Implementation of POS-tagging through Hidden Markov Model and Viterbi decoding algorithm

Alberto Cardini[a]

[a]Università degli studi di Firenze, Bachelor of Science in Computer Engineering

This study presents a supervised Hidden Markov Model (HMM) for Part-of-Speech tagging, focusing on parameters learning, Out-Of-Vocabulary handling, additive smoothing and Viterbi decoding. Benchmarks indicate that excessive smoothing degrades performance, while qualitative tests confirm the model's ability to master syntactic structures, even on nonsense words. However, results also highlight limitations in semantic disambiguation and susceptibility to dataset biases due to first-order Markov assumptions.

The POS-Tagging problem concerns how we can teach a computer to perform the grammatical analysis of a sentence. The supervised HMM-based statistical model described and implemented in this report takes as input a sentence of length N and returns a tag sequence of the same length. Along with the Viterbi algorithm, used to decode the sequence, the learning routine is also explained.

1. Theory of Hidden Markov Models

A Hidden Markov Model (HMM) is a probabilistic model for sequential data in which the system is assumed to evolve over time through a finite set of hidden states (tags), forming a Markov chain, while each state generates an observable output (words) according to a state-dependent probability distribution. An HMM is defined by 5 components (Rabiner 1989):

- $T = \{t_i\}_{i=1,...,N}$: a set of N states.
- $O = \{o_i\}_{i=1,...,M}$: a set of M observations.
- $A = \{a_{ij}\}$: transition probability matrix.
- $B = b_i(o_t)$: emission probability matrix.
- $\pi = \{\pi_i\}_{i=1,...,N}$: initial state probability distribution.

Calculation of these quantities is simplified by two assumptions:

- First-order Markov assumption:

$$P(t_i \mid t_1, ..., t_{i-1}) = P(t_i \mid t_{i-1}) \quad [1]$$

- Output Independence:

$$P(o_i \mid t_1, ..., t_T, o_1, ..., o_T) = P(o_i \mid t_i) \quad [2]$$

Thus:

$$a_{ij} = P(t_i \mid t_j) \quad [3]$$
$$b_i(o_j) = P(o_j \mid t_i) \quad [4]$$

In POS tagging, a_{ij} represents the probability of observing tag i given that the previous tag was j. For example, $a_{\texttt{NN,DT}} = P(\texttt{NN} \mid \texttt{DT})$ denotes the probability of encountering a noun immediately after a determiner. This assumption restricts the model to local contextual information, since each tag decision depends only on the immediately preceding tag. As we will see later, the model may fail to correctly disambiguate lexically ambiguous words when their correct tag depends on long-range syntactic or semantic context.

Moreover, $b_i(o_j)$ represents the probability of tag i generating word j. For instance, $b_{\texttt{DT}}(the) = P(the \mid \texttt{DT})$ is the probability of finding *the* given that we have in our hands the tag `DT`.

2. Learning procedure

Rabiner's 1989 third problem when using an HMM λ to infer a sequence is about finding A, B and π in order to maximize $P(O \mid \lambda)$.

The training phase aims to resolve this problem by using a dataset. In this particular case, we use a fully observed dataset composed by several sentences where each word is annotated with the right tag. Such dataset allows us to easily compute both [3] and [4]. We find the maximum likelihood estimate of the transition probability by counting, out of the times we see the first tag in a labelled corpus, how often it is followed by the second:

$$P(t_i \mid t_j) = \frac{\#(t_i, t_j)}{\#(t_j)} \quad [5]$$

And the emission probability is found by computing the number of times that the given word is annotated with such tag over that tag's overall occurrence in the dataset:

$$P(o_j \mid t_i) = \frac{\#(o_j, t_i)}{\#(t_i)} \quad [6]$$

Also, the estimation of π is straightforward; it can be found by calculating the number of sentences in which a tag is related to the first word over the total number of sentences in the dataset. We choose to calculate these quantity as:

$$P(t_i \mid \texttt{<START>}) = \frac{\#(t_i, \texttt{<START>})}{\#\text{ sentences}} \quad [7]$$

Where `<START>` is the special tag used to calculate the probabilities for the tags that do not have a predecessor. Another special tag, `<END>`, is used to calculate the probability of a certain tag ending the sentence.

[2]To whom correspondence should be addressed. E-mail: alberto.cardini(AT)edu.unifi.it

3. Viterbi algorithm for sequence decoding

Rabiner's 1989 second problem concerns how to choose the *best* state sequence corresponding to an observation sequence given the model λ.

Such a problem is called **decoding** and in POS tagging is about choosing the tag sequence $t_1, ..., t_n$ that is most probable given the observation sequence of n words $w_1, ..., w_n$. In other words (Jurafsky and Marting 2025):

$$\hat{t}_{1:n} = \underset{t_1,...,t_n}{\operatorname{argmax}} P(t_1, ..., t_n \mid w_1, ..., w_n) \quad [8]$$

Which, leveraging Bayes' Rule and both the HMM assumptions, becomes:

$$\hat{t}_{1:n} \approx \underset{t_1,...,t_n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i \mid t_i)}^{emission} \underbrace{P(t_i \mid t_j)}_{transition} \quad [9]$$

The sequence is found by performing the Viterbi algorithm with adequate A, B and π parameter on the input sentence (Jurafsky and Marting 2025).

Given a sentence of length n and a vocabulary of m tags, the algorithm creates an $m \times n$ matrix V where each element v_{ij} holds the probability that the HMM is in state j after seeing the first i observations and passing through the most probable state sequence $t_1, ..., t_{i-1}$. The value of each cell v_{ij} is computed by recursively taking the most probable path that could lead us to this cell. Formally:

$$v_{ij} = \max_{t_1,...,t_i-1} P(t_1, ..., t_{i-1}, o_1, ..., o_t, t_i = j \mid \lambda) \quad [10]$$

Viterbi operates recursively, so the formula becomes:

$$v_{ij} = \max_{k=1}^{n}(v_{i-1,k} \cdot a_{kj} \cdot b_{ij}) \quad [11]$$

The complete path is then backtracked and returned.

4. Implementation

Further we discuss the implementation choices made to perform POS-tagging and test the theoretical results. The code can be downloaded and tested at this link.

Dataset. As said before, the dataset used for this report is completely annotated; therefore, techniques like EM (Expectation-Maximization) aren't needed. The exact dataset can be found at this link.

Training. Estimating B with [6] can lead to an undefined probability for observed words that do not occur in the dataset, resulting in breaking the maximum likelihood mechanism of Viterbi. Indeed, B has a column only for the words that are in the dataset; unknown words do not have a probability. We managed this aspect by replacing very rare words in the dataset with the `<UNK>` placeholder. In this way, we can compute $P(\texttt{<UNK>} \mid t_i)$, which is the probability of tag i generating an unknown word. Therefore, the filter function for the normalization of the dataset is:

$$w_i = \begin{cases} w_i & \text{if frequency} > \phi \\ \texttt{<UNK>} & \text{else} \end{cases} \quad [12]$$

The transition matrix A is also calculated by counting, so it is affected by the same problem as B. This time the normalization is not as convenient, because if we map the rare tag couples to `<UNK>` we could end up generalizing couples that are grammatically impossible with couples that are linguistically possible but very rare, ending up with a limitation of the model's capabilities. Instead, we use *additive smoothing*, which helps give a $P \neq 0$ to those tag couples that are not present in the dataset. So [5] becomes:

$$P(t_i \mid t_j) = \frac{\#(t_i, t_j) + l}{\#(t_j) + l \cdot |V|} \quad [13]$$

Where l is the smoothing coefficient and $|V|$ is the size of the vocabulary (needed to keep the matrix stochastic). For $l = 1$, this technique is called *Laplace smoothing* (there are better values for l, and generally better smoothing techniques, Chen and Goodman 1996).

We applied smoothing also to the emission probability, which, paired with the normalization, makes the model even more robust.

As previously mentioned, we added two additional artificial tags to the tag set: `<START>` and `<END>`. These two tags are fundamental to calculate the probability of a certain tag i starting or ending the sentence. Formally, the transition probability is:

$$P(t_k \mid \texttt{<START>}),\ P(\texttt{<END>} \mid t_k) \quad [14]$$

On the other hand:

$$P(\texttt{<START>} \mid t_k),\ P(t_k \mid \texttt{<END>}) \quad [15]$$

does not make sense and can be ignored.

Viterbi. The implementation of the algorithm follows the pseudocode provided by Jurafsky and Marting 2025. As a preliminary step, we normalize the input sentence by replacing unknown words with the `<UNK>` placeholder. Subsequently, the `<START>` and `<END>` tags are excluded from the tag set and the procedure is performed.

All probabilities are computed in log-space to prevent numerical underflow.

Benchmarking. The model is evaluated on both token-level accuracy (single tags) and sentence-level accuracy (entire sequences). Benchmarking is performed by running inference over a test set and comparing the predicted sequences with the ground truth annotations.

5. Results

In this section, we review the outcomes of the experiments made to stress and bend the model's capabilities. Sentences with different levels of syntactic and semantic ambiguities were given to the model. The model run over these test sentences with different smoothing coefficients. The training outputs, along with the overall benchmarks, are displayed in the plots at the end of the report.

Training. The training procedure was performed using a corpus of approximately 13K fully tagged sentences, testing different values for the smoothing parameter l: 0.01, 0.1, 1, and 30.

Figure 1 displays the resulting transition probability matrices A as heatmaps for the different values of l. In these maps, the tags are sorted by log-probability. The visual representation uses a color gradient where darker shades represent lower probabilities, while higher probabilities reach white.

For $l = 1$, in Figure 1a, a large uniformly colored chunk is visible at the bottom of the heatmap. This effect represents the fact that impossible couples like $P(t_i \mid \texttt{<END>})$ (impossible because `<END>` can only be a t_i tag and cannot be a t_{i-1} tag) got their probability uniformly colored along the entire row. In fact, one can see that the row for `<END>` has only the color corresponding to:

$$\log P(t_i \mid \texttt{<END>}) = \log \frac{1}{1 \cdot |V|} = \log \frac{1}{50} \approx -3.912 \qquad [16]$$

Another occurrence of this phenomenon is visible for the predeterminer tag `PDT` (e.g., *all*, *both*), whose row is uniformly colored except for the only reasonable couples: `DT` | `PDT`, which has a bright color because a predeterminer is commonly followed by a determiner (e.g., *all the*, *both the*), and `PRP$` | `PDT`, with a darker color because apparently in the dataset it is less common to find a predeterminer followed by a possessive pronoun (e.g., *all her*, *both his*).

Almost white spots appear for the pairing `<END>` | `.` ; this means that almost every sentence in the dataset finishes with a dot, making it almost certain to be the last "word" of the sentence. Another example of an almost certain pairing is `CD` | `$`, because every time we use the `$` symbol, it is followed by an amount of money, thus a cardinal number (`CD`).

For $l = 30$, in Figure 1d, we observe a situation of total over-smoothing. The uniformly colored chunk at the bottom expanded, erasing valuable information acquired during the training stage. Looking again at the `PDT` tag, we can see that the nuances mentioned before have been deleted. In fact, now the pairing `PRP$` | `PDT` has the same probability as the nonsense pairing `UH` | `PDT` (e.g., *all oops*).

For $l = 0.1$ and $l = 0.01$, in Figures 1b and 1c, we see in the heatmap that the uniform chunks are small to non-existent. With these values of l, the training information is valued, and at the same time, little probability is given to unknown pairings to permit the model to infer on never-seen sentences.

These heatmaps allow us to understand the limits of additive smoothing techniques. We end up having the same probability for common and rare expressions simply because both aren't in the dataset (Chen and Goodman 1996).

Model accuracy. A test set of approximately 1.4K sentences was used to benchmark the model's performance in terms of token-level and sentence-level accuracy (Figure 2). Multiple iterations of the test were conducted for each considered value of l: Figure 2b for $l = 30$, Figure 2a for $l = 1$, Figure 2c for $l = 0.1$, and Figure 2d for $l = 0.01$.

The overall performance trend is illustrated in Figure 3. The results show that accuracy increases inversely with the value of l, reaching a plateau as l approaches smaller values for both token-level ($\approx 94.1\%$) and sentence-level ($\approx 29.7\%$) predictions. Conversely, for very high values of the smoothing coefficient, the model's performance drops significantly; by assigning similar probabilities to known and unknown pairs, the model effectively 'dilutes' the patterns learned during the training process

Example sentences. To better understand how the Viterbi algorithm works, beside the performances on the test set, we run it over some crafted test sentences. For each sentence, is useful to analyse both the emission probability matrix B and the Viterbi path matrix V (with selected tags marked in highlighted). The main model uses $l = 1$; however, for certain cases, we employ $l = 0.01$ to highlight specific behaviors.

The first one is a standard sentence with no syntactic nor semantic ambiguities:

```
John is going to the office .
```

The algorithm returns the correct sequence:

```
NNP VBZ VBG TO DT NN .
```

As observed in the emission matrix (Figure 4a), a straightforward sentence such as this yields high probability values in specific cells, leaving little ambiguity for the model. Viterbi, by combining the values of a_{ij}, $b_i(j)$, and v_{t-1}, successfully infers the correct solution.

Let us consider the exact same sentence without the final period; the resulting tag sequence is:

```
NNP VBZ VBG TO DT .
```

The word *office* is incorrectly tagged as "." (period). We can observe from Figure 5 (used an $l = 0.01$ model for clearer visualization, though the effect persists regardless of smoothing) that the model correctly identifies *office* as a noun (`NN`) based on emission probabilities (Figure 5a), yet it selects the period tag. This counter-intuitive behaviour is caused by two factors. First, in the training dataset, the vast majority of sentences end with a period, leading the model to estimate $P(\texttt{<END>} \mid \texttt{.}) \approx 1$. Second, in its termination step, the Viterbi algorithm selects the last tag by combining the accumulated path probability, the emission probability, and specifically the transition probability to the `<END>` state. Since this last component is overwhelmingly high for the period tag, the algorithm favours the dot over any other candidate, dominating the emission probability.

With the second sentence, we introduce some lexical ambiguities:

```
The complex houses many students .
```

The sentence contains two words that can be misunderstood if the general context is ignored. The phrase *complex houses* can be seen either a `JJ` | `NNS` (plural noun + adjective) or a `NN` | `VBZ` (singular verb + noun). In this specific context, only the second pair is correct. The results are as follows:

```
DT JJ NNS JJ NNS .
```

The model selects the wrong pair. We can see in Figure 1a that `JJ` | `NNS` got a much brighter color then `NN` | `VBZ`, indicating that the first pair is statistically stronger. Figure 6a shows high values for $P(\textit{complex} \mid \texttt{JJ})$ and lower values for $P(\textit{complex} \mid \texttt{NN})$, which indicates a bias towards considering *complex* an adjective rather than a noun. Therefore, due to [1]

and [11] the overall context is ignored in favor of local maximum likelihood. While Viterbi ensures the global coherence of the sequence, it remains strictly constrained by the local probabilities provided by the transition matrix. If the transition matrix (especially when flattened by smoothing) suggests that an incorrect transition is statistically more probable, Viterbi will follow that path, effectively 'forgetting' the long-range grammatical implications introduced by the initial tags.

As a last test we consider the following sentence full of gibberish that will likely not be in the vocabulary learned for the dataset:

```
The gloopy zibber quops the flimflam .
```

Considering a smoothing factor of $l = 0.001$, the resulting sequence is:

```
DT JJ NN VBZ DT NN .
```

Which is surprisingly correct. In the case of entirely unknown words, the emission probabilities are largely uninformative. In Figure 7a, we see that the nonsense words are assigned almost uniform probabilities across all tags. Effectively, what we observe in the heatmap are the values for $P(\texttt{UNK} \mid t_i)$, which are higher for tags that more frequently generate unknown words. Indeed, the sentence is normalized following [12], and then the Viterbi algorithm is executed. Thus, in scenarios where emission probabilities are uniform, the transition probabilities become the determining factor. If we rearrange the words of the sentence as follows:

```
The quops flimflam gloopy the zibber .
```

We obtain the exact same tag sequence returned by the model for the previous sentence. This ultimately demonstrates that the model has effectively mastered the syntactic backbone of English, enabling it to correctly infer the structure even when applied to nonsense words.

Conclusion

From the results of the several test performed on ad hoc sentences and on the outcomes of the evaluations made over the accuracy on the test set, we can see that the two a priori theoretical simplifications ([1] and [2]) play a fundamental role in the inference abilities of the model, making it ignore the overall context of the sentence.

The quality of the dataset also seems to play a big role in the model accuracy. We have seen how biases in the dataset can lead to avoidable errors.

Another crucial factor is the smoothing we apply to the model. Without smoothing, the algorithm wouldn't be able to do inference over quantities that are not present in the ground truth, thus becoming a sort of lookup-table of the dataset. But over-smoothing leads the model to forget what it learned from the dataset, making it less performant. The trend is clearly visible in Figure 3.

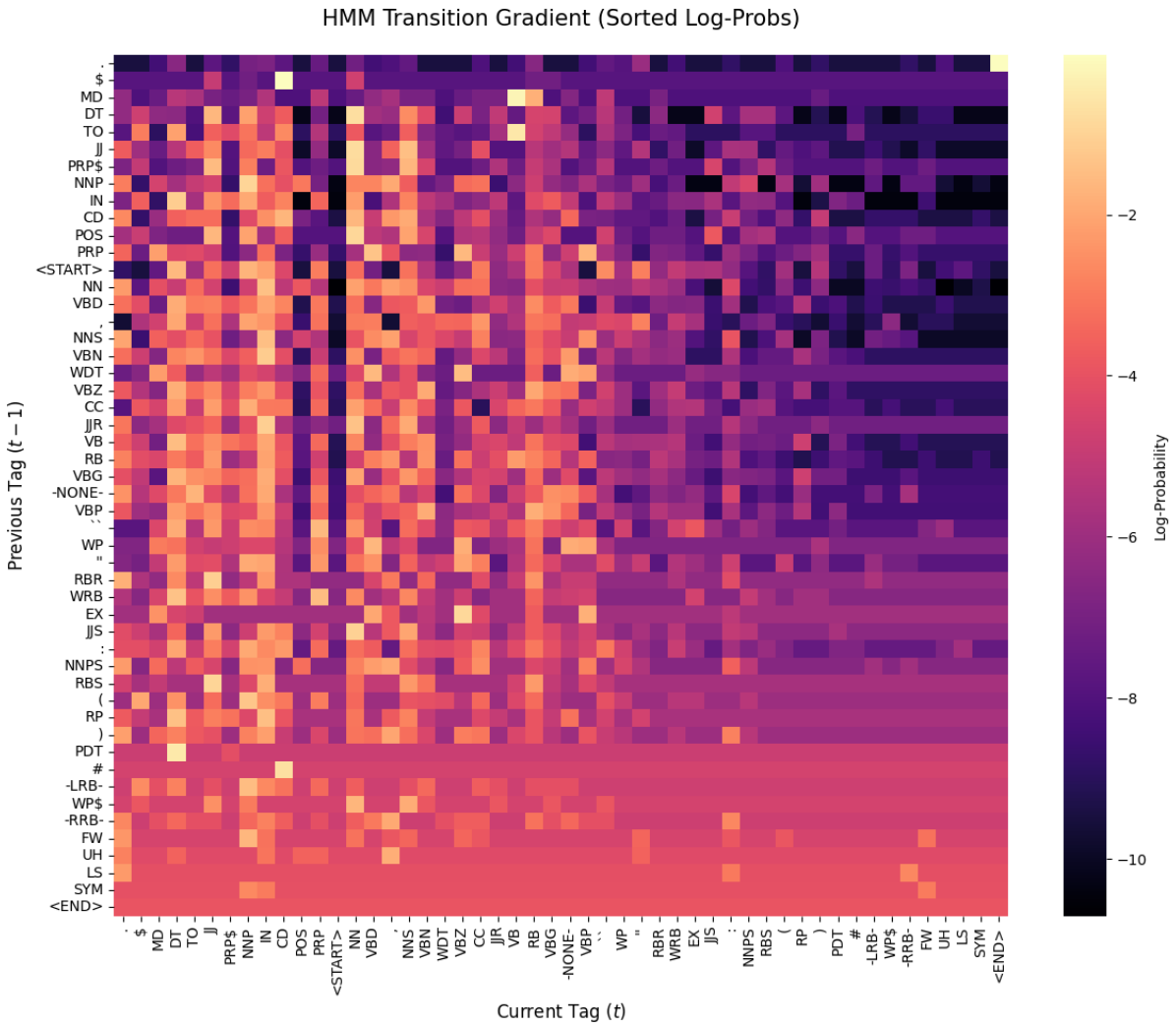
Overall, the model maintains good performance for such a simple architecture.

References

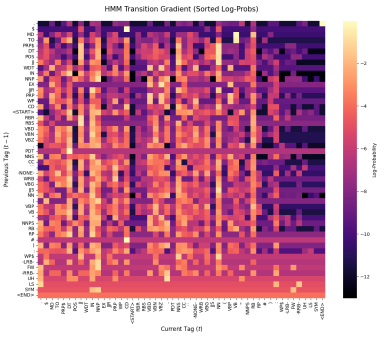
Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. *CoRR*, cmp-lg/9606011.

Jurafsky, D. and Martin, J. H. (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*. 3rd edition. Online manuscript released August 24, 2025.

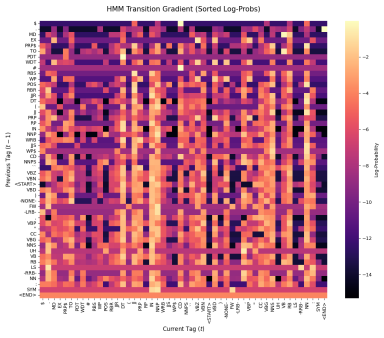
Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.



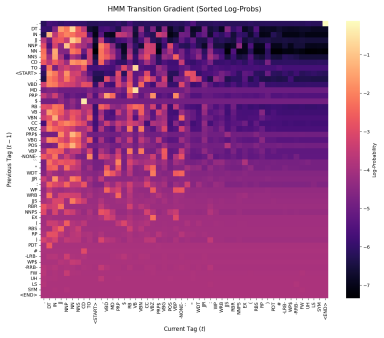
(a) $l = 1$ (laplace smoothing)



(b) $l = 0.1$



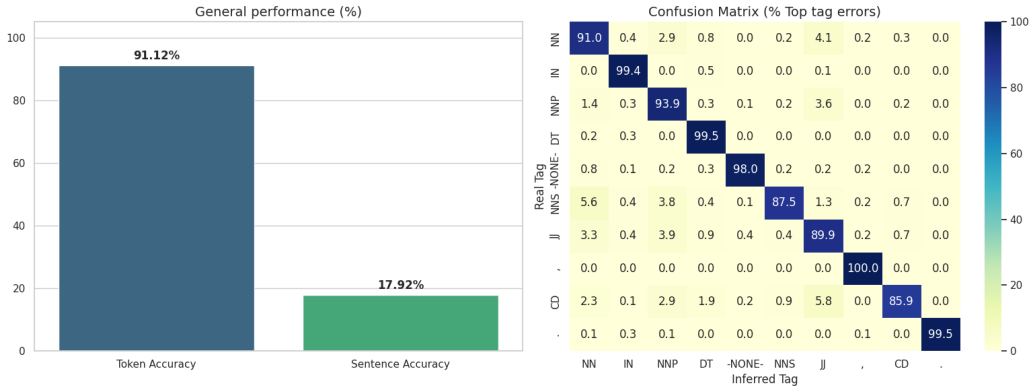
(c) $l = 0.01$



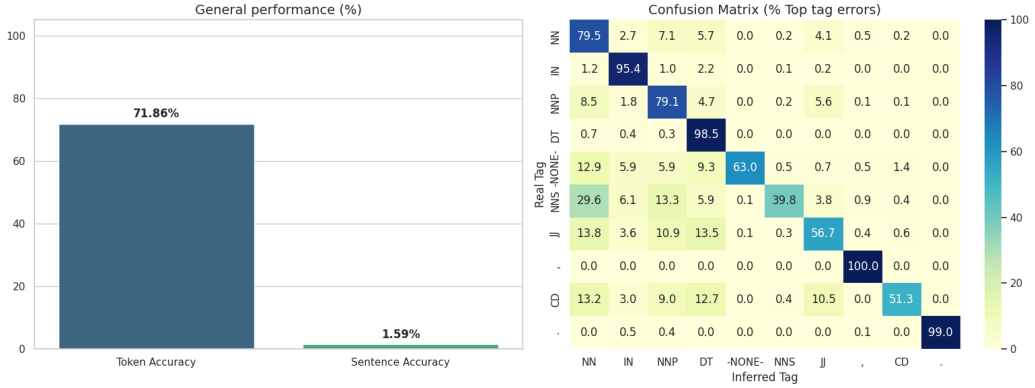
(d) $l = 30$ (over-smoothing)

Fig. 1. Transition probability values for each combination of two tags displayed as an heat-map for different values of the additive smoothing coefficient *l*. Brighter colors indicates higher probability and darker colors means lower probability.

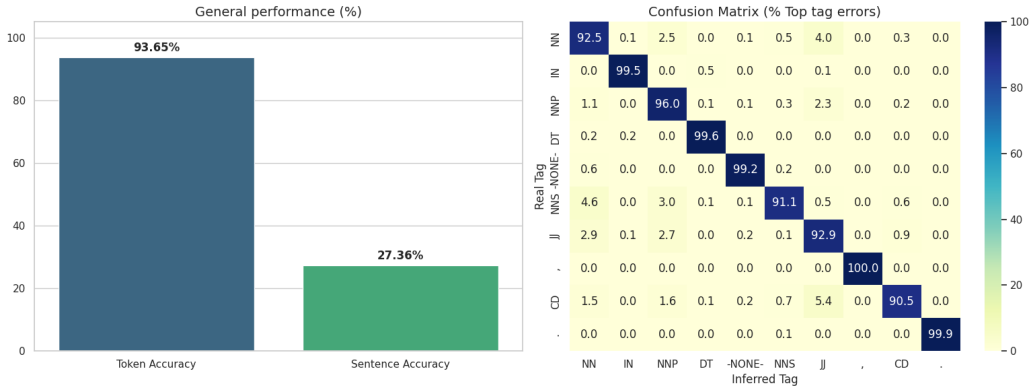
Fig. 2. For different values of l the bar chart compares token-level and sentence-level accuracy, while the side plot identifies the most frequent tag substitution errors.



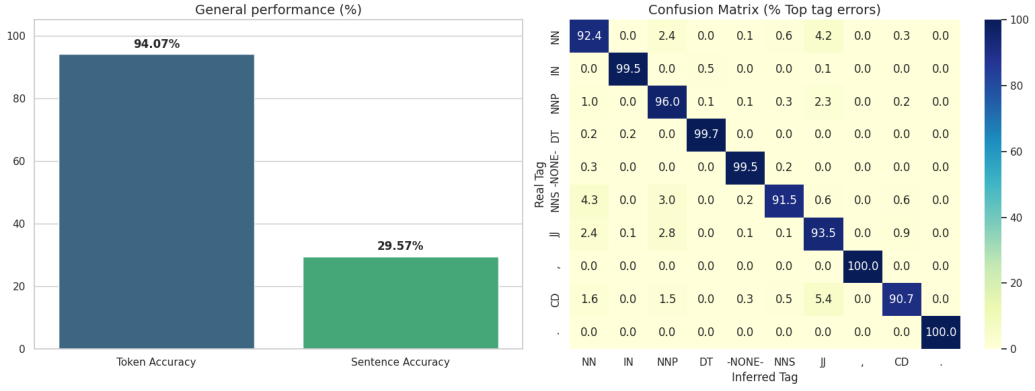
(a) $l = 1$ (laplace smoothing)



(b) $l = 30$

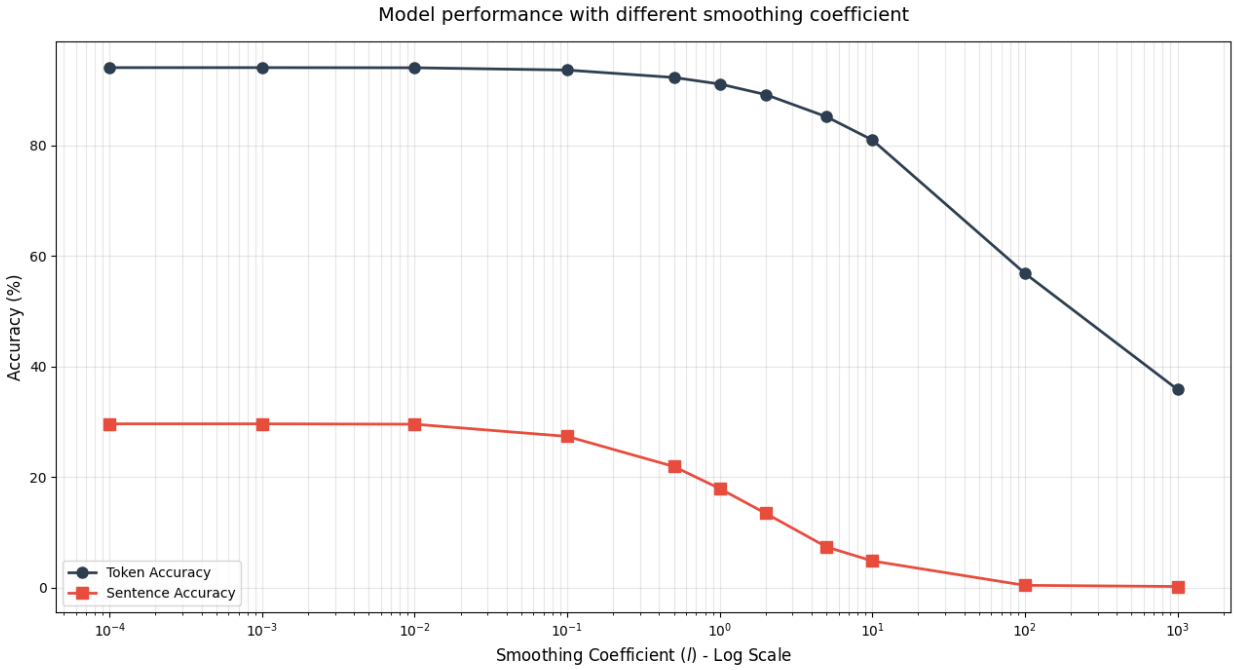


(c) $l = 0.1$



(d) $l = 0.01$

Fig. 3. HMM performance metrics as a function of the Laplace smoothing coefficient l. The x-axis is plotted on a logarithmic scale to illustrate model sensitivity across several orders of magnitude. While Token Accuracy remains relatively robust, Sentence Accuracy exhibits higher sensitivity to variations in l.



(a) Emission heatmap ($l = 1$)

(b) Viterbi heatmap ($l = 1$)

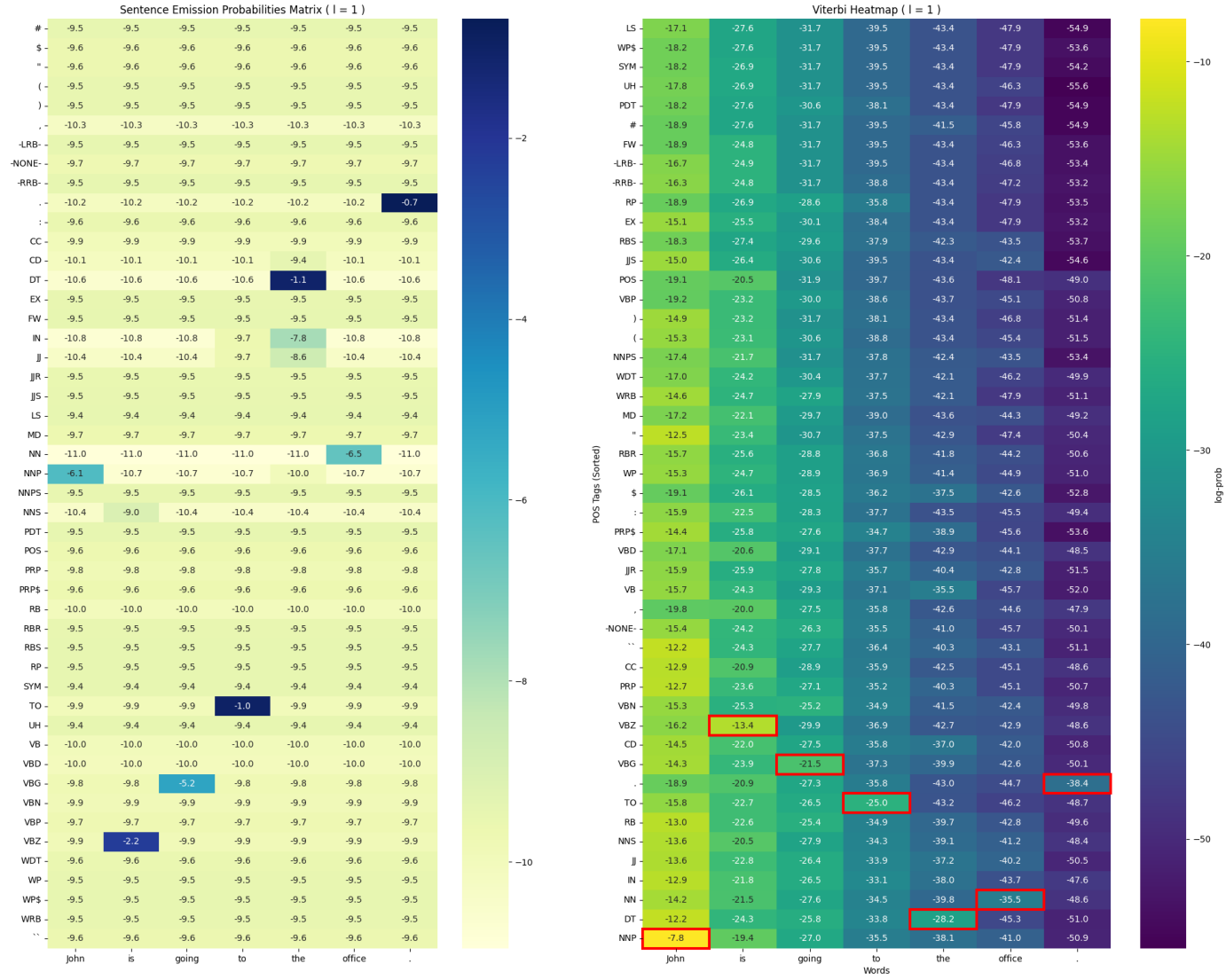


Fig. 4. Emission probability heatmap for the unambiguous sentence "`John is going to the office.`". The strong signals in the emission matrix allow the Viterbi algorithm to easily infer the correct tag sequence. Brighter colors indicates lower probability and darker colors means higher probability.

(a) Emission heatmap ($l = 0.001$)

(b) Viterbi heatmap ($l = 0.001$)

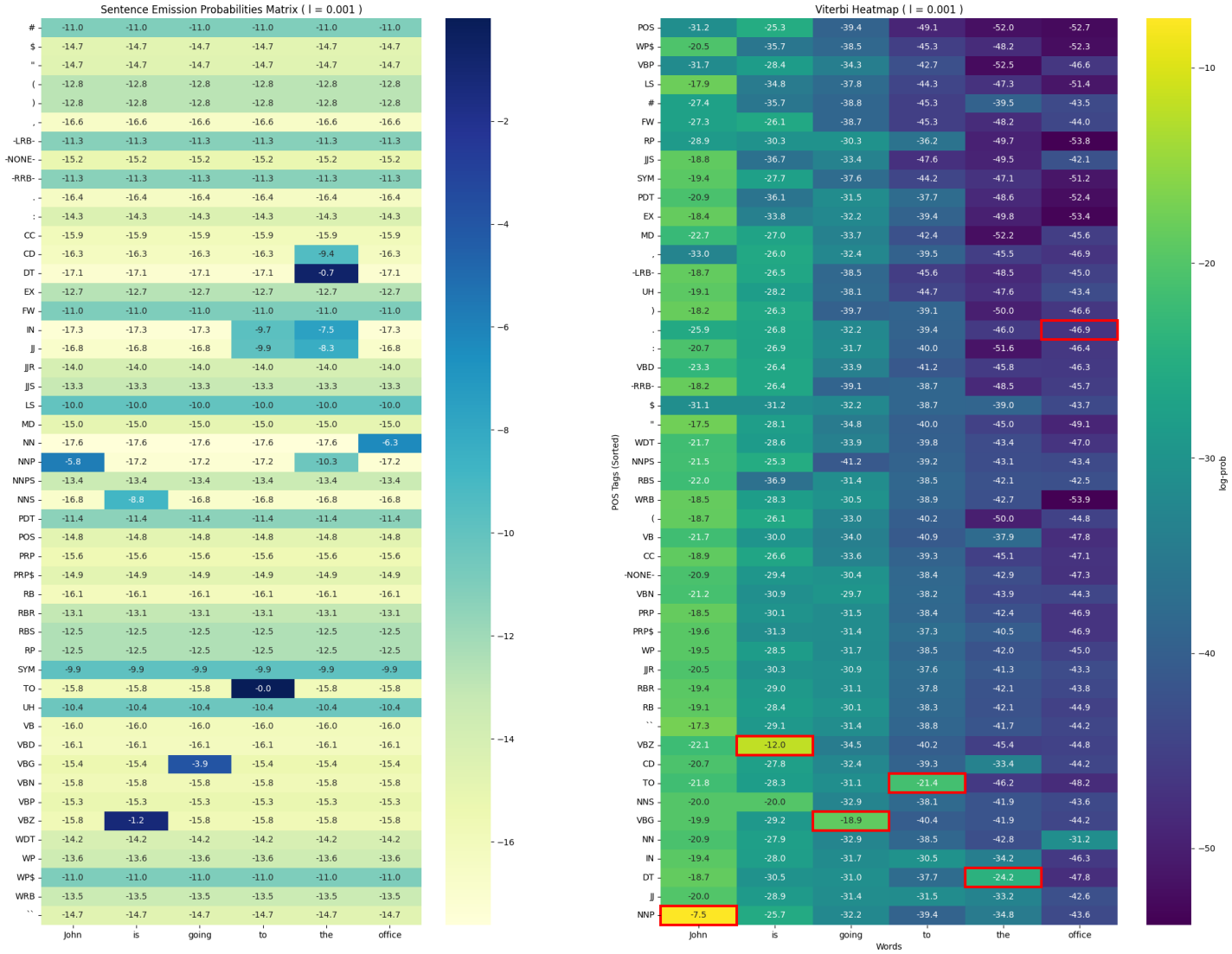


Fig. 5. Analysis of the "missing period" bias. Despite the high emission probability identifying "office" as a Noun (`NN`), the strong transition probability towards the <END> state forces the model to incorrectly tag it as a period (.). Brighter colors indicates lower probability and darker colors means higher probability.

(a) Emission heatmap ($l = 1$)

(b) Viterbi heatmap ($l = 1$)

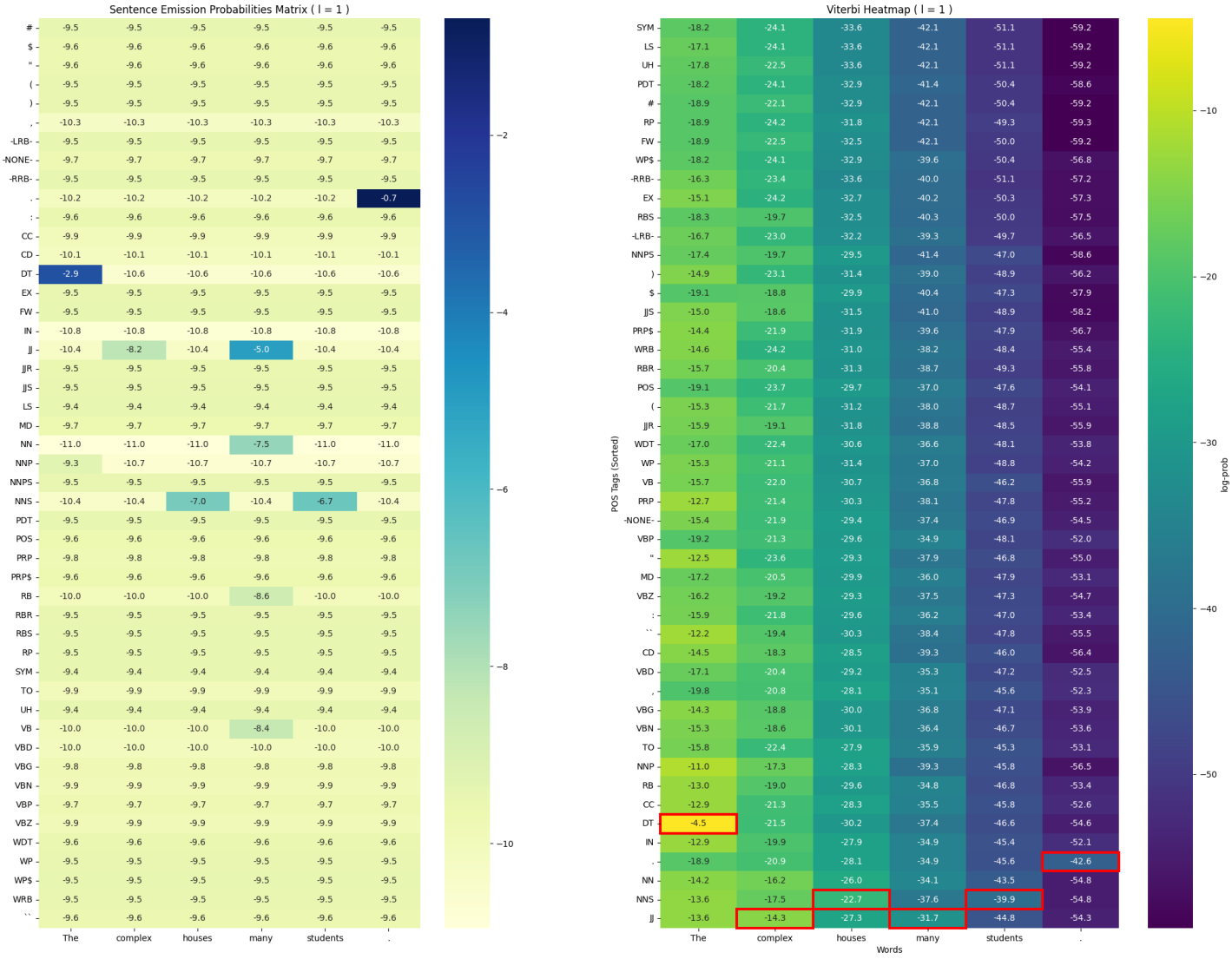


Fig. 6. Visualization of lexical ambiguity. The model incorrectly favors the statistically more frequent pair "Adjective-Noun" (`JJ-NNS`) over the contextually correct "Noun-Verb" (`NN-VBZ`) for the phrase "*complex houses*". Brighter colors indicates lower probability and darker colors means higher probability.

(a) Emission heatmap ($l = 0.001$)

(b) Viterbi heatmap ($l = 0.001$)

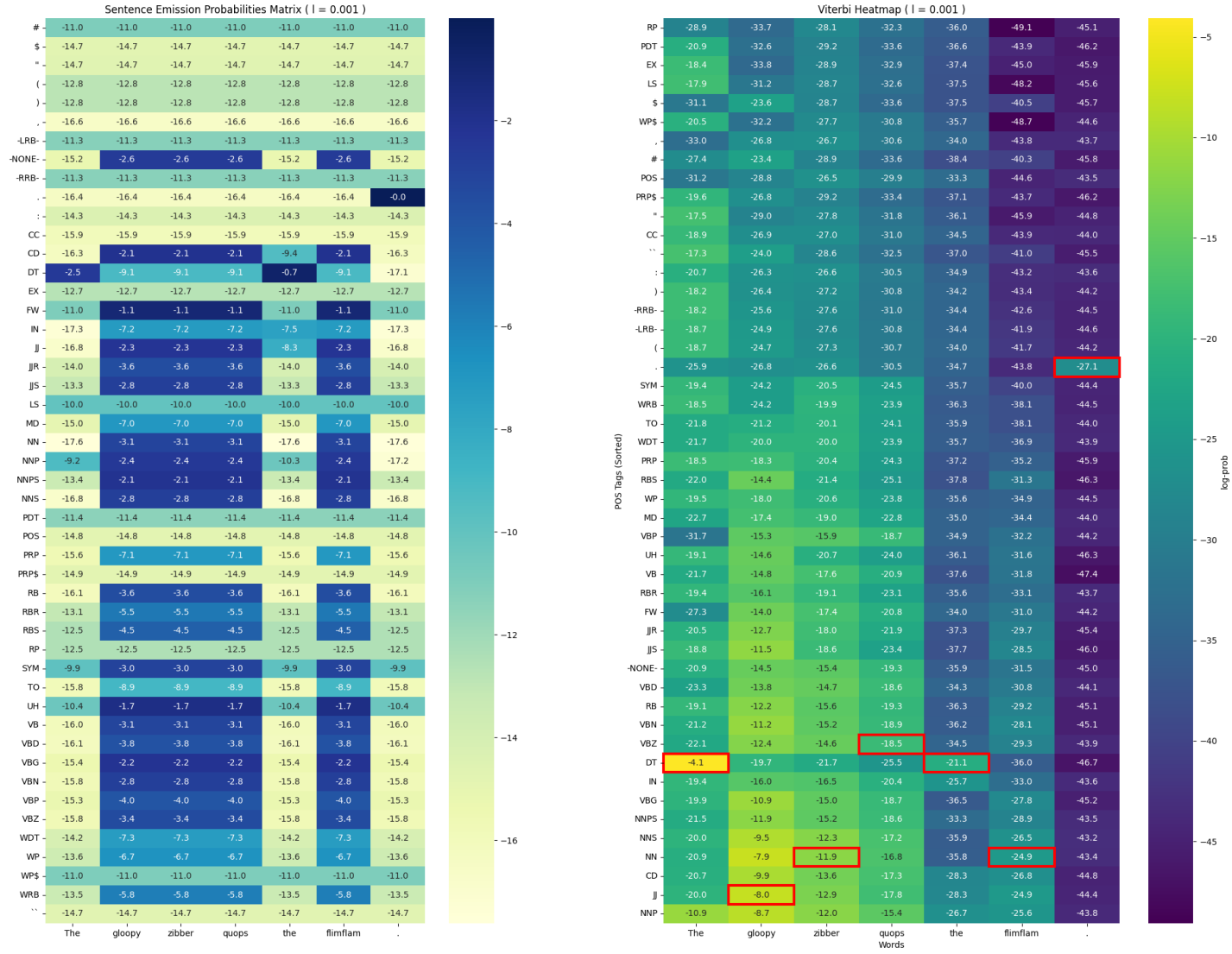


Fig. 7. Inference on Out-Of-Vocabulary (OOV) words. Since emission probabilities are uniform for unknown tokens (<UNK>), the correct syntactic structure is reconstructed solely based on the transition probabilities learned during training. Brighter colors indicates lower probability and darker colors means higher probability.