24-bit Languages

Yiran Wang[1], **Taro Watanabe**[2], **Masao Utiyama**[1], **Yuji Matsumoto**[3]
[1]National Institute of Information and Communications Technology (NICT), Kyoto, Japan
[2]Nara Institute of Science and Technology (NAIST), Nara, Japan
[3]RIKEN Center for Advanced Intelligence Project (AIP), Tokyo, Japan
yiran.wang@nict.go.jp, taro@is.naist.jp,
mutiyama@nict.go.jp, yuji.matsumoto@riken.jp

Abstract

We propose a contrastive hashing method to compress and interpret the contextual representation of pre-trained language models into binary codes. Unlike previous work that generates token-level tags, our method narrows the representation bottleneck to codes with only 24 bits, retaining task-relevant information in a more interpretable and fine-grained format without sacrificing performance (in most cases). We provide experiments and discussions on various structured prediction tasks, such as part-of-speech tagging, named entity recognition, and constituency parsing, to demonstrate the effectiveness and interpretability of our method.

1 Introduction

Pre-trained language models (Devlin et al., 2019; Liu et al., 2019; Lewis et al., 2020; Radford et al., 2019; He et al., 2021) have already become the de-facto infrastructure of modern natural language processing. They have significantly improved performance on various tasks and, at the same time have profoundly and permanently changed the research paradigm. However, lacking interpretability still keeps them a black box to humans, the inability to explain their decision-making mechanisms hinders researchers from further improving them. Fortunately, two recently published papers, which focus on compressing and interpreting continuous representation as discrete tags from pre-trained language models, have shed some light on this issue.

On the one hand, Li and Eisner (2019) propose to compress the contextual representation from pre-trained language models into discrete tags. They utilize the variational information bottleneck (Tishby and Zaslavsky, 2015; Alemi et al., 2017) to nonlinearly interpret high-dimensional continuous vectors into discrete tags, retaining only the information that aids the downstream parsing task. These obtained tags form an alternative tag set and contain necessary syntactic properties. Moreover, the mechanism of the variational information bottleneck, on which their method relies, is to maximize the mutual information between latent discrete tags and targets, while simultaneously minimizing the mutual information between inputs and latent discrete tags. In this way, only the task-relevant information remains in these tags.

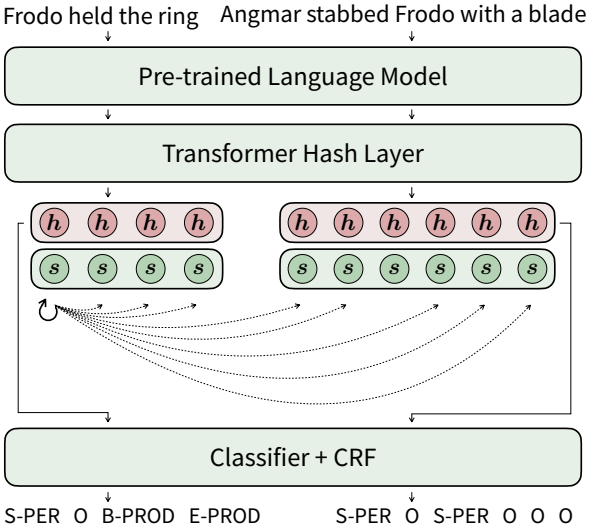

Figure 1: The architecture of the hashing stage model for named entity recognition. The transformer hash layer (§3.1) produces both contextual representation $\boldsymbol{h}$ and ego-attention scores $\boldsymbol{s}$ (§3.1) for the task-specific fine-tuning and contrastive hashing (§2.1), respectively. Solid lines indicate the positive instance, while dotted lines show negatives. Note that the token *Frodo* appears twice in different sentences, thus, to avoid including false positives and false negatives (§2.2), there is no arrow pointing from the first *Frodo* to the second one.

On the other hand, Kitaev et al. (2022) similarly collapse vectors into discrete tags by employing a narrow bottleneck that limits the size of the discrete token vocabulary. Their approach consists of two stages. In the first stage, the contextual vectors of tokens are mapped to discrete tags via the vector quantization method (van den Oord et al., 2017). In the second stage, tags are fed into a down-

Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers), pages 408–419
November 1–4, 2023.

stream model, referred to as the *read-out network* in the original paper, for downstream constituency parsing. Importantly, this read-out network has no access to the continuous vectors but only to these discrete tags, therefore, these tags are forced to encode all the needed syntactic information. Their model achieves comparable performance with only a few bits required for each word.

Different from the two methods above, we provide a novel contrastive hashing method to obtain binary codes from high-dimensional hidden states of pre-trained language models. We push the compression limit by further narrowing the information bottleneck to 24 bits. Following Kitaev et al. (2022), we also introduce a stage to verify whether the information is properly preserved in these binary codes. Additionally, we train an extremely lightweight model using these binary codes as the sole inputs. Experiments show that it successfully reproduces comparable or even slightly better performance than the original full-size model.

Moreover, our method hashes vectors into bit-level binary codes, rather than using token-level tags as in the two previous works. Therefore, the compressed codes are much more interpretable and compact. More specifically, our hashing results not only indicate whether the syntactic properties of two given tokens are different, but also distinguish exactly which bits they differ in.

Our method builds upon contrastive hashing. We introduce a recently proposed Hamming similarity approximation (Hoe et al., 2021) to combine contrastive learning with deep hashing methods. In addition, we introduce an instance selection strategy aimed at mitigating issues related to contextual false positives and false negatives. Moreover, we design a novel transformer-based hash layer, in which each attention head corresponds to a single bit. The entire model is trained to learn to hash by using both the downstream task objective and the contrastive hashing objective simultaneously. These two objectives share a portion of the attention matrix from the hash layer, ensuring that the learned binary codes are likely to properly preserve task-relevant information.

2 Proposed Method

For many tasks, the standard approach of modern language processing is first feeding the input sentence, i.e, $w_1, \ldots, w_n$, into a pre-trained language model to assign each token a continuous vector, i.e., $\boldsymbol{x}_i \in \mathbb{R}^d$, and leveraging them in the downstream task. In this work, we aim to interpret these continuous vectors as discrete binary codes, i.e., $\boldsymbol{c}_i \in \{-1, +1\}^K$, which contains task-relevant information as well. In this way, our method converts continuous vectors to an interpretable format, thereby making the internal mechanism more transparent and comprehensible.

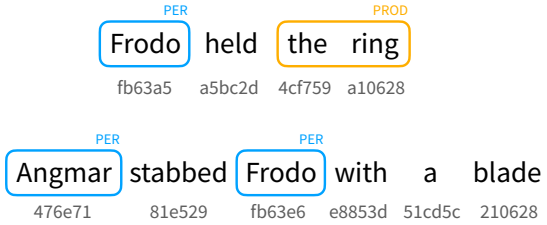

Figure 2: Examples of our method on the named entity recognition task. We assign each word a binary code, i.e., these hexadecimal numbers, and use them as the sole input to recognize entities. PER and PROD are the entity labels for *person* and *product*, respectively.

Our framework consists of two stages. In the first stage, i.e., *hashing stage*, we learn to hash the continuous vectors as discrete tokens. We append a transformer-based hash layer (§3.1) to the end of a pre-trained language model and train the entire model to learn to hash by fine-tuning it on the downstream task. Novelly, we employ the contrastive hashing method (§2.1) and carefully exclude potentially false positive and negative instances with a selection strategy (§2.2). After training, we utilize the hash layer to re-annotate the entire dataset by assigning each token a binary code.

In the second stage, i.e., the *validation stage*, we evaluate whether these binary codes preserve task-relevant information or simply contain meaningless bits. Using these binary codes as the sole inputs, we train a much more lightweight model from scratch. Experiments show that even with such limited capability, our model still achieves comparable or even slightly better performance than the original full-size model. Therefore, we claim that our method properly preserves task-relevant information in these binary codes. The pseudocode can be found in Algorithm 1.

2.1 Contrastive Hashing

Contrastive learning (Chopra et al., 2005; Oord et al., 2018; Chen et al., 2020; Zbontar et al., 2021; Grill et al., 2020) has already been shown to be an effective representation learning method. Its fundamental concept involves employing an encoder

network to map instances into a continuous representation, i.e., $\boldsymbol{x} \in \mathbb{R}^d$. It then pulls together the positive pairs and pushes apart the negative pairs by applying the following objective function[1].

$$\mathcal{L}_{\text{self}} = -\log \frac{\exp s(\boldsymbol{x}, \boldsymbol{x}^+)}{\sum_{\boldsymbol{x}' \in \mathcal{X}} \exp s(\boldsymbol{x}, \boldsymbol{x}')}$$
$$= \log \sum_{\boldsymbol{x}' \in \mathcal{X}} \exp s(\boldsymbol{x}, \boldsymbol{x}') \quad -s(\boldsymbol{x}, \boldsymbol{x}^+)$$

where $\mathcal{X}$ is the instance batch, and $s(\boldsymbol{x}, \boldsymbol{y})$ returns the similarity between the two given instances. Contrastive learning commonly expects instances uniformly distributed on a unit hypersphere. Therefore, the most commonly used similarity function is the cosine function,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|} \tag{1}$$

Deep hashing methods (Cao et al., 2017; Su et al., 2018; Hoe et al., 2021) also aim at mapping instances into informative representation but in discrete space, i.e., $\boldsymbol{c} \in \{-1, +1\}^K$. They first utilize an encoder network to map instances to continuous score vectors, i.e., $\boldsymbol{s} \in \mathbb{R}^K$, and then obtain binary codes by taking signs, i.e., $\boldsymbol{c} = \text{sign}(\boldsymbol{s})$. Besides, deep hashing methods also pull together the positive pairs by encouraging all their bits to become the same and at the same time making negatives pairs have as many as possible different bits. Commonly, this is implemented as Hamming similarity. To be more specific, for two given score vectors, $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^K$, the similarity is defined as,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{K} \text{sign}(\boldsymbol{x}_i) \cdot \text{sign}(\boldsymbol{y}_i) \tag{2}$$

We notice that deep hashing shares the common fundamental concept with contrastive learning, except it represents instances in a K-dimensional Hamming space, i.e., $\{-1, +1\}^K$, instead of a unit hypersphere, i.e., $\mathbb{R}^{d-1}$. Therefore, we propose introducing Hamming similarity to extend the contrastive learning to learn to hash.

However, the Hamming similarity above is not differentiable, introducing it directly is intractable. Recently, Hoe et al. (2021) proposed a novel similarity function that takes the sign of one of its inputs before computing their cosine similarity. They demonstrate that maximizing this similarity preserves semantic information as well. Therefore, we instead introduce this approach to our contrastive learning framework to learn to hash.

$$s(\boldsymbol{x}, \boldsymbol{y}) = \cos(\boldsymbol{x}, \text{sign}(\boldsymbol{y})) \tag{3}$$

[1]We omit the temperature τ for clarity.

Algorithm 1 PyTorch-like style pseudocode.

```
def flatten(tokens):
    """
    removes <pad> and concatenates the remaining tokens.
    e.g., say the <pad> token is 0, and the given tokens are,
    >>> [[1, 2, 3, 4, 5], [6, 7, 0, 0, 0], [8, 9, 10, 0, 0]]
    then this function returns
    >>> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    """

def compute_hash_loss(x, y, tokens):
    # Equation 3
    score = cos(x[:, None], y[None, :].sign(), dim=-1)
    score = score / tau # [tok, tok]

    # excludes potentially false positives and negatives
    mask = tokens[:, None] == tokens[None, :] # [tok, tok]
    score[mask ^ eye] = -float('inf')

    # Equation 4
    return (score.logsumexp(dim=-1) - score.diag()).mean()

def fine_tuning_step(plm, task_model, inputs, targets):
    h1, s1 = plm(inputs) # [bsz, snt, dim], [bsz, snt, K]
    h2, s2 = plm(inputs) # [bsz, snt, dim], [bsz, snt, K]

    task_loss1 = compute_task_loss(task_model(h1), targets)
    task_loss2 = compute_task_loss(task_model(h2), targets)
    task_loss = task_loss1 + task_loss2

    s1 = flatten(s1) # [tok, K]
    s2 = flatten(s2) # [tok, K]
    tokens = flatten(inputs) # [tok]

    hash_loss1 = compute_hash_loss(s1, s2, tokens)
    hash_loss2 = compute_hash_loss(s2, s1, tokens)
    hash_loss = hash_loss1 + hash_loss2

    # Equation 10
    return task_loss + beta * hash_loss

def reannotate(plm, dataset):
    new_dataset = []

    for inputs in dataset:
        _, s = plm(inputs) # [bsz, snt, k]
        codes = s.sign() # [bsz, snt, k]
        new_dataset.extend(codes)

    return new_dataset

def validation_step(lite_task_model, codes, targets):
    logits = lite_task_model(codes)
    task_loss = compute_task_loss(logits, targets)

    return task_loss
```

`plm`: the pre-trained language model with an additional transformer layer; `task_model`: the task-specific model; `lite_task_model`: the lightweight task-specific model with binary code embedding; `bsz`: the batch size; `snt`: the sentence length; `tok`: the total number of tokens in this batch.

2.2 Instance Selection

One of the most appealing properties of contrastive learning is that it successfully converts tasks from *wh-questions* to *yes-no questions*. Conventional classification requires specifying target labels for all instances, but contrastive learning only demands knowing whether two instances are identical or not.

Due to this benefit, effective representation learning becomes possible even in unsupervised settings.

Gao et al. (2021) pass instances into a neural network twice to obtain two semantically identical but slightly augmented representations, i.e., $\boldsymbol{x}$ and $\boldsymbol{x}^+$, relying on the independently sampled dropout masks (Srivastava et al., 2014). They employ the objective $\mathcal{L}_{\text{self}}$ to perform representation learning, treat these two views as positive to each other, and consider all existing instances in the batch as negatives. This simple method surprisingly works well and results in expressive representation.

Furthermore, in supervised settings, Khosla et al. (2020) proposed leveraging label information by introducing an objective function capable of handling cases with multiple positive instances.

$$\begin{aligned}\mathcal{L}_{\text{sup}} =& \frac{-1}{|\mathcal{X}^+|} \sum_{\boldsymbol{x}^+ \in \mathcal{X}^+} \log \frac{\exp s(\boldsymbol{x}, \boldsymbol{x}^+)}{\sum_{\boldsymbol{x}' \in \mathcal{X}} \exp s(\boldsymbol{x}, \boldsymbol{x}')} \\ =& \log \sum_{\boldsymbol{x}' \in \mathcal{X}} \exp s(\boldsymbol{x}, \boldsymbol{x}') \\ & - \frac{1}{|\mathcal{X}^+|} \sum_{\boldsymbol{x}^+ \in \mathcal{X}^+} s(\boldsymbol{x}, \boldsymbol{x}^+)\end{aligned}$$

where the $\mathcal{X}^+$ is the set of positive instances. Obviously, the first term of $\mathcal{L}_{\text{sup}}$ and $\mathcal{L}_{\text{self}}$ are identical. The difference between their the second terms is that $\mathcal{L}_{\text{self}}$ pulls together only one positive while $\mathcal{L}_{\text{sup}}$ pulls together all positive instances.

However, we observe that tokens are likely assigned different information in varying contexts, making it challenging to determine whether two identical tokens truly form a positive pair. For example, in Figure 1, the token *Frodo* appears in both sentences. It serves as the subject in the first sentence and as the object in the second, resulting in dissimilar parses. Therefore, identical tokens may contain distinct task-relevant information and, in such cases, deserve different binary codes.

Since it is difficult to determine whether two identical tokens contain identical task-relevant information in practice, we opt not to include them in either the positive or the negative set. For the numerator part of the objective function, we remove all identical token pairs and retain only the augmented version of themselves as the sole positive instance, thereby reverting to the single positive instance scenario. For the denominator part, we also remove all identical tokens from $\mathcal{X}$ to exclude potential false negatives.

$$\mathcal{L}_{\text{hash}} = -\log \frac{\exp s(\boldsymbol{x}, \boldsymbol{x}^+)}{\sum_{\boldsymbol{x}' \in \{\boldsymbol{x}^+\} \cup \mathcal{X}^-} \exp s(\boldsymbol{x}, \boldsymbol{x}')} \quad (4)$$

Where $\mathcal{X}^-$ only contains tokens that are different from $\boldsymbol{x}$. More specifically, as shown in Figure 1, we consider the second *Frodo* as neither a positive nor a negative instance to the first *Frodo*, so we remove it from both the numerator and the denominator.

The pseudocode of this objective function can be found in the `compute_hash_loss` of Algorithm 1.

3 Architecture

Before introducing our transformer-based hashing layer, we briefly review the mechanism of multi-head attention (Vaswani et al., 2017). The attention layer first projects the input vectors into queries, keys, and values. It then constructs output vectors by aggregating desired information from these key-value pairs.

$$s_{i,j}^h = \frac{(\mathbf{W}_q^h \boldsymbol{x}_i)^\top (\mathbf{W}_k^h \boldsymbol{x}_j)}{\sqrt{d_h}} \quad (5)$$

$$a_{i,j}^h = \text{softmax}_j \left(s_{i,j}^h\right) \quad (6)$$

$$\boldsymbol{z}_i^h = \sum_j a_{i,j}^h (\mathbf{W}_v^h \boldsymbol{x}_j) \quad (7)$$

$$\boldsymbol{o}_i = \mathbf{W}_o \left[\boldsymbol{z}_i^1, \ldots, \boldsymbol{z}_i^H\right] \quad (8)$$

where $\mathbf{W}_q^h, \mathbf{W}_k^h, \mathbf{W}_v^h \in \mathbb{R}^{d_h \times d}$ are the projection weights of query, key, and value of the h-th head, respectively. The $\mathbf{W}_o \in \mathbb{R}^{d \times (H \times d_h)}$ is the output weight, d, d_h, H are the input dimension, head dimension, and the number of heads, respectively. $[\cdot, \ldots, \cdot]$ indicate concatenation and bias terms are omitted for clarity. These hidden states $\boldsymbol{o}_i$ are then fed into a feed-forward network to obtain the output vectors $\boldsymbol{h}_i = \text{FFN}(\boldsymbol{o}_i) \in \mathbb{R}^d$ for downstream tasks. Conventionally, the head size d_h is simply bounded to d and H, but we let the d_h become an independent hyper-parameter, therefore, d does not have to equal to $d_h \times H$ in our implementation.

3.1 Transformer Hash Layer

Intuitively speaking, the mechanism of attention is to selectively aggregate information from tokens. The attention score $s_{i,j} \in \mathbb{R}$ estimates the amount of desired information that token i may obtain from token j. Specifically, $s_{i,i}$ estimates how much desired information is retained in token i itself. Furthermore, by increasing the number of heads to K,

the vector $\boldsymbol{s}_{i,i} \in \mathbb{R}^K$ reflects the desired information scores of token i from K different aspects, and can produce K bits by taking their signs.

Therefore, we add an additional transformer layer with its number of heads increased to K, and use the diagonal entries $\boldsymbol{s}_{i,i}$ of its attention matrix as the hashing scores to learning to hash, and take their signs to generate binary codes as the hashing results after training, i.e., $\boldsymbol{c}_i = \text{sign}(\boldsymbol{s}_{i,i})$. Since $\boldsymbol{s}_{i,i}$ represents a form of attention directed at oneself, to distinguish it from the commonly known term *self-attention*, we use the term *ego-attention* to describe it in the remainder of this paper.

In summary, the full attention matrix $\boldsymbol{s}_{i,j}$ is utilized in a dual manner: it not only serves the conventional purpose in the Transformer architecture for computing the output vector for target prediction, but also lends its diagonal entries $\boldsymbol{s}_{i,i}$ to learn to hash. Given that a portion of the attention matrix is shared between these two objectives, the learned binary codes are inclined to preserve task-relevant information. This hypothesis is demonstrated by our experimental results in the validation stage.

3.2 Hashing Stage Architecture

The architecture of the hashing stage model, as shown in Figure 1, consists of one pre-trained language model, one transformer-based hash layer, and the task-specific layers. We initialize RoBERTa (Liu et al., 2019) with the checkpoint `roberta-base` as the pre-trained language model.

Part-of-speech Tagging We employ an one-layered classifier and a conditional random field (CRF) (Lafferty et al., 2001) to compute the log-likelihood and utilize the Viterbi algorithm (Forney, 1973) for inference.

Named Entity Recognition We transform the sequence of vectors from the sub-token level back to the token level by taking the average of the sub-token vectors of each individual token. We use the same task-specific layers as part-of-speech tagging.

Constituency Parsing Similarly, we generate the token-level representation by averaging the vectors of sub-tokens. In addition, following Zhang et al. (2020), we use a biaffine span classifier along with a tree-structured CRF. We identify the most probable tree from all valid trees using the Cocke-Kasami-Younger (CKY) algorithm (Kasami, 1965). Following Kitaev et al. (2022), we also incorporate GPT-2 (Radford et al., 2019) using the `gpt2-medium` checkpoint for incremental parsing.

3.3 Validation Stage Architecture

As mentioned above, this stage is only to validate if the task-relevant information has been properly preserved in these binary codes, and is not to distill knowledge into a lightweight model. In this stage, we introduce an extremely lightweight model to ensure that the model lacks the capacity to learn the tasks from scratch. As such, any performance gains can only be owed to the information already preserved within the binary codes. The architecture for this validation stage consists of a binary code embedding layer, a conventional one-layered transformer as encoder, and the same task-specific layers used during the hashing stage.

The binary code embedding layer produces code embeddings through constructing instead of looking up. For a given binary code, $\boldsymbol{c} \in \{-1, +1\}^K$, the binary code embedding layer simply flips the direction of each bit embedding $\boldsymbol{b}_i$, and returns the concatenation of these flipped vectors, where $\boldsymbol{b}_i \in \mathbb{R}^{d/K}$ is the embedding of the i-th bit.

$$\boldsymbol{w} = [c_1\boldsymbol{b}_1, \ldots, c_K\boldsymbol{b}_K] \in \mathbb{R}^d \tag{9}$$

Compared with the learned discrete tags of Kitaev et al. (2022), our binary codes literally encode information at the bit level, while their tags remain at the token level. Thus, although Kitaev et al. (2022) emphasize that their model requires only K bits per word, in practice, their model demands an embedding matrix with shape $2^K \times d$, while our real bit-level embedding needs only $K \times \frac{d}{K}$.

3.4 Training and Inference

In the hashing stage, we balance the task-specific loss $\mathcal{L}_{\text{task}}$ and the hashing loss $\mathcal{L}_{\text{hash}}$, as the `fine_tuning_step` function in Algorithm 1. Besides, our training procedure is also simpler than Kitaev et al. (2022), since we don't need to employ the k-mean algorithm (Ackermann et al., 2012) to initialize the centroids in the first two epochs.

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \beta \cdot \mathcal{L}_{\text{hash}} \tag{10}$$

In the validation stage, we re-annotate the entire dataset first and then use the task-specific loss $\mathcal{L}_{\text{task}}$ only to train the lightweight model with only these binary codes as inputs. The procedures for `reannotate` and `validation_step` are described in Algorithm 1, respectively.

Model	POS RoBERTa		NER RoBERTa		Parsing RoBERTa		Parsing GPT2	
	Acc	$\|\theta\|$	F_1	$\|\theta\|$	F_1	$\|\theta\|$	F_1	$\|\theta\|$
Kitaev et al. (2019)	-	-	-	-	95.59	342.8M	93.95	362.5M
Kitaev et al. (2022)	-	-	-	-	95.55	361.4M	94.97	381.1M
Baseline	98.27	134.2M	90.24	134.2M	95.92	136.0M	95.04	422.5M
16 bits	98.37	132.6M	90.21	132.6M	96.00	134.4M	95.02	420.4M
	98.38	0.6M	90.28	0.6M	95.24	2.9M	93.76	5.3M
24 bits	98.38	134.2M	90.27	134.2M	95.92	136.0M	95.14	422.5M
	98.38	0.6M	**90.39**	0.6M	95.51	2.9M	93.82	5.3M
32 bits	98.40	135.7M	90.12	135.7M	95.97	137.6M	95.15	424.6M
	98.41	0.6M	90.31	0.6M	**95.65**	2.9M	**94.02**	5.3M

Table 1: The main results on three datasets. The results of our methods are displayed in two rows, which indicate the performance in hashing and validation stages, respectively. $|\theta|$ columns show the number of parameters, and the bold numbers indicats **the best validation performance** of each setting.

4 Experiments

4.1 Settings

We implement our models with the deep learning framework `PyTorch` (Paszke et al., 2019) and fetch weights of pre-trained language model from `huggingface/tramsformers` (Wolf et al., 2020).

For each batch, we keep collating sentences until the total number of tokens reaches 1024. The reason that we don't use the number of sentences as batch size is to stabilize contrastive learning, since it is performed at token-level, not at sentence-level. We employ AdamW (Kingma and Ba, 2014; Loshchilov and Hutter, 2019) with 50,000 training steps and 6% warm-up steps. In the hashing stage, we evaluate the performance with different number of bits, specifically $K \in \{16, 24, 32\}$.

We run experiments on a single NVIDIA Tesla V100 graphics card. The hashing stage training takes about 2 hours, while the validation stage requires only around 30 minutes. We run the experiments four times with different random seeds. The reported numbers in the following tables are their averages. For comparison, we additionally conduct a baseline experiment for each task without using the contrastive hashing loss, i.e., $\beta = 0$.

Part-of-speech Tagging We conduct experiments on the English Penn Treebank (Marcus et al., 1993) datasets. The task involves assigning a syntactic label to each token in a given sentence. We report the accuracy scores on the test split.

Named Entity Recognition The OntoNotes English dataset (Pradhan et al., 2013) is used for evaluation. We transform span annotations into the BIOES encoding scheme (Ramshaw and Marcus, 1995), and report the F1 scores on the test split.

Constituency Parsing We evaluate on the English Penn Treebank (Marcus et al., 1993). Following Zhang et al. (2020) and Kitaev et al. (2022), we transform the original tree into those of Chomsky normal form and adopt left binarization with `NLTK` (Bird et al., 2009). We report the F1 scores on the WSJ test split.

4.2 Main Results

As presented in Table 1, experiments on the part-of-speech tagging show that 32 bits achieve slightly better results than 16 bits and 24 bits on both stages. Besides, we notice that results in the validation stage are constantly superior to hashing stage results, no matter how many bits are used.

For named entity recognition, we achieve 90.39 in F_1 score with 24 bits, which is even slightly higher than its hashing stage performance, i.e., 90.27. For 16 bits and 32 bits, the validation stage performance also consistently surpasses their hashing stage performance. We hypothesize that this is because hashing the ego-attention scores may implicitly exclude some unconfident attention scores that might lead to wrong predictions. For example, consider a token that barely contains the desired information of a query, it should be ignored by getting a small attention score. However, if the

$s(\boldsymbol{x}, \boldsymbol{y})$	$\mathcal{L}_{\text{contrastive}}$	NER
$\cos(\boldsymbol{x}, \boldsymbol{y})$	$\mathcal{L}_{\text{self}}$	$90.12 \rightarrow 88.74$
	$\mathcal{L}_{\text{sup}}$	$90.07 \rightarrow 86.91$
	$\mathcal{L}_{\text{hash}}$	$90.19 \rightarrow 88.94$
$\cos(\boldsymbol{x}, \text{sign}(\boldsymbol{y}))$	$\mathcal{L}_{\text{self}}$	$90.15 \rightarrow 90.21$
	$\mathcal{L}_{\text{sup}}$	$90.19 \rightarrow 90.04$
	$\mathcal{L}_{\text{hash}}$	$90.27 \rightarrow \mathbf{90.39}$

Table 2: Comparison of different similarity functions and objective functions on the OntoNotes dataset. The numbers on the left and right sides of $\rightarrow$ represent the hashing and validation performance, respectively.

network unconfidently assigns it an attention score that is only slightly less than 0, then its information still occupies a certain proportion in the final output. On the contrary, our method truncates the attention scores to be -1 or $+1$, and eases the issue in some degree.

For constituency parsing, our method outperforms Kitaev et al. (2022) with 32 bits in the bidirectional parsing task, even they introduce much more tags, i.e., 256 in total. Besides, our 16 bits and 24 bits settings also achieve remarkable performance and are only slightly inferior to theirs. In this task, all experiments in the validation stage show worse results than the corresponding hashing stage results. We hypothesize that this is because constituency parsing is a span-level classification task, token-level hashing is unable to capture the span information completely. This may also be the reason that our method works well on part-of-speech and named entity recognition tasks since they are just at the token level.

For all these tasks, with such a lightweight model in validation stages, our codes still reproduce comparable or even slightly better performance than the original full-size model. We claim that these results demonstrate that our learned binary codes have properly preserved task-relevant information.

4.3 Ablation Studies

Table 2 shows that the similarity and objective functions are essential to our method. Using the cosine similarity, the model shows relatively high performance in the hashing stage, however, the naive cosine similarity can not preserve information properly, as its performance dramatically drops in validation stage. Furthermore, the fact that $\mathcal{L}_{\text{hash}}$ consistently outperforms both $\mathcal{L}_{\text{sup}}$ and $\mathcal{L}_{\text{self}}$ demonstrates our hypothesis that false positives and false negatives are harmful.

β	0	0.001	0.005	0.01	0.05
NER	90.24	90.25	90.27	90.10	90.02
	79.60	90.29	**90.39**	90.24	90.23

Table 3: Named Entity Recognition experiments with β. The two rows display hashing and validation performance, respectively.

Additionally, as indicated in Equation 10, the coefficient β serves to balance the two terms. According to Table 3, even though the contrastive hashing loss requires only a minor proportion of the overall loss, demonstrated by the optimal performance of a small $\beta = 0.005$, it is critical for preserving information. Experiments reveal that removing the contrastive hashing loss, i.e., $\beta = 0$, results in a dramatic performance drop.

4.4 Case Studies

We present the hashing and constituency parsing results in Figure 3 to demonstrate the interpretability of our learned binary codes. For comparison with Kitaev et al. (2022), we use the exact same examples as in their paper. Additional parsing results can be found in Appendix D.

We begin by discussing bidirectional parsing. In our transformer-based hash layer, each head corresponds to a single bit, and these heads operate independently of one another. This design allows each bit to capture distinct and orthogonal syntactic and semantic properties. Notably, we observe that the generated binary codes cluster based on the part-of-speech properties. For example, the past tense verbs *brought* and *approved* receive similar codes even when they appear in different sentences, differing by only four bits. Similarly, the common nouns *groceries* and *proposal* share 28 bits, highlighting their shared noun properties.

Moreover, since both *groceries* and *proposal* finalize a similar noun phrase, the article *the* before them is assigned the same code. However, the article *the* before the *council* retains quite different bits. We hypothesize these bits indicate the varied attachment locations. Besides, for the two sentences on the left side, the final attachments *him* and *himself* determine the attachment location of the *for* phrases. We observe that there are only 2 bits differ between them, and hypothesize these two bits reflect the differences in the attachment

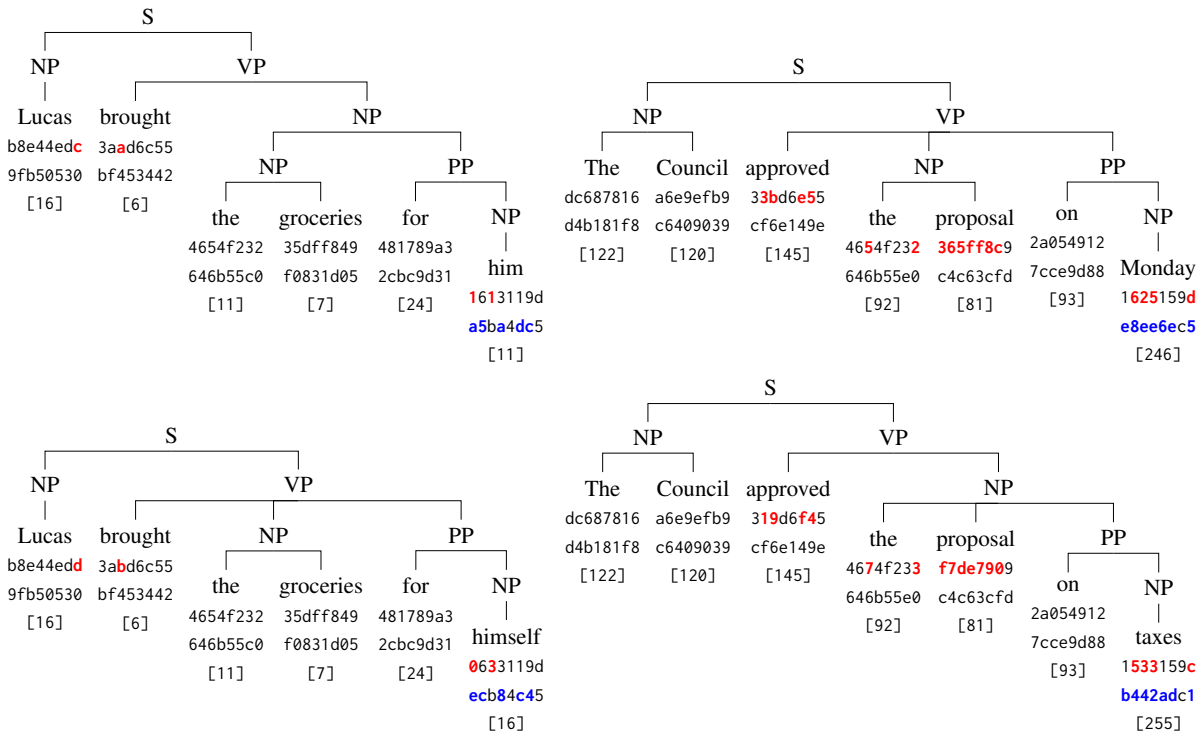

Figure 3: Examples of the hashing and constituency parsing results. There are three numbers below each token, the first two are represented in hexadecimal (32 bits), and indicate the hashing results of the bidirectional (RoBERTa) and unidirectional (GPT2) pre-trained language models, respectively. The third number is taken from Kitaev et al. (2022) for comparison and is represented in decimal. The red and blue parts indicate the exact different bits.

locations. Apart from that, the subject *Lucas* and the predicate verb *brought* also flip one bit, respectively, to indicate the different phrase structures. Similarly, for the right side sentences, *Monday* and *taxes* differ in 5 bits, and the attachment locations of all the phrases that depend on this phrase are influenced, thus, *approved*, *the*, and *proposal* alters their bits as well.

Besides, incremental parsing disallows the information from future tokens, and the future tokens potentially contain syntactic properties that is needed for committing parsing decisions. Therefore, compressed codes should not only retain the already revealed information but also be open to all possible upcoming tokens, as called *speculation free* in Kitaev et al. (2022). Therefore, needed information is mostly distributed in the last tokens, and thus they are likely to obtain varied codes reflecting varied phrases. For example, on the left side, the last noun tokens *him* and *himself* obtain quite different codes, 5 bits different in total, more than the 2 bits in the bidirectional parsing case above. Besides, incremental parsing model also commits similar bits for the article *the* before *groceries* and *proposal*, i.e., only 1 different bit, but assigns a much different code to the article *the* before *council*, which has 15 nonidentical bits. By comparison, even Kitaev et al. (2022) also assign them distinct tags, e.g., 11, 92, and 122, but it is hard for them to tell how different they are and where the differences lie exactly. Thus, we claim that our binary codes are much more informative and interpretable.

4.5 Bit Distribution

To further analyze what specific information is preserved by each bit, we display the bit distribution for named entity recognition in Figure 4.

The sub-figure above illustrates the distribution of bits related to different syntactic information, which serves to indicate the boundary of each entity. It is noteworthy that the bit distributions for the non-entity label `O` are uniform, such that in all these positions the probability of being assigned a 1 is roughly around 50%. In contrast, the distribution of bits for other labels exhibits a clear bias. For instance, on the 9-th bit position, we observed that the label `S` and `B` have 80% and 73% probabilities of being assigned a 1, while the numbers drop to only 47% and 17% for the `E` and `I` labels. We hypothesize the reason is that both `S` and `B` can in-

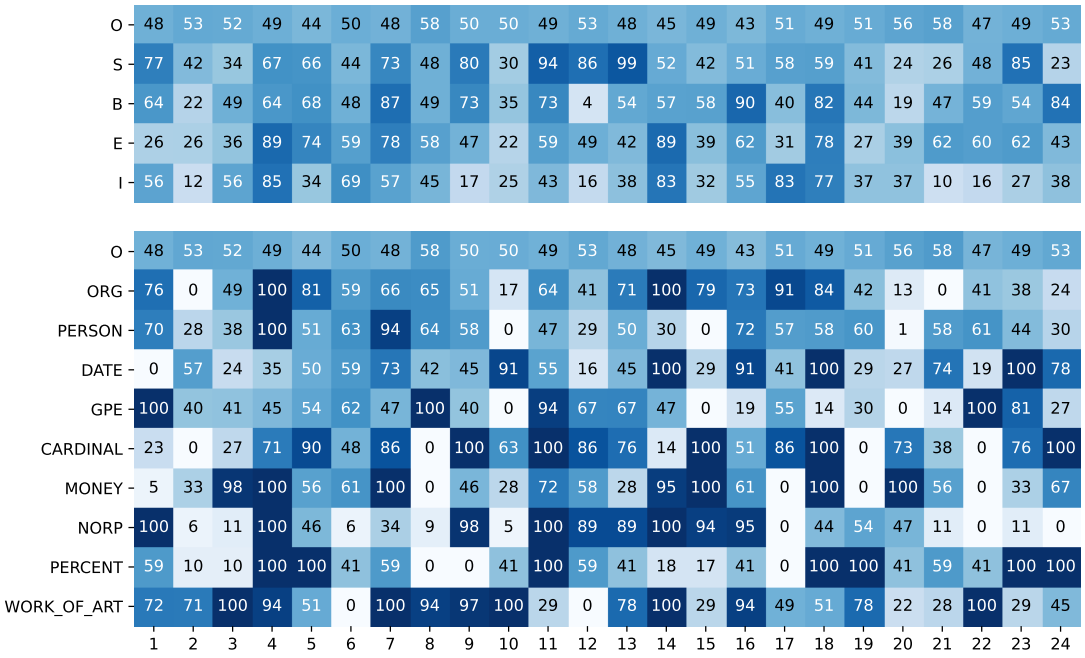

Figure 4: The heatmap of bits distribution. The sub-figure above shows the distribution of bits concerning different syntactic information, while the one below corresponds to semantic information. The number inside cell represents the probability of this label being assigned a 1 at the n-th bit position. For example, the 72 at the bottom left corner indicates that among all of the `WORK_OF_ART` labels, 72% of them are assgiend a 1 at the first bit position.

dicate the beginning of an entity, but such syntactic function is not shared by the other two labels.

The sub-figure below shows the bit distribution related to semantic information and reveals more distinct distributional features. Although the non-entity label `O` continues to display uniform distribution characteristics, labels `MONEY`, `NORP`, and `PERCENT` show that the probabilities at the 4-th and 17-th bits are skewed to 100% and 0%, respectively. Such a clear tendency, low entropy in other words, suggests that task-relevant information is clearly and deterministically preserved within these bits, such that each bit carries a distinct meaning.

5 Conclusions

In this paper, we have proposed a contrastive hashing method to generate interpretable binary codes from pre-trained language models. We designed a transformer-based hash layer, incorporated it into the contrastive hashing framework, and introduced a novel instance selection strategy to exclude false positives and negatives. Experimental results indicate that our lightweight model achieves superior performance and preserve task-relevant information properly with even fewer bits. Further analyses show that the generated binary codes retain syntactic and semantic information in a highly interpretable and fine-grained format. Although we only focus on structured prediction tasks in this paper, as a novel interpretable representation, our method can be easily adapted to other tasks and may inspire future research on designing efficient architectures.

6 Limitations

Although our methods surpass previous work, there is still room for improvement in tasks not at the token level, e.g., constituency parsing. Besides, even the limit has been pushed to 24 bits, which is much better than previous work. However, this is still not the theoretical limit. For example, the total number of labels of named entity recognition is 73, thus, the limit is $\lceil \log_2 73 \rceil = 7$ bits, which is still fewer than ours. We remain solving this limitation and further narrowing the information bottleneck as future work.

References

Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. 2012. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17:2–1.

Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. 2017. Deep variational information

bottleneck. In *International Conference on Learning Representations*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S. Yu. 2017. Hashnet: Deep learning to hash by continuation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR.

Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

G David Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. 2020. Bootstrap your own latent: A new approach to self-supervised learning.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. In *International Conference on Learning Representations*.

Jiun Tian Hoe, Kam Woh Ng, Tianyu Zhang, Chee Seng Chan, Yi-Zhe Song, and Tao Xiang. 2021. One loss for all: Deep hashing with a single cosine similarity based learning objective. In *Advances in Neural Information Processing Systems*.

Tadao Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 18661–18673. Curran Associates, Inc.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Nikita Kitaev, Thomas Lu, and Dan Klein. 2022. Learned incremental representations for parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3086–3095, Dublin, Ireland. Association for Computational Linguistics.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Xiang Lisa Li and Jason Eisner. 2019. Specializing word embeddings (for parsing) by information bottleneck. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2744–2754, Hong Kong, China. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. 2013. Towards robust linguistic analysis using OntoNotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152, Sofia, Bulgaria. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Shupeng Su, Chao Zhang, Kai Han, and Yonghong Tian. 2018. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle.

Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. 2017. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Feng Wang and Huaping Liu. 2021. Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2495–2504.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Perric Cistac, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. pages 38–45. Association for Computational Linguistics.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stephane Deny. 2021. Barlow twins: Self-supervised learning via redundancy reduction. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12310–12320. PMLR.

Yu Zhang, Houquan Zhou, and Zhenghua Li. 2020. Fast and accurate neural crf constituency parsing. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4046–4053. International Joint Conferences on Artificial Intelligence Organization. Main track.

A Dataset Statistics

DATASET	TRAIN	DEV	TEST	LABEL
POS	39,832	1,700	2,416	45
NER	59,924	8,528	8,262	73
PARSING	39,832	1,700	2,416	143

Table 4: Statistics of these three datasets.

B Ablation Study on Temperature τ

τ	0.01	0.02	0.05	0.1	0.2
NER	90.19	90.08	90.02	90.27	90.13
	89.13	89.12	89.81	**90.39**	90.16

Table 5: OntoNotes ablation study results with the temperature τ, which controls the strength of penalties on hard negative samples (Wang and Liu, 2021).

C Hyper-parameter Settings

HYPER-PARAM	POS	NER	PARSING
β	0.05	0.005	0.001
τ	0.1	0.1	0.1
DROPOUT	0.1	0.1	0.5
LEARNING RATE	5e-5	7e-5	5e-5
DROPOUT	0.1	0.2	0.3
LEARNING RATE	5e-4	3e-3	1e-3

Table 6: Hyper-parameters on all tasks. The first block shows the hyper-parameters on hashing stage, while the second one shows the validation stage.

D More Hashing and Parsing Results

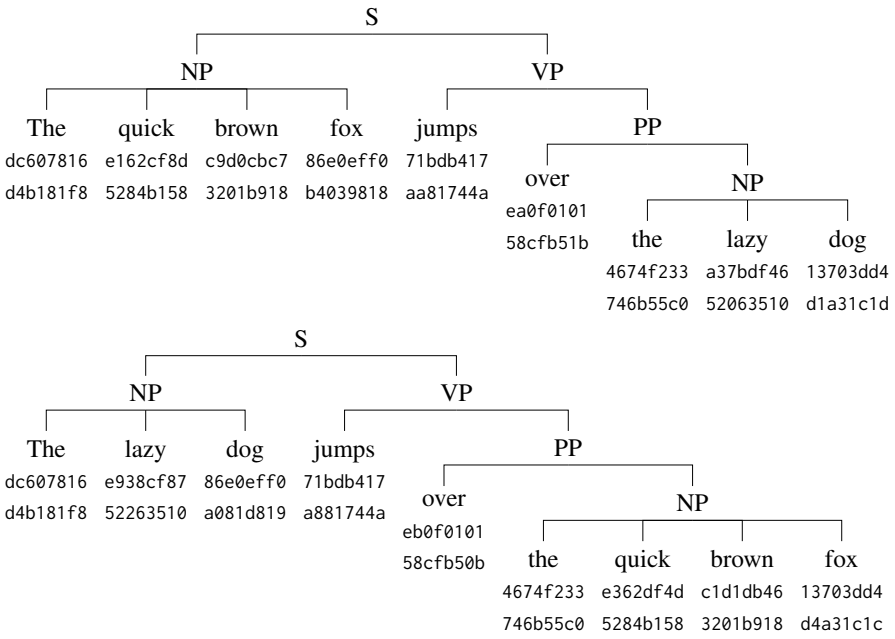

Figure 5: Derivation of the sentence *The quick brown fox jumps over the lazy dog*, and the sentence *The lazy dog jumps over the quick brown fox*.

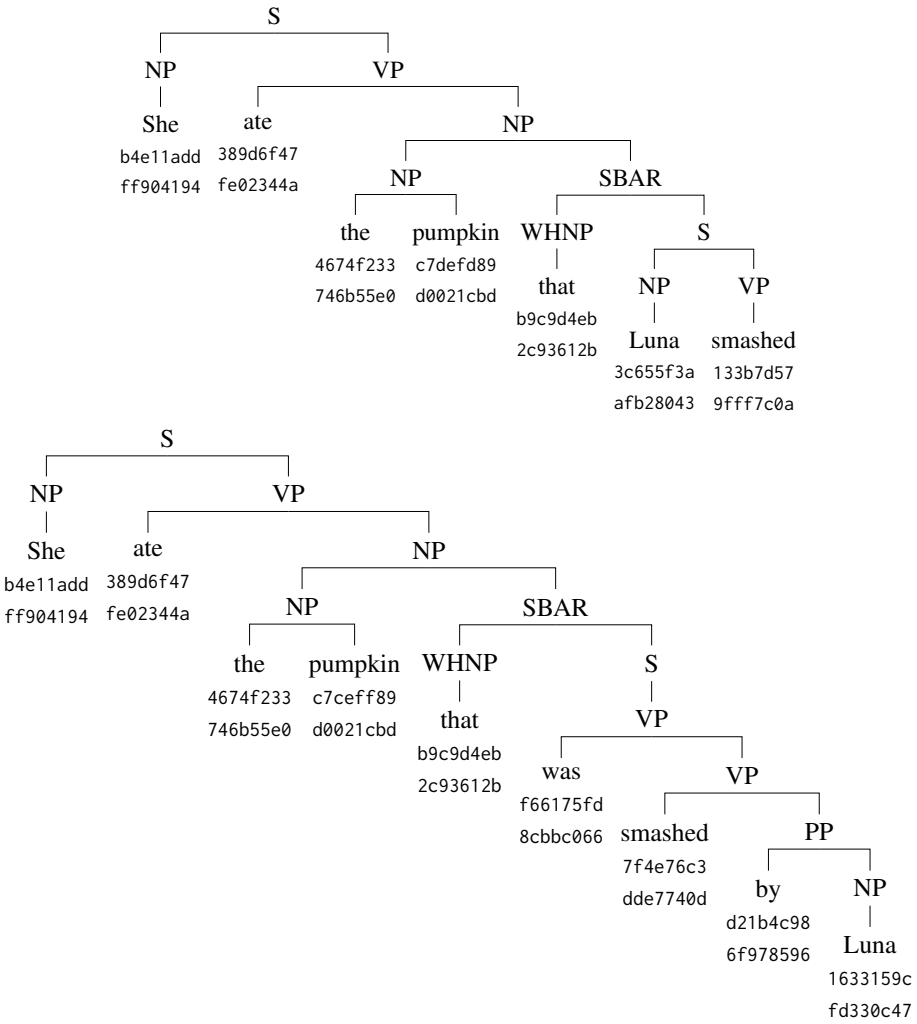

Figure 6: Derivation of the sentence *She ate the pumpkin that Luna smashed*, and the sentence *She ate the pumpkin that was smashed by Luna*.