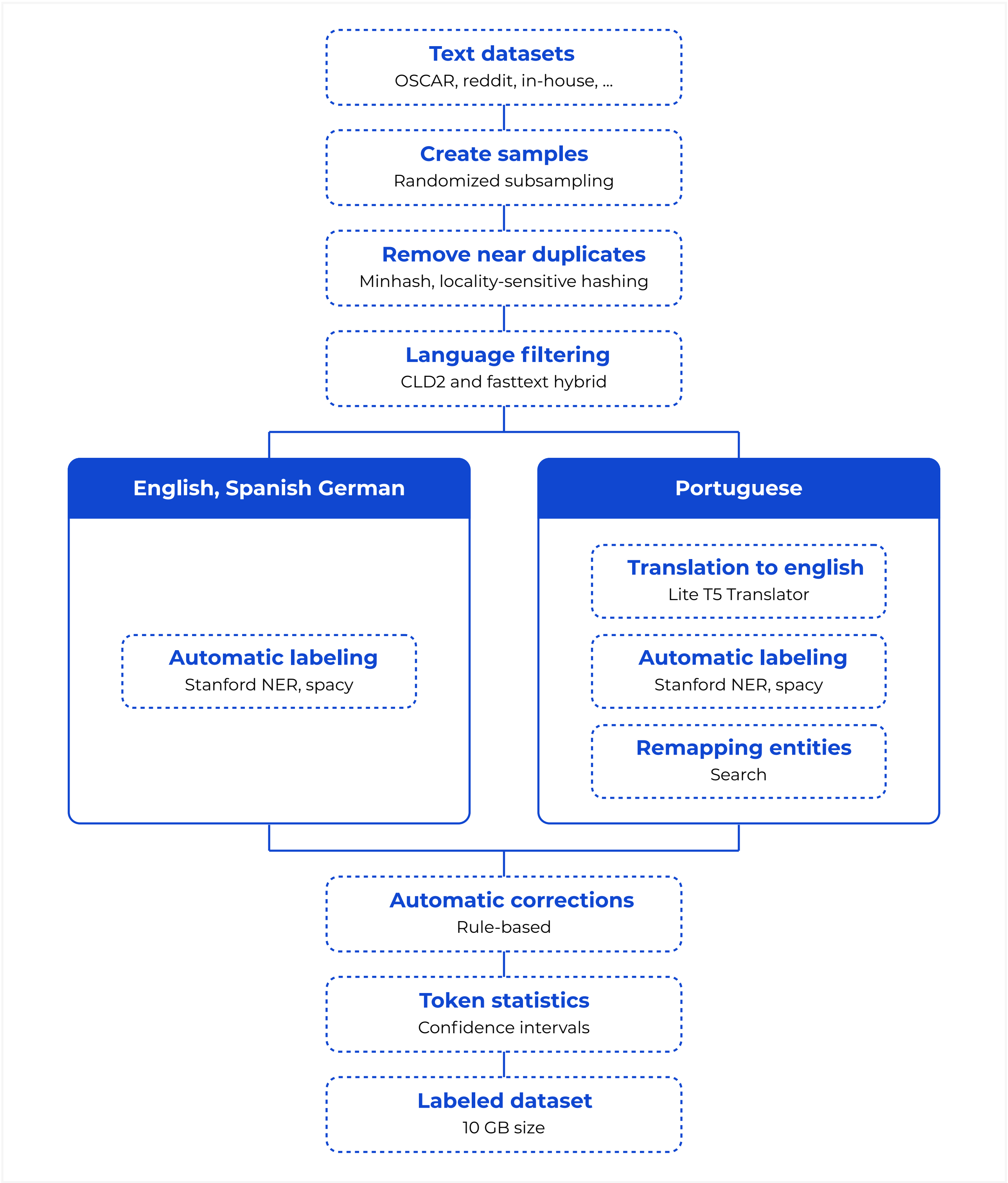
Our approach is incremental, bootstrapping from existing datasets and open-source tools:

1. Prepare a large and diverse, **unlabeled** dataset automatically.
2. **Annotate** the corpus automatically using multiple fast, yet low-accuracy, open-source tools.
3. **Correct** annotation errors semi-automatically.
4. **Train** a stand-alone NER pipeline using Tensorflow with convolutional neural networks.
5. Optimize NER for **production**: squeeze model size and performance with Tensorflow Lite, DAWG, mmap, and other tricks.

Let's go over these steps one by one.

1. Diverse dataset

Garbage in, garbage out, thus making the quality of training data essential. We spent a lot of effort here, building a powerful pipeline and combining automated and manual labeling in a feedback loop:



PII Tools' pipeline for building an annotated NER dataset.

To include the kinds of contexts that could contain personal names, we use various text sources:

- OSCAR corpus: a large set of crawled text from web pages by Common Crawl.
- Reddit: forum comments from reddit.com.
- Gutenberg: books from the Gutenberg Project.
- Enron emails: published corporate emails.
- WikiLeaks: documents and articles on various topics.
- Extensive gazetteers of personal names and cities. Cities (home addresses) and personal names appear in similar contexts, with similar formatting, and are a common source of confusion for NER models in general. In other words, paying close attention to cities and addresses pays off in name detection.
- In-house dataset of documents, a proprietary part of PII Tools, including Excel spreadsheets, PDFs (incl. for OCR), Word, and various corner-case formats and contexts.

After resampling and deduplicating, we filtered the texts down to only those containing the languages we care about: English, German, Portuguese, and Spanish to begin with.

One might think that identifying language in texts is an easy problem to solve. However, just as with NER, existing tools have large issues across different languages, such as slow speeds or low accuracy. You can find more information about this topic in Radim's paper on Language Identification.

After internal testing and evaluating the available solutions, we ended up creating a hybrid of a simple & fast n-gram language detector based on CLD2 and fastText language identification. This is what we used for the "corpus language filtering" mentioned above.

2. Semi-supervised annotation

The next part of the dataset building process is to automatically annotate the raw corpora above. That is, mark the position of personal names in each corpus document. We use existing NER engines for this initial bootstrap step. This technique is sometimes called "weak labeling".

Basically, Stanford NER and spaCy were the only viable options here, given that the dataset was quite large. Even for these two relatively fast libraries, it took several days to annotate the entire corpus.

Now we had a large amount of text annotated with personal names. However, it contained a lot of mistakes – neither Stanford's NERs nor spaCy's out-of-the-box models work particularly well outside of their trained domain. Another step was to clean the data and remove the obvious (classes of) errors.

This made for a bit of a manual task. We had to go through a subset of NER results, identify the most common patterns of errors, write transformation rules, and re-annotate. For instance, no "Ben & Jerry" is not a person, and "Stephen Crowd carpenter" is a person's name with occupation slapped on the end. These and other similar mistakes are easily removed in bulk once we get to know them, using our superior "biological neural network" AKA our brain.

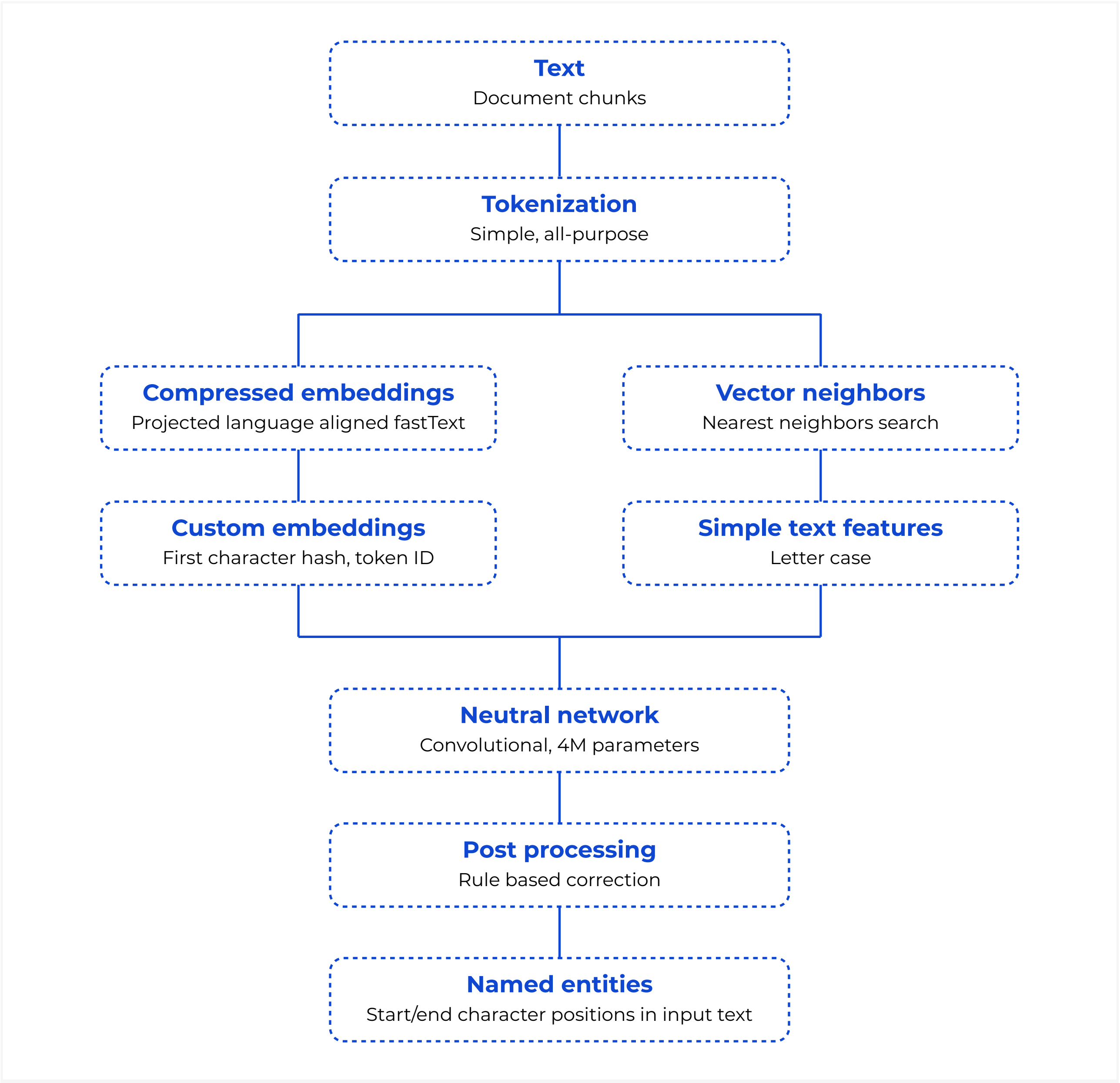
We use this dataset preparation strategy for English, Spanish, and German. For Brazilian Portuguese, we had to resort to a slightly different approach, since we found no existing NER with sufficient enough performance. Instead, we translated Portuguese texts with the Lite-T5 translator to English, recognized named entities with the Stanford NER and spaCy, then simply remapped them back into the original text. The automatic translation took a few weeks but required zero manual work.

In the end, we built 10 GB (20M training samples) of annotated text, with surprisingly high quality considering it was produced mostly automatically.

3. Neural network model

PII Tools is written in Python, and so is the model pipeline. We used Tensorflow to define and train our production model on the annotated corpus above. The pipeline uses several kinds of features to maximize accuracy.

As mentioned earlier, we used a lightweight convolutional neural network as the heart of our NER model. The model accepts a text, tokenizes it to a sequence of tokens (token positions), and spits out a sequence of name positions. A position is an offset within the original document, so we can exactly match and remediate detected names later.



Inference on a trained model.

We had a bad experience with off-the-shelf tokenizers. They usually try to be "smart" about tokenization but fail in many cases because there is not enough space to be smart in this component. We had the greatest success with a simple tokenizer that recognizes 4 token groups: alphabetic sequences, whitespace, digits, and other characters. Trying to be smarter than that always provided worse results.

4. Input Features

Another critical point here is how to choose token features. Features have to be fast to compute and as informative as possible. The best performance-information trade-off was achieved by combining:

- simple "letter case" features;
- precomputed language aligned fastText vectors, projected with umap to a lower dimension;
- trained short token embedding to cover the tokens not in precomputed vectors;
- trained vector embedding of the first character of a token to differentiate between token groups and to differentiate characters inside the "other" token group.

For the pre-trained embedding vectors, we have to keep a dictionary of relevant tokens and their vectors, which could potentially be quite large, consuming too much RAM. Thus, for each language, we picked a subset of:

- the most frequent tokens,
- tokens with the highest probability to be a part of a name (as calculated on the training dataset), and
- tokens with the tightest confidence interval to be a part of a name.

Furthermore, we included another set of tokens whose vectors were in the close neighborhood of the tokens identified above, and assigned them the vector of their respective neighbor. This optimization trick increased our dictionary size 2-4x, depending on the language, while costing negligible extra space.

We then massaged features through a convolutional network with standard 1D convolutions and leaky ReLU activation units. By deepening the network to +10 layers, we can achieve a reception field of +20 tokens, which is enough for this task. This way we avoided costly LSTM, GRU, or transformer's attention units. Outputs of the neural networks are tags indicating whether the current token is the beginning, middle, or end of a person's name, or a non-name token.

5. Post-processing

To ensure the model output is reasonable, we apply a few lightweight post-processing steps, such as ignoring person names that do not contain any statistically strong name token, or where all tokens are common words. For this, we leveraged the corpus statistics again.

Such a "corrected" output can be fed back into the training corpus, so that the NN model learns not to make this type of mistake again. In practice, we find that having post-processing steps is highly desirable anyway, both as a quick approximation of a retrained model and also as a fail-safe paradigm for data patterns that CNNs have trouble picking up from a limited number of training samples.

6. Model minimization

Accuracy is only one of our design goals. Being practical in terms of computational speed (CPU) and memory footprint (RAM) is equally important.

Neutral network model
TensorFlow Lite, 16 MB

Embeddings
umap-projected fastText, 45 MB

Neighbors dictionary
4.5 MB

Common tokens
1 MB

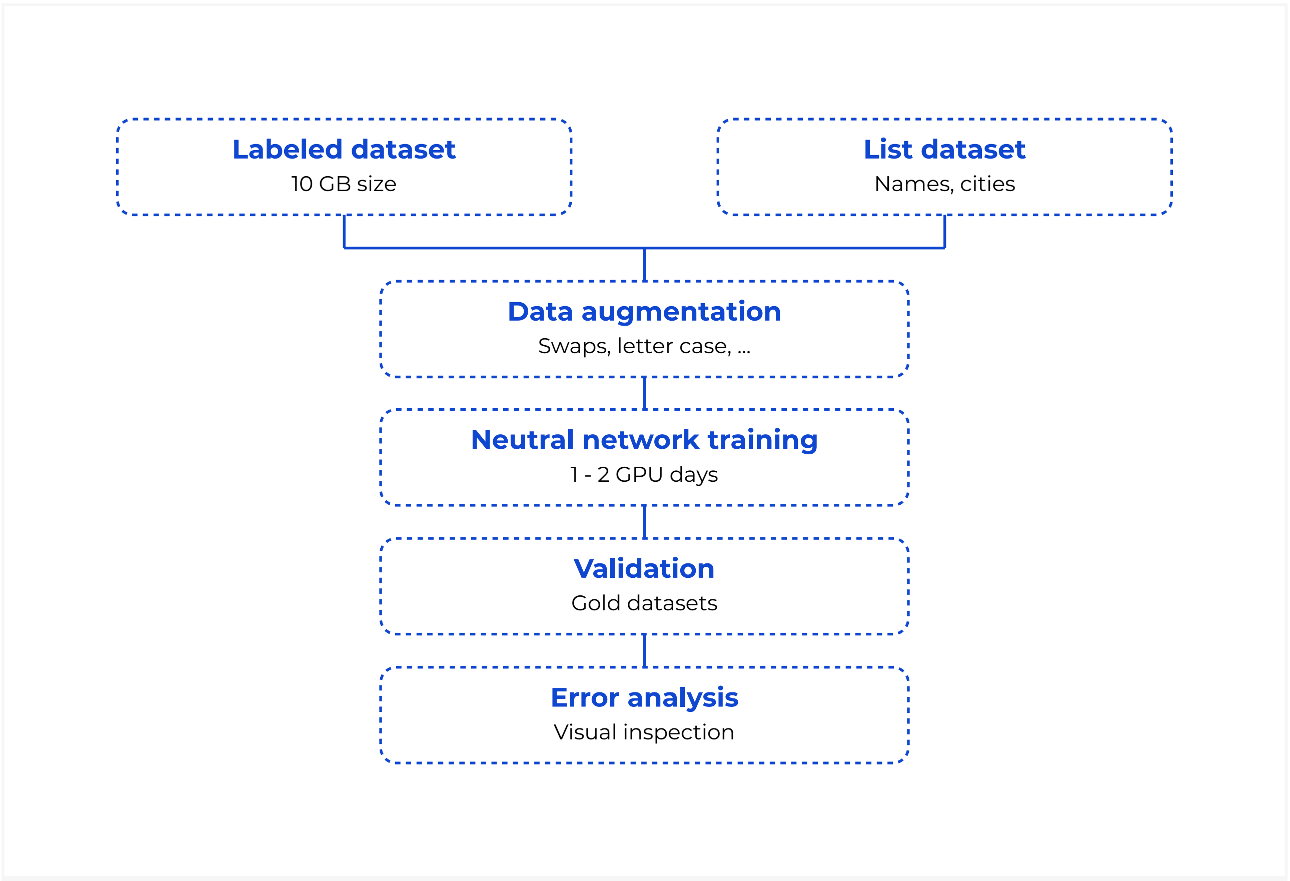
The entire trained model for the four target languages takes up about 70 MB of disk space, and slightly more after being loaded in RAM. Parts are shared in RAM between individual scanner processes using MMAP.

To compress the necessary model dictionaries and save even more RAM and CPU, we use DAWG, an excellent FSA-based storage. Our (versioned) model representation takes up roughly 70 MB of disk space, which includes:

- the trained Tensorflow Lite convnet model,
- precomputed vector embedding for 300k+ tokens, for all the languages combined,
- a token dictionary, and
- the corpus token statistics.

PII Tools uses parallelization for performance, so some of these data structures are shared in RAM between worker processes using mmap. This allows further memory reduction on heavy-load systems with many scan workers.

7. Training and optimization



NER traning pipeline.

Training the whole model takes 1 to 2 days on low-end server hardware with 8 CPUs and 1 GPU. 10 GB of training data have to be processed while applying data augmentation on the fly.

A few simple tricks help with dataset diversification and resulting model robustness. One is swapping letter casing – making all the words lowercase, or swapping the letter case.

Another trick we found useful is to inject more information into training with multitasking. The neural network is not only trained on predicting token tags, but it also has to guess the language of the text in the receptive field, guess how likely a computed token is inside a name, and predict tags for the tokens that are close but not exact. This acts as a regularizer for the training and the model internalizes additional information without needing to explicitly provide it on input.

Comparing PII Tools' NER with other solutions

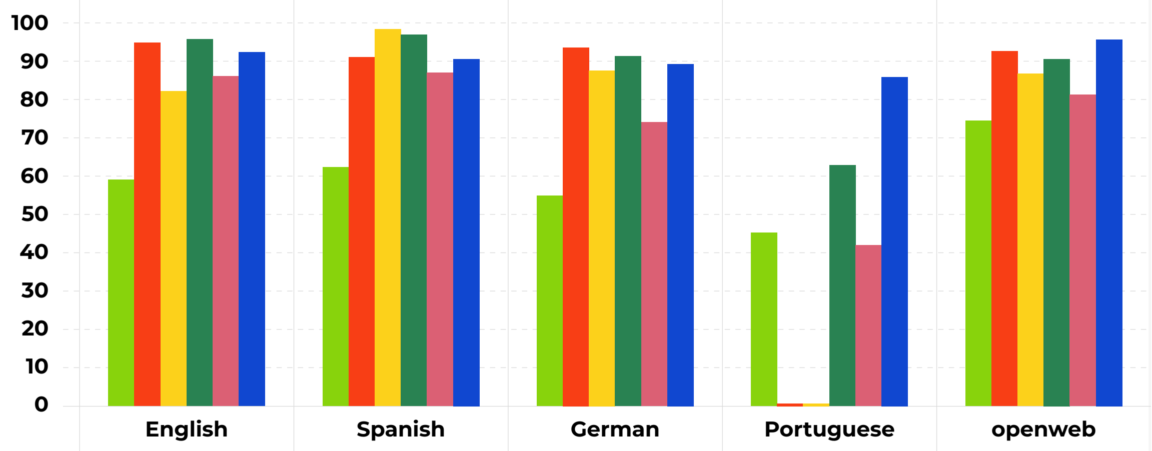
Accuracy

We measured F1 scores for personal names detected by each software. F1 scores range from 0 to 100, the higher the better. A "hit" (true positive) means the entire name was matched exactly, from the beginning to the end.

This is the strictest metric; we specifically did not calculate success over the number of correctly predicted tokens nor the individual characters.

Performance benchmark [F1 score – higher is better]

	English	Spanish	German	Portuguese	openweb
🟩 **list**	59.1	62.3	54.9	45.2	74.5
🟥 **Stanford**	94.9	91.1	93.6	.....	92.6
🟨 **Stanza**	82.2	98.4	87.6	.....	86.7
🟩 **Flair**	95.8	96.9	91.3	62.9	90.6
🟥 **SpaCy**	86.1	87.0	74.1	42.0	81.3
🟦 **PII Tools**	92.3	90.6	89.3	85.9	95.7



F1 scores of NEW software that detect person names

In all tests, our NER is among the best performers, however, keep in mind that accuracy was just one of our 5 design goals. For example, we "lost" a few F1 points on purpose by switching from Tensorflow to TF lite, trading accuracy for a **much smaller and faster model**.

For the Brazilian Portuguese (LGPD) and OpenWeb dataset, PII Tools is the clear winner. As mentioned above, OpenWeb reflects "real data" the most, so this is a huge achievement. Let's take a look at a few examples:

Text samples and NER detections (actual expected names are in **bold**)

	Stanford	Stanza	Flair	SpaCy	PII Tools
Calvin Klein founded Calvin Klein Company in 1968.	ok	ok	ok	ok	ok
Nawab Zulfikar Ali Magsi did not see **Shama Parveen Magsi** coming.	ok	ok	ok	fail	ok
bill carpenter was playing football.	fail	fail	fail	ok	ok
Llanfihangel Talyllyn is beautiful.	fail	fail	fail	fail	ok
Skin Games and Jesus Jones were one the first western bands.	fail	fail	fail	fail	fail

These samples are rather far towards the edge, however, they serve in gaining an idea of what detectors have to deal with and how successful they are.

Performance

Now let's examine other aspects of our requirements. Detection performance was measured on a CPU, our PII Tools NER was artificially restricted to a single process with a single thread, and the others were left on the default settings.

Speed and memory comparison

	speed short [kB/s]	speed long [kB/s]	startup time [s]	RAM usage [MB]
list	350.0	1100.0	1.1	1420
Stanford	30.0	30.0	1.5	248
Stanza	0.4	1.2	6.1	1073
Flair	0.1	0.1	5.2	5341
SpaCy	70.0	200.0	0.5	123
PII Tools	35.0	230.0	1.3	387

Overall, FLAIR and Stanza are definitely out due to their super slow speeds and high RAM usage. A worthy competitor from the performance perspective is spaCy, whose authors put a great deal of effort into optimization. Unfortunately, spaCy's tokenization quirks and opinionated architecture proved too inflexible for our needs.

Likewise, the Stanford NER is the most accurate among the open-source alternatives, but it's quite rigid – it's incredibly difficult to update its models or add a new language. Plus, its GNU GPL license won't be to everyone's liking.

Flexibility

Our focus on industry use calls for frequent model modifications: adding new languages, new kinds of documents (document contexts in which names may appear), fixing detection errors.

In order to adapt the PII Tools NER model quickly, we built a pipeline that utilizes several "weaker" NERs and automatic translation tools to create a huge training corpus from varied sources. This focus on real-world data, along with a robust automated Tensorflow training pipeline, allows for adding new languages and controlling NER outputs more easily than the open source solutions.

While developing the PII Tools NER, we implemented most components from scratch, including:

- a large scale annotated dataset (proprietary data)
- a tokenizer (critical; none of the open-source variants do this part well)
- token features (input to the neural network)
- convolutional neural network (NN architecture)
- data augmentation and training pipeline (for grounded model updates)
- various tricks to push performance or to squeeze a lot of information into a smaller parameter space

About PII Tools

Our mission

The mission of PII Tools is to bring modern automation technology into the world of data privacy.

We equip infosec, legal and data protection teams with accurate and easy-to-use software for discovery and remediation of sensitive data.

What PII Tools brings to the game is clarity and quality, both in technology and in the business model. Our context-aware PII detectors come from years of cutting edge research into Artificial Intelligence.

Why did we start PII Tools?

- Clarity and honesty. AI is quickly becoming a vacuous buzzword. As an AI-first company and world-leading PhD experts in machine learning, we think the cyber security world deserves better.
- Focus on technology. Intelligent automation doesn't have to be a marketing after-thought. We bring over a decade of research excellence, award-winning open source software used by thousands of companies world-wide, and over a thousand peer-reviewed citations.
- Passion for pragmatic solutions. With a strong background in IT services, we know how to listen to customers and combine quality engineering with cutting edge research.

Our history

RARE Technologies Ltd: providing industry excellence since 2012

- **MARCH 2010**
  First public release of Gensim, our open source NLP software.
- **MAY 2014**
  Incorporated UK office in Bristol, UK
- **OCTOBER 2015**
  Launch of corporate Machine Learning and NLP trainings
- **JANUARY 2017**
  Office opening in Brno, CZ
- **JANUARY 2018**
  Launch of PII Tools
- **AUGUST 2024**
  PII Tools redeveloped and sold by PII Tools Limited, a Cyprus company
- **NOW**
  Serving customers with superior PII Discovery world-wide

Contact us

Email:
info@pii-tools.com

Website:
https://pii-tools.com

Address:
25 Spyrou Araouzou, Berengaria 25, 3036 Limassol, Cyprus

PII Tools is a product of PII Tools Ltd.