NUGGET: Neural Agglomerative Embeddings of Text

Guanghui Qin [1] **Benjamin Van Durme** [1]

Abstract

Embedding text sequences is a widespread requirement in modern language understanding. Existing approaches focus largely on constant-size representations. This is problematic, as the amount of information contained in text can vary. We propose a solution called NUGGET , which encodes language into a representation based on a dynamically selected subset of input tokens. These *nuggets* are learned through tasks like autoencoding and machine translation, and intuitively segment language into meaningful units. We demonstrate NUGGET outperforms related approaches in tasks involving semantic comparison. Finally, we illustrate these compact units allow for expanding the contextual window of a language model (LM), suggesting new future LMs that can condition on larger amounts of content.

1. Introduction

You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!

Ray Mooney

Embedding language into dense representations is a central pursuit in modern Natural Language Processing and Machine Learning. Recent work on text encoding has largely focused on fixed-dimensional representations that use either one or a constant number of vectors, e.g., DAN (Iyyer et al., 2015), DPR (Karpukhin et al., 2020), or TSDAE (Wang et al., 2021). At the other extreme, COLBERT (Khattab & Zaharia, 2020) represents and indexes content by storing the final BERT (Devlin et al., 2019) layer encoding of nearly every input token. Unfortunately a fixed dimensional representation risks not scaling to long texts, while a solution like COLBERT comes at significant cost. We propose that a flexible balance can be found, leading to a *"semantically useful level of granularity"* (Rudinger et al., 2017).

[1]Department of Computer Science, University of Johns Hopkins, USA. Correspondence to: Guanghui Qin <gqin2@jhu.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

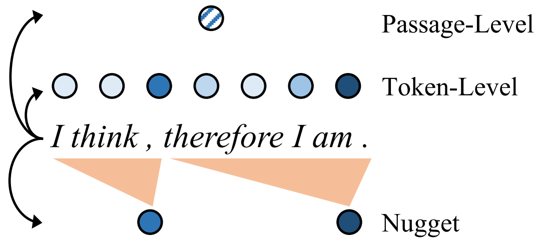

Figure 1. Three approaches to embedding text. Token-level models map each token to a vector, while passage-level models map the whole passage into a single vector. NUGGET generates a dynamic number of vectors, where each nugget encodes a segment of text.

Our solution, NUGGET , is an encoding strategy employing hard-attention to map linguistic input into a fractional number of dynamically selected embeddings called *nuggets*. As the nugget selection process is non-differentiable, we build a residual connection between the selector and decoder to allow gradient propagation, enabling the model to be trained in an end-to-end manner via tasks such as autoencoding or machine translation. This approach allows the number of vectors to grow with input length, trading performance against memory as a configurable compression ratio.

NUGGET leads to an *intrinsically* interesting representation, where the encoder learns to favor clausal text delimiters, such as punctuation and conjunction words. Moreover, without any explicit guidance during training, each resultant nugget encodes a contiguous segment of text preceding these clausal delimiters, such as illustrated in fig. 1.

We demonstrate that *extrinsically* these nuggets outperform prior unsupervised approaches in experiments on document-level paraphrase selection and related passage retrieval.

Finally, through an experiment on language modeling we show that NUGGET can provide context information to other models in an efficient way. Looking ahead, we believe fractional representation strategies like NUGGET will allow for exciting new developments in large language models (LLMs). As nuggets support highly accurate reconstruction, they hold promise as a compressed unit of language that could enable scaling LLMs to condition on significantly longer textual inputs.

2. Background

Token-level Embeddings are commonly used in NLP. To map tokens to individual vectors, Pennington et al. (2014) uses the word co-occurrence matrix as features, while Mikolov et al. (2013) maps words to vectors by training a model to reconstruct the context. Instead of static mappings, encoders such as CoVe (McCann et al., 2017), ELMo (Peters et al., 2018), BERT (Devlin et al., 2019) and BART (Lewis et al., 2020) generate contextualized token embeddings.

Unsupervised methods for passage embedding Early related work modeled passages as topic distributions (Landauer et al., 1998; Blei et al., 2003). With neural networks, researchers map the sentence into one or a fixed number of vectors. Some researchers try to derive a sentence representation from the pretrained encoder without fine-tuning (Wang & Kuo, 2020; Li et al., 2020). Researchers also treat it as an *unsupervised learning* task. Kiros et al. (2015) trains sentence encoding by predicting the surrounding sentences. Bowman et al. (2016); Wang et al. (2021); Mahabadi et al. (2021) explore autoencoding to map sentences into single vectors. With a contrastive objective, Carlsson et al. (2021) learns to have similar representations of the same sentence with two independent encoders, while SimCSE (Gao et al., 2021) uses different dropout masks on the same encoder. Giorgi et al. (2021) is similar but relies on document structure to identify positive sentence pairs. Recently, Li et al. (2022) propose to model texts by denoising a sequence of Gaussian vectors, leading to better controllability.

Supervised methods for passage embedding To construct datasets for general-purpose sentence encoders, it is common to extract sentence pairs from datasets such as natural language inference and question answering (Conneau et al., 2017). SBERT (Reimers & Gurevych, 2019) fine-tunes the BERT model (Devlin et al., 2019) and uses mean pooling over the token embeddings as the sentence encoding. In the domain of dense information retrieval, people map documents into vectors to measure their similarity. Some models simply reuse the token-level encodings: Khattab & Zaharia (2020) uses all token embeddings as the index of the document, while Karpukhin et al. (2020) only reuses the embedding of the `CLS` token. Gao & Callan (2021); Oğuz et al. (2022) show that continual training can produce information-rich `CLS` representations.

The methods mentioned above use a single vector or all tokens as the representation. Tan et al. (2022) increase the number of vectors by introducing pseudo sentences, while Zhang et al. (2022) append `View` pseudo tokens to the BERT (Devlin et al., 2019) self-attention; both have fixed-sized vectors, regardless of the lengths of the input. Rudinger et al. (2017), who helped inspire this work, decomposed sentences into a variable number of *propositional* embeddings, relying on a linguistic processing pipeline.

3. Approach

We use a modified transformer encoder-decoder architecture. Let $\mathbf{w} = \{w_i\}_{i=1}^n$ denote the input sequence, where n is the number of tokens. A transformer encoder is used to map them into contextualized embeddings:

$$\mathbf{X} = \texttt{Encoder}(\mathbf{w}),$$

where $\mathbf{X} \in \mathbb{R}^{n\times d}$ and d is the hidden dimension. Instead of feeding the entire $\mathbf{X}$ into the transformer decoder, we use a "nugget generator", denoted by `Nugget`, to produce a latent variable $\mathbf{Z}$ that are fed as the inputs of the decoder:

$$\begin{aligned} \mathbf{Z} &= \texttt{Nugget}(\mathbf{X}), \\ p(\mathbf{y} \mid \mathbf{Z}) &= \texttt{Decoder}(\mathbf{Z}) \end{aligned} \tag{1}$$

where $\mathbf{Z} \in \mathbb{R}^{k\times d}$, $k \le n$ is the number of "nuggets" generated by `Nugget`, and $\mathbf{y}$ is the target sequence. Note that k is not a constant number and depends on $\mathbf{X}$. `Decoder` is a transformer module with causal masking and is conditioned on $\mathbf{Z}$ via cross-attention.

In the remainder of this section we introduce the form of `Nugget` and the corresponding training strategies.

3.1. Nugget Generator

Instead of producing vectors that do not correspond to actual tokens, such as the `CLS` or averaged pooling over all token embeddings, we leverage the fact that contextual token embeddings carry the semantics of their surrounding texts, and use them as document representations. We use a feedforward network to measure the amount of context information of every token embedding, then select the most informative vectors as the output:

$$\mathbf{s} = \texttt{FFN}(\mathbf{X}), \tag{2}$$

$$\mathbf{X}' = \texttt{TopK}_k(\mathbf{s}, \mathbf{X}), \tag{3}$$

$$\mathbf{Z} = \texttt{Nugget}(\mathbf{X}) = \mathbf{X}'\mathbf{W}^V, \tag{4}$$

where $\mathbf{s} \in \mathbb{R}^n$ are a list of scores, `TopK` is an operator to pick the top k elements in $\mathbf{X}$ sorted by $\mathbf{s}$, and $\mathbf{X}' \in \mathbb{R}^{k\times d}$ are the selected embeddings, $\mathbf{W}^V$ is a trainable parameter, and $\mathbf{Z} \in \mathbb{R}^{k\times d}$ are the latent variables, called nuggets.

Choice of k If we let k be a constant, then `Nugget` falls back to a fixed-dimensional representation. Instead, we let k grow with the length of the text by setting $k = \lceil n \cdot r \rceil$, where the compression ratio $0 < r \le 1$ is a hyperparameter.

Alternative viewpoint Equivalently, one can also view `Nugget` as *hard attention*. Let $\mathbf{q} \in \mathbb{R}^d$ denote a trainable query vector, and we use $\mathbf{X}$ as both keys and values. We can regard eq. (2) as the attention logits:

$$\mathbf{s} = \left(\mathbf{q}\mathbf{W}^Q\right)\left(\mathbf{X}\mathbf{W}^K\right)^\top,$$

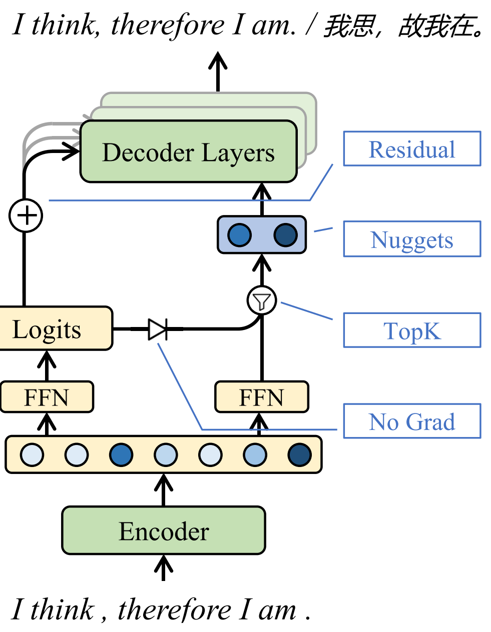
I think, therefore I am. / 我思，故我在。
Decoder Layers
Residual
Nuggets
TopK
Logits
FFN
FFN
No Grad
Encoder
I think , therefore I am .

Figure 2. The architecture of NUGGET . The diode symbol means that the gradient cannot be back-propagated.

where $\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{d\times d}$ are trainable parameters. In the next step, instead of aggregating the values $\mathbf{X}$, we use *hard attention* to take the top-k values in $\mathbf{X}\mathbf{W}^V$ with $\mathbf{s}$ as keys.

3.2. Ensuring Differentiability

Note that the TopK operator in eq. (3) is not differentiable, thus the parameters in eq. (2) do not receive any gradient signals. Therefore, we build a *residual connection* between the encoder and the decoder to propagate the gradients back to the Nugget. Specifically, we append the attention logits $\mathbf{s}$ to the cross attention in the decoder by:

$$\mathbf{a}_\iota = \frac{1}{\sqrt{d}}\left[\left(\mathbf{Z}\mathbf{W}^Q\right)\left(\mathbf{x}^{\text{tgt}}\mathbf{W}^K\right)^\top + \mathbf{s}\right], \tag{5}$$

where $\mathbf{a}_\iota$ is the cross-attention logits for the target token $\mathbf{x}^{\text{tgt}}$ in one attention head at one of the decoder layers, and it will be fed into a SoftMax operator to produce an attention distribution. Note that we have replaced the source tokens with the nuggets $\mathbf{Z}$. In addition to attending to the nugget vectors, the attention score directly takes into account the nugget logits $\mathbf{s}$. As the cross-attention is differentiable, it can be viewed as a residual connection that allows the gradients to be back-propagated to the hard attention parameters. The architecture of NUGGET is shown in fig. 2.

Gradient analysis To interpret the gradients on $\mathbf{s}$, we can rewrite it as:

$$\frac{\partial \ell}{\partial \mathbf{s}} = \sum_\iota \left(\frac{\partial \mathbf{a}_\iota}{\partial \mathbf{s}} \cdot \frac{\partial \ell}{\partial \mathbf{a}_\iota}\right) = \frac{1}{\sqrt{d}} \cdot \sum_\iota \frac{\partial \ell}{\partial \mathbf{a}_\iota}, \tag{6}$$

where ℓ is the loss value, and the summation on the subscript ι is taken over all target tokens, attention heads, and decoder layers. eq. (6) shows that the gradient on the $\mathbf{s}$ is proportional to that on all $\mathbf{a}_\iota$. Consequently, *the nugget logit s_i tends to increase if the model tends to pay more attention to the corresponding nugget vector $\mathbf{z}_i$*. As the bottleneck of the model is to limit the number of nuggets, the model learns to select the token embeddings that contain the maximal amount of contextual information.

Different from previous work with residual connections (He et al., 2017), the introduction of eq. (5) to NUGGET is propagating gradients to the logits $\mathbf{s}$, which otherwise cannot be learned. The absolute values of $\mathbf{s}$ do not greatly affect the cross-attention of the decoder, and we do not observe much performance difference in experiments when ablating $\mathbf{s}$ in eq. (5) during inference.

3.3. Informed Nugget Encoding

The assumption behind NUGGET is that certain tokens function as nuggets to aggregate the surrounding semantics. However, the nugget selection is done after the encoding process, thus cannot affect its attention behavior. To inform the encoder of the selected nuggets, we *prepone* the calculation of $\mathbf{s}$ to the l-th layer of the encoder:

$$\mathbf{s} = \texttt{FFN}(\mathbf{X}^{(l)}), \tag{7}$$

where $\mathbf{X}^{(l)}$ are the hidden states of the encoder in the l-th layer, and we suppose the encoder has $L \geq l$ layers in total. With $\mathbf{s}$ and the compression ratio r, we are able to tell apart the nugget and non-nugget tokens. Akin to the “segment embedding” in Devlin et al. (2019), we add 2 “type embedding” vectors, denoted by $\mathbf{e}^n$ and $\mathbf{e}^o$, to the hidden states of nugget and non-nugget tokens in the l-th layer, which are then fed into the next layer:

$$\mathbf{X}^{(l+1)} = \texttt{SelfAttn}(\mathbf{X}^{(l)} + \mathbf{E}), \tag{8}$$

where $\mathbf{E} \in \mathbb{R}^{n\times d}$ are the type embedding matrix. We call this the **nugget feedback**.

Note that the encoding $\mathbf{X}$ used in eq. (3) are still the embeddings in the last layer. The updated nugget encoding is illustrated in fig. 3.

Stabilized training In practice, we found that the training of nugget selection in eq. (2) can be unstable when the features fed into eq. (8) are being updated. We adopted the common practice for fine-tuning pretrained LMs (Howard & Ruder, 2018) to freeze the bottom l layers of the encoder, which stabilized our training curves. [1]

[1]Freezing bottom layers may also help preserve the multilingual ability of a pretrained multilingual language model; this was not tested in our experiments.

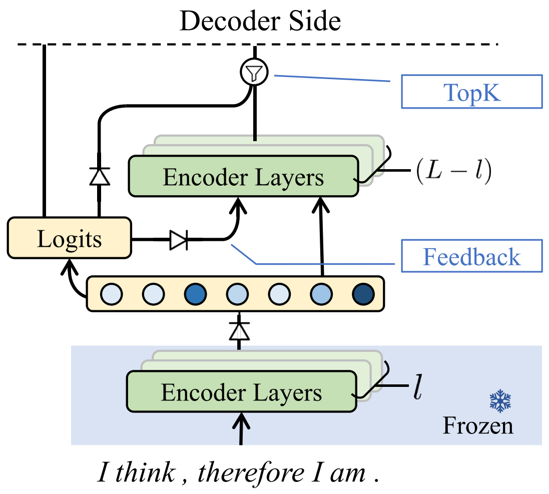

Figure 3. The encoder of NUGGET with *feedback*. The bottom l layers do not receive gradient signals from back-propagation.

3.4. Learning

The model parameters θ are optimized by minimizing the negative log likelihood:

$$\ell = -\sum_{\mathbf{w},\mathbf{y}\in\mathcal{D}} \log p(\mathbf{y} \mid \mathbf{w}; \theta),$$

where the inputs $\mathbf{w}$ and outputs $\mathbf{y}$ are sampled from the dataset $\mathcal{D}$. The dataset $\mathcal{D}$ can be a monolingual corpus, in which case $\mathbf{y}$ should be identical to $\mathbf{w}$ and the NUGGET is trained as an *autoencoder*. Following previous work (Wang et al., 2021), we may randomly delete tokens from $\mathbf{w}$ as noise. The dataset can also be bitexts, then the target document $\mathbf{y}$ is translated from $\mathbf{w}$. In this case, NUGGET is trained as a *machine translation model* (McCann et al., 2017).

4. Experiment Setup

While we could apply the NUGGET concept to a variety of existing models, for experiments here we build on the architecture of BART (Lewis et al., 2020). We start with the checkpoint in Tang et al. (2020), which is a model with 12 layers of encoder and decoder, and is optimized for many-to-many machine translation. It contains 602M parameters, with 256M in the embedding matrix, 152M in the encoder and 203M in the decoder.

For the dataset, we use the English-to-Chinese subset of WMT19 corpus (Barrault et al., 2019), the same corpus used by Tang et al. (2020), as our datasets. WMT19 is comprised of individual sentences, and we concatenate the adjacent sentences together to recover the document structure, similar to the practice of Junczys-Dowmunt (2019). We limit each document to a maximum length of 128 sub-words. The model is trained to translate English documents into Chinese

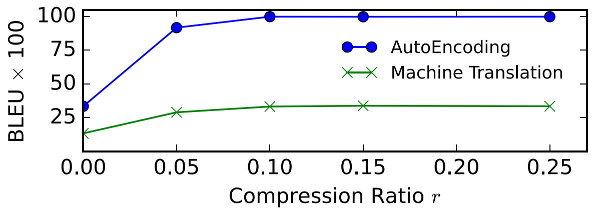

Figure 4. The micro-averaged BLEU value of the texts generated from nuggets with the input document as the reference. Note that $r = 0.0$ indicates that a single vector is used for each document. Results are reported on the dev set of WMT19.

documents. For the autoencoding (AE) objective, we use English documents on both the source and target sides.

We explored different compression ratios r from 0.05 to 0.25. We freeze the bottom 3 layers ($l = 3$) in section 3.3 across our main experiments, and we provide a study of the effect of the number of frozen layers in section 7.1. We put more training details in appendix B.1.

5. Intrinsic evaluation

In this section, we conduct experiments to investigate the impact of compression ratio r. We also discuss the behaviors of the nuggets and their relationship to the textual forms.

5.1. What is a sufficient compression ratio?

The compression ratio r controls the trade-off between space efficiency and the "semantic completeness" of the nuggets. Prior to applying NUGGET to downstream tasks to find a sufficient compression ratio, we propose to use beam search with a beam size of 5 to decode texts from the generated nuggets and measure their difference from the inputs with the BLEU (Papineni et al., 2002) metric.

We evaluate the model on the dev set of the English-to-Chinese subset of WMT19, where sentences are concatenated to document with a maximum length of 128 tokens. The experiment results are shown in fig. 4. With both the AE and MT training objectives, the performance starts to be saturated with a compression ratio of $r = 0.1$. It shows that with 10% of tokens as nuggets, the model has already gained sufficient information about the source documents. In the case of autoencoding, the BLEU value is higher than 0.99 when $r \geq 0.1$, meaning NUGGET reconstructs the inputs nearly verbatim, achieving almost lossless text encoding.

5.2. What is selected as nuggets?

Instead of uniformly selecting tokens, the scorer (eq. (2)) of NUGGET prefers certain tokens. fig. 5 shows the top-6 most frequent tokens selected by NUGGET , and they are mostly delimiter words, like punctuation tokens (commas

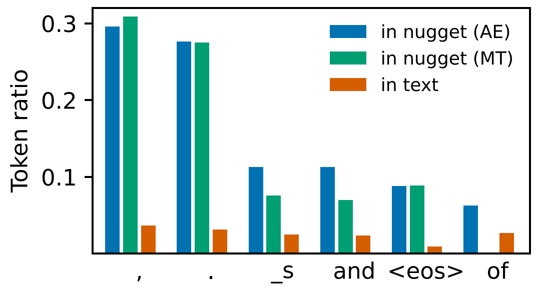

Figure 5. The 6 most frequent tokens selected by NUGGET . We show their ratio in the nuggets with the AE and MT training objectives compared to that in normal texts The statistics are sampled from 128 documents of lengths up to 128. The compression ratio is set as $r = 0.1$ for both models.

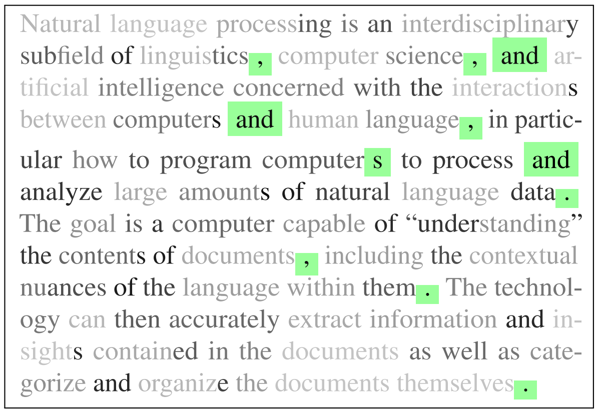
Natural language processing is an interdisciplinary subfield of linguistics , computer science , and artificial intelligence concerned with the interactions between computers and human language , in particular how to program computer s to process and analyze large amounts of natural language data . The goal is a computer capable of "understanding" the contents of documents , including the contextual nuances of the language within them . The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves .

Figure 6. Example texts processed by NUGGET . Tokens in darker colors have higher scores, and those with green backgrounds are selected as nuggets. The compression ratio is set as $r = 0.1$ and AE is set as the training objective.

and periods), conjunctions, and prepositions. Previous work on the study of transformer language models shows that a large amount of self-attention focuses on the delimiter tokens, such as punctuations, and they may be used as no-op Clark et al. (2019). However, our study suggests that they may also serve as *summary tokens*, as predicting the end of a segment requires the model to understand the semantics of the preceding texts.

It is worth noting that in our case study, NUGGET prefers EOS while BOS is never selected, contrary to the practice of Wang et al. (2021). Also, NUGGET is not necessarily selecting the most frequent tokens. For example: the type '*the*', which makes up 5.2% of all tokens in the corpus, accounts for only 0.7% of selected nuggets. An example text is shown in fig. 6, where commas, periods, and the conjunction '*and*' are selected as nuggets.

We note that the preference of NUGGET on text delimiters is not specific to English. In appendix D, we show similar results of fig. 5 in 9 other languages.

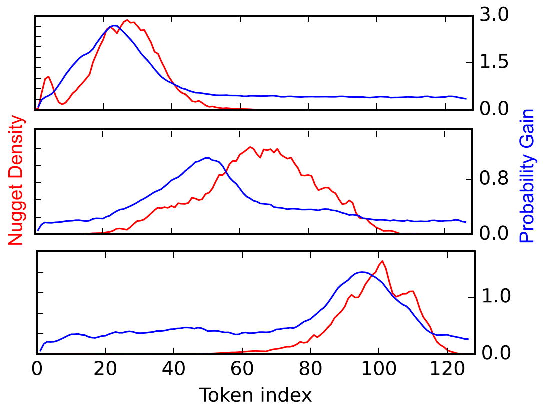

Figure 7. The red curve shows the distribution of token indices in the input documents of the 3rd, 6th, and 9th nuggets, and the blue curve shows the probability gain of every token given the corresponding nugget. The distribution is averaged over 10k documents. Compression ratio r is set as 0.1.

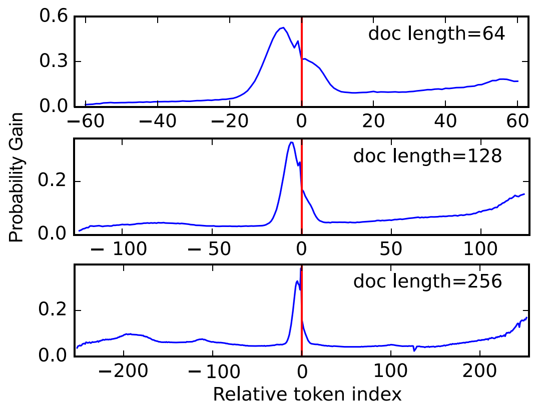

Figure 8. The probability gain conditioned on a single nugget. Graphs are averaged over all nuggets of 10k documents by centering the nugget and showing the relative indices of the tokens. The ratio r is set as 0.1. Refer to appendix C for a complete version.

5.3. What is encoded in each nugget?

The model is optimized to encode information into nuggets, but it is unclear how that information is distributed across them. Thus we propose a method to probe the semantics of individual nuggets.

We run teacher-forcing decoding on a document with a model trained with the autoencoding objective, but expose only 1 nugget during decoding. Suppose the j-th nugget is exposed, then we calculate the "probability gain" by

$$g_i^j = p(y_i \mid \mathbf{y}_{<i}, \mathbf{z}_j) - p(y_i \mid \mathbf{y}_{<i}). \tag{9}$$

where g_i^j measures the increment of probability mass the model has on the i-th token compared to the unconditional decoding. We order the nuggets in each document by the indices of their corresponding tokens, and average $\mathbf{g}^j$ across the j-th nuggets of all documents. The curves of $\mathbf{g}$ are plotted in fig. 7. We can see that the exposure of a nugget can improve the decoding of its preceding texts. Combined with our discovery in section 5.2, we speculate that NUGGET is learning a *divide-and-conquer* strategy, encoding each segment with its ending delimiter tokens.

Note that this experiment made use of documents of length 128 tokens. We then force decoded documents of lengths of 64 and 256 as well, illustrated in fig. 8. These results suggests the properties of nuggets are generalizable to documents with different lengths.

6. What are they good for?

With the nice properties that we observe in section 5, can NUGGET be useful for NLP applications? When *used alone*, NUGGET can be help measure the semantic similarity between texts. NUGGET can efficiently encode long texts with fewer vectors, so we evaluate the use of NUGGET in a **document similarity test**. Also, NUGGET can be used as an *auxiliary module* to provide long-context semantics to other models with minimal information loss. To focus on the language itself and exclude other factors, we propose to integrate NUGGET into a language model and treat it as a **long-range sequence model**.

6.1. Document similarity test

It is common to use semantic textual similarity (STS) to evaluate text representation models (Reimers & Gurevych, 2019; Wang et al., 2021). However, existing datasets for STS, such as Cer et al. (2017), are built on short sentences. To extend this problem to long documents, we built 2 document similarity test datasets based on the corpus of PARABANK (Hu et al., 2019) and WikiText-103 (Merity et al., 2016). [2]

6.1.1. TASKS AND DATASETS

Paraphrase identification on PARABANK PARABANK is a large-scale English paraphrase dataset. It is built on single sentences that are extracted from documents, and we recover the original documents by concatenating adjacent sentences up to 256 tokens. To make this problem difficult, sentences are randomly removed from documents and paraphrases with a probability of 20% independently. For each document, in addition to its paraphrase, we find another 19 negative paraphrases retrieved by the BM25 algorithm (Robertson et al., 2009), and the model is asked to identify the correct paraphrases among 20 candidate paraphrases.

Passage re-ranking on WikiText-103 WikiText-103 is a collection of Wikipedia articles. With the leading section as the query, we randomly sample one section in the same article as the target document and retrieve 19 sections from other articles with the BM25 algorithm as negative examples. The model is asked to rank those 20 passages according to their relevance to the leading section.

Task	Corpus	#queries	$\overline{L_q}$	$\overline{L_d}$
PI	ParaBank	1024	241.3	242.1
RR	WikiText-103	1024	287.0	333.8

Table 1. Data statistics for the task paraphrase identification (PI) and passage re-raking (RR), where $\overline{L_q}$ and $\overline{L_d}$ denote the average number of tokens in query and document.

We put the statistics of the dataset in table 1. Please refer to appendix A for a detailed description of the dataset.

6.1.2. MODEL CONFIGURATIONS AND BASELINES

For those two experiments, we set the compression ratio r as 0.05, 0.1, 0.15, and 0.25, and use the training objectives of both AE and MT. We include the TSDAE model as our baseline (Wang et al., 2021). TSDAE is an auto-encoding model that is trained to reconstruct the input texts with the mean-pooling [3] of all the token embeddings as the bottleneck. For fairness, we re-train the TSDAE model on WMT19 with the checkpoint of mBART under their training configurations, where 60% [4] of input tokens are dropped as noise. As a reference, we also tried replacing the training objective of TSDAE with machine translation.

We do not include the unsupervised models with contrastive learning objectives as baselines, such as Carlsson et al. (2021) and Gao et al. (2021), as they are orthogonal to our contribution: future work will consider contrastive learning for further tuning NUGGET . We refer the readers to Wang et al. (2021) for a comparison between contrastive learning and AE objectives.

We include the approach of ColBERT (Khattab & Zaharia, 2020) as a reference, but replace the encoder with BART (that we call "ColBART"). ColBART uses the last hidden states of mBART encoder as the sentence embeddings.

For single-vector representation models, we adopt the commonly used cosine similarity to measure the similarity between texts. For multi-vector models (NUGGET and ColBART) we adopt the `MaxSim` algorithm proposed by Khattab & Zaharia (2020) but replace the max with a mean

[2] Those 2 datasets are released in `https://github.com/hiaoxui/nugget-data`

[3] To aggregate the token embeddings, we tried using 1) mean-pooling 2) max-pooling 3) the embedding of the `CLS` token. Consistent with the findings in table 7 in Wang et al. (2021), mean-pooling performs best.

[4] We tried 0% (no noise), but training with noise works better.

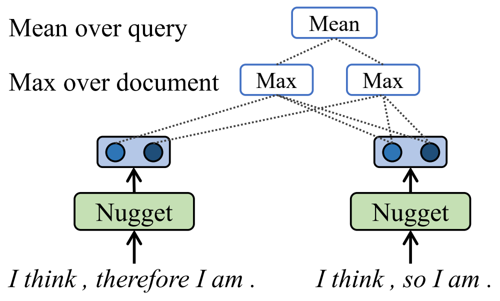

Figure 9. The MaxSim algorithm in Khattab & Zaharia (2020).

		ratio	obj.	multi.	PI	RR
NUGGET		0.25	AE	✓	92.30	44.81
		0.05	MT	✓	92.11	40.54
		0.1	MT	✓	96.69	50.04
		0.15	MT	✓	97.31	52.36
		0.25	MT	✓	**97.38**	**56.51**
TSDAE			AE	×	95.59	50.48
			MT	×	95.04	45.86
ColBART				✓	94.83	52.44

Table 2. Results on paraphrase identification (PI) and passage reranking (RR), reported as MRR×100. "obj." denotes training objective and "multi." denotes multi-vector representation.

operator because we have variable numbers of vectors:

$$m_{q,d} = \frac{1}{I} \sum_i \max_j \texttt{cos}(\mathbf{q}_i, \mathbf{d}_j), \tag{10}$$

where $\mathbf{q}_i$ ($\mathbf{d}_j$) is the i-th (j-th) vector representation of the query q (document d), I is the number of query vectors, and `cos` is the cosine similarity measurement. [5] The algorithm is illustrated in fig. 9.

6.1.3. Experiment results

Results are shown in table 2. Generally speaking, NUGGET trained with the MT objective is more suitable for text similarity measurement without further tuning. A higher ratio leads to better performance, and a ratio of 0.05 (0.15) can make NUGGET achieve comparable performance as ColBART does on the PI (RR) task, while ColBART uses 20x (6.7x) more vectors to encode the text.

In practice, we found that the AE model with a low compression ratio r does not perform well, with a performance gap to TSDAE. We speculate it is because NUGGET with the AE objective does not corrupt the inputs as TSDAE does, while Wang et al. (2021) points out the importance of noisy training for similarity tasks. We leave exploring noising strategies to future work.

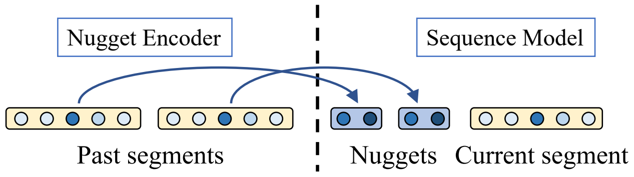

Figure 10. The architecture of NUGGET sequence model. Past segments are compressed with NUGGET and then fed into the sequence model, together with the tokens in the current segment.

6.2. Long-range sequence modeling

An autoregressive sequence model predicts the next token conditioned on past tokens:

$$p(y_i \mid \mathbf{y}_{1:i-1}). \tag{11}$$

When the contexts get longer, the computation can be costly for transformers, which suffer from their quadratic time and space complexity. However, one can compress the history information with NUGGET, and use nuggets as a substitute for the tokens. We rewrite eq. (11) as

$$p(y_i \mid y_{i-s:i-1}, \texttt{Nugget}(\mathbf{y}_{1:i-s-1})), \tag{12}$$

where we use NUGGET to encode all history tokens except for the most recent s tokens. That is, distant information is compressed before being fed into the sequence model.

In experiments, we adopt the decoder part of the mBART as a language model, where the self-attention module is used to read recent tokens and the cross-attention module is used to read nuggets. To let NUGGET encoder work efficiently, we split the distant tokens by the segment length s and encode each segment independently. The architecture of our NUGGET LM is illustrated in fig. 10.

6.2.1. Dataset, training, and metric

We use WikiText-103 (Merity et al., 2016) as the dataset with perplexity (PPL) as the evaluation metric. Models are trained on the training set until convergence. All the results are reported on the test set. We exclude all out-of-vocabulary tokens during the evaluation. [6] Please refer to appendix B.2 for more training details.

6.2.2. Model configurations and baselines

We set the segment length s as 128 for all the experiments. As the context can be very long, we only encode the past

[5] We explored another 2 algorithms: 1) Apply `MaxSim` from both sides to make it symmetric; 2) formulating it as a weighted bipartite matching problem. We found `MaxSim` works better.

[6] Because mBART works on subwords with the BPE tokenizer (Gage, 1994), we take the production of the probabilities over subwords to compute the probability of the complete word. Note that our method theoretically underestimates the model performance.

		h=1	h=2	h=4	h=8
NUGGET	$r = 0.05$	29.88	29.25	28.24	28.14
	$r = 0.1$	29.83	29.21	28.44	**28.10**
TSDAE		30.09	29.55	29.01	28.77
Compressive		-	-	30.52	-
Transformers		($h = 0$) 31.46			

Table 3. Experiment results on language modeling. Performance is evaluated with perplexity (PPL). Above: NUGGET language models with access to different numbers of history segments. Below: Transformer LMs with full attention with context lengths of 128 and 256, and compressive transformers (Rae et al., 2020).

Configuration	PI	RR
NUGGET ($l = 3$)	**96.69**	**50.04**
NUGGET ($l = 0$)	69.82	29.20
NUGGET ($l = 6$)	93.24	48.84
NUGGET ($l = 9$)	84.03	47.36
No feedback	96.29	49.81
Chunking	95.56	42.41
NUGGET ($r = 0.033$)	**89.07**	**49.49**
Sentence boundary	87.91	38.40

Table 4. The experiment results for the ablation study. The performance is measured by MRR$\times$100.

h segments as inputs to the language model in addition to the current segment, where we set h as 1, 2, 4, and 8, with a corresponding context length of 256, 384, 640, and 1152. We start NUGGET LM training from the checkpoints trained with the AE objective and explored the compression ratios of 0.05 and 0.1. As a baseline, we replace the NUGGET with TSDAE and apply the same numbers of history segments.

We use compressive Transformers (Rae et al., 2020) as another baseline, which compresses the past hidden states into fewer vectors. We adopt the "mean pooling" strategy in the paper and compress the most recent 512 tokens into 32 tokens, achieving a similar compression ratio as the model with $r = 0.05$. As a reference for the original transformer LM, we introduce a "full attention model" with a context length of 128. It attends to all tokens without NUGGET , and is equivalent to $h = 0$ in the NUGGET LM experiment.

6.2.3. EXPERIMENT RESULTS

Results are shown in table 3. All NUGGET -assisted models can achieve lower PPL compared to full attention baseline, meaning that the history information provided to LM is effectively utilized. More history segments (larger h) are helpful, though the improvement becomes marginal.

Though NUGGET outperforms the single-vector baseline TSDAE, the difference between $r = 0.05$ and $r = 0.1$ is insignificant. It might be because that $r = 0.05$ has already encoded sufficient information about the content, according to our analysis in section 5.1.

7. Discussion

7.1. Ablation studies

As an ablation study we run NUGGET without the nugget feedback (section 3.3). By default NUGGET uses the features of layer 3 (denoted by $l = 3$) to select nuggets and freeze the parameters below it. Raising l can make the features to the NUGGET selector more contextualized, but also reduce the size of trainable parameters. In the ablation study we explored setting l as 0, 6, and 9, where $l = 0$ corresponds to the embedding matrix.

The selector (section 3.1) is learned with gradient descent with the algorithm in section 3.2. To ablate this module we propose 2 rule-based selectors to replace eq. (2):

- **Chunking selector** We first equally split the document into $\lceil n \cdot r \rceil$ chunks, where n is the number of tokens. For each chunk, we select the last punctuation token (comma or period) as the nugget. If no punctuation exists in the chunk, we select the last token.
- **Sentence boundary selector** As we concatenate the sentences in WMT19 to form documents, we use the ending tokens of sentences as the nuggets. 3.3% of tokens are selected nuggets on average, thus we train a nugget model with $r = 0.033$ as a comparison.

We conduct experiments with those configurations on the tasks of paraphrase identification and passage reranking. By default, we use machine translation as the training objective and use a compression ratio $r = 0.1$ (or $r = 0.033$ for the "sentence boundary" experiments). The results are shown in table 4. One can see that the learned nugget selector is better than rule-based selection, and the optimal features for eq. (2) should be derived from layer 3. The model can also be benefited if NUGGET informs the selection of nuggets via the feedback module.

7.2. Language modeling with long contexts

Previous work has explored ways to enlarge the effective context size for transformer-based encoders (Tay et al., 2022; Qin et al., 2023). As NUGGET provides certified minimal information loss with a high compression ratio, it may enable a complementary approach for long-context modeling.

Large LMs enable in-context learning (ICL) (Brown et al., 2020; Chowdhery et al., 2022), where prior task examples are concatenated as a prefix to a new example which the LM "reasons" over. ICL is constrained by the length of context an LM may condition on: working with compressed nuggets may enable more ICL signal at the same context size.

Wei et al. (2022) demonstrated that ICL performance on complex tasks may be improved by prompting an LM to generate intermediate reasoning steps ahead of a final answer. Transformers suffer from quadratic time complexity, so decoding a *chain of thought* is an expense if one only cares about the final response. Would it be sufficient to decode a chain of nuggets, thereby decreasing runtime?

8. Conclusion and future work

We proposed NUGGET to encode texts with a dynamic numbers of vectors. With auto-encoding or machine translation training, NUGGET naturally segments the input texts following subsentential structures. We demonstrate NUGGET can be useful for semantic similarity and language modeling, achieving better performance than comparable baseline models. To further improve NUGGET for downstream tasks, we will consider additional training approaches such as through contrastive learning, in addition to considering applications of NUGGET to large-scale language modeling.

Acknowledgement

We appreciate the proofreading done by Elias Stengel-Eskin. Thanks to the anonymous reviewers for their feedback.

This research relies on the following open-source software: PyTorch (Paszke et al., 2019), Lightning AI (Falcon & The PyTorch Lightning team, 2019), and Huggingface Transformers (Wolf et al., 2020).

This work was supported by IARPA BETTER (#2019-19051600005). The views and conclusions contained in this work are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, or endorsements of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

Barrault, L., Bojar, O., Costa-jussà, M. R., Federmann, C., Fishel, M., Graham, Y., Haddow, B., Huck, M., Koehn, P., Malmasi, S., Monz, C., Müller, M., Pal, S., Post, M., and Zampieri, M. Findings of the 2019 conference on machine translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pp. 1–61, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5301. URL https://aclanthology.org/W19-5301.

Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. *Journal of Machine Learning Research (JMLR)*, 3(Jan):993–1022, 2003.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. Generating Sentences from a Continuous Space. In *The SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, 2016.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners, 2020.

Carlsson, F., Gogoulou, E., Ylipaa, E., Gyllensten, A. C., and Sahlgren, M. Semantic Re-Tuning with Contrastive Tension. In *International Conference on Learning Representations (ICLR)*, 2021.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. SemEval-2017 Task 1: Semantic Textual Similarity - Multilingual and Cross-lingual Focused Evaluation. In *International Workshop on Semantic Evaluation*, 2017.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. PaLM: Scaling Language Modeling with Pathways, 2022.

Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. What Does BERT Look At? An Analysis of BERT's Attention. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

Falcon, W. and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL `https://github.com/Lightning-AI/lightning`.

Gage, P. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38, 1994.

Gao, L. and Callan, J. Condenser: A Pre-training Architecture for Dense Retrieval. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Gao, T., Yao, X., and Chen, D. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Giorgi, J., Nitski, O., Wang, B., and Bader, G. DeCLUTR: Deep Contrastive Learning for Unsupervised Textual Representations. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Howard, J. and Ruder, S. Universal Language Model Fine-tuning for Text Classification. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

Hu, J. E., Rudinger, R., Post, M., and Van Durme, B. ParaBank: Monolingual Bitext Generation and Sentential Paraphrasing via Lexically-constrained Neural Machine Translation. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2019.

Iyyer, M., Manjunatha, V., Boyd-Graber, J., and Daumé III, H. Deep Unordered Composition Rivals Syntactic Methods for Text Classification. In *International Joint Conference on Natural Language Processing (IJCNLP)*, 2015.

Junczys-Dowmunt, M. Microsoft Translator at WMT 2019: Towards Large-Scale Document-Level Neural Machine Translation. In *Conference on Machine Translation (WMT)*, 2019.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. Dense Passage Retrieval for Open-Domain Question Answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Khattab, O. and Zaharia, M. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *ACM Special Interest Group on Information Retreival (SIGIR)*, 2020.

Kingma, D. P. and Ba, J. L. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. Skip-Thought Vectors. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

Landauer, T. K., Foltz, P. W., and Laham, D. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3): 259–284, 1998.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

Li, B., Zhou, H., He, J., Wang, M., Yang, Y., and Li, L. On the Sentence Embeddings from Pre-trained Language Models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Li, X. L., Thickstun, J., Gulrajani, I., Liang, P., and Hashimoto, T. B. Diffusion-LM Improves Controllable Text Generation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

Mahabadi, R. K., Belinkov, Y., and Henderson, J. Variational Information Bottleneck for Effective Low-Resource Fine-Tuning. In *International Conference on Learning Representations (ICLR)*, 2021.

McCann, B., Bradbury, J., Xiong, C., and Socher, R. Learned in Translation: Contextualized Word Vectors. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer Sentinel Mixture Models, 2016.

Mikolov, T., Corrado, G., Chen, K., and Dean, J. Efficient Estimation of Word Representations in Vector Space, 2013.

Oğuz, B., Lakhotia, K., Gupta, A., Lewis, P., Karpukhin, V., Piktus, A., Chen, X., Riedel, S., Yih, W.-t., Gupta, S., and Mehdad, Y. Domain-matched Pre-training Tasks for Dense Retrieval. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2022.

Papineni, K., Roukos, S., Ward, T., and Zhu, W. BLEU: A method for automatic evaluation of machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Pennington, J., Socher, R., and Manning, C. D. GloVe: Global Vector for Word Representation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. S. Deep contextualized word representations. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

Qin, G., Feng, Y., and Van Durme, B. The NLP Task Effectiveness of Long-Range Transformers. In *Annual Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2023.

Rae, J. W., Potapenko, A., Jayakumar, S. M., and Lillicrap, T. P. Compressive Transformers for Long-Range Sequence Modelling. In *International Conference on Learning Representations (ICLR)*, 2020.

Reimers, N. and Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

Robertson, S., Zaragoza, H., et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

Rudinger, R., Duh, K., and Durme, B. V. Skip-Prop: Representing Sentences with One Vector Per Proposition. In *International Conference on Computational Semantics (IWCS)*, 2017.

Tan, H., Shao, W., Wu, H., Yang, K., and Song, L. A Sentence is Worth 128 Pseudo Tokens: A Semantic-Aware Contrastive Learning Framework for Sentence Embeddings. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

Tang, Y., Tran, C., Li, X., Chen, P.-J., Goyal, N., Chaudhary, V., Gu, J., and Fan, A. Multilingual Translation with Extensible Multilingual Pretraining and Finetuning, 2020.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient Transformers: A Survey. *ACM Computing Surveys*, 55 (6):1–28, 2022.

Wang, B. and Kuo, C.-C. J. SBERT-WK: A Sentence Embedding Method by Dissecting BERT-based Word Models. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2146–2157, 2020.

Wang, K., Reimers, N., and Gurevych, I. TSDAE: Using Transformer-based Sequential Denoising Auto-Encoder for Unsupervised Sentence Embedding Learning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. Transformers: State-of-the-Art Natural Language Processing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Zhang, S., Liang, Y., Gong, M., Jiang, D., and Duan, N. Multi-View Document Representation Learning for Open-Domain Dense Retrieval. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

A. Data construction for document similarity test

We build two datasets for the document-level semantic similarity test. Those 2 datasets can be downloaded in `https://github.com/hiaoxui/nugget-data`. We discuss the details of the dataset construction in this section.

A.1. Paraphrase identification

The document-level paraphrase identification dataset is derived from PARABANK (Hu et al., 2019). PARABANK is a large-scale English paraphrase dataset constructed with a Czech-English neural machine translation system. We use the v1.0 of its release downloaded from `https://nlp.jhu.edu/parabank/`.

PARABANK is sentence-level, but it does not shuffle the sentence orders. To recover the document structure, we concatenate the adjacent sentences to make "fake documents". The concatenation strategy is applied to both the documents and their paraphrases by iterating their sentences in parallel until one of them reaches the 256-token limit. The construction process produces a list of "`(document, paraphrase)`" pairs.

To make the problem difficult, we delete 20% of sentences randomly and independently on both sides. In practice, a sentence will be included in the documents with a probability of 80%, and sentences are drawn independently on the document and paraphrase sides. A robust model should be able to identify the paraphrased sentences even if they are not positionally aligned with their original sentences.

To collect negative examples, we run a BM25 algorithm (Robertson et al., 2009) with the document as the query and paraphrases as candidates. Since the corpus PARABANK is too large to be efficiently indexed, and the most challenging negative examples always come from the same document, we try to run a sliding window around the query document with a window size of 1024 documents. BM25 retrieves 19 negative examples from the candidates, and the model is asked to identify the correct paraphrase.

A.2. Passage reranking

This task asks the model to identify a document with a similar topic to the query document. We start from the WikiText-103 data (Merity et al., 2016), which is a collection of Wikipedia articles. We split the dataset into articles, and use the texts in sections as passages. As the validation and test splits of WikiText are too small to generate challenging negative examples, we work on the training split. Note that WikiText is released with a raw version and a tokenized version, and we use the raw version without masking out any `UNK` tokens.

The first section of each article is usually a general introduction about the article, thus we use it as the query document. We randomly select another section from the same article as the answer passage, and uses the BM25 algorithm to retrieve 19 negative examples from all but the first sections of other articles.

The statistics of the above two datasets are shown in table 1.

B. Training details

B.1. Machine translation and auto-encoding training

We used the same codebase and training configurations for both the auto-encoding (AE) and machine translation (MT) objectives. Both models are initialized from the checkpoint of mBART (Tang et al., 2020), which is a many-to-many machine translation model. We used the Adam (Kingma & Ba, 2015) optimizer with a learning rate of 5×10^{-5}. Each model is trained until convergence on the dev set.

We build a document-level MT dataset from the English-to-Chinese subset of WMT19 (Barrault et al., 2019). The dataset is constructed so that adjacent sentences are concatenated to make document (Junczys-Dowmunt, 2019) with up to 128 tokens. The document might not be full and always contain complete sentences, as we do not break the sentences. The MT model is trained to translate English into Chinese. For the AE objective, we use the same dataset but replace the target Chinese documents with the inputs.

Every model is trained on 4 NVIDIA RTX 6000 GPUs with 24GB $\times$ 4 GPU memory. With a batch size of 16 on each card, the MT model can converge in approximately 48 hours. The AE model usually converges in 36 hours.

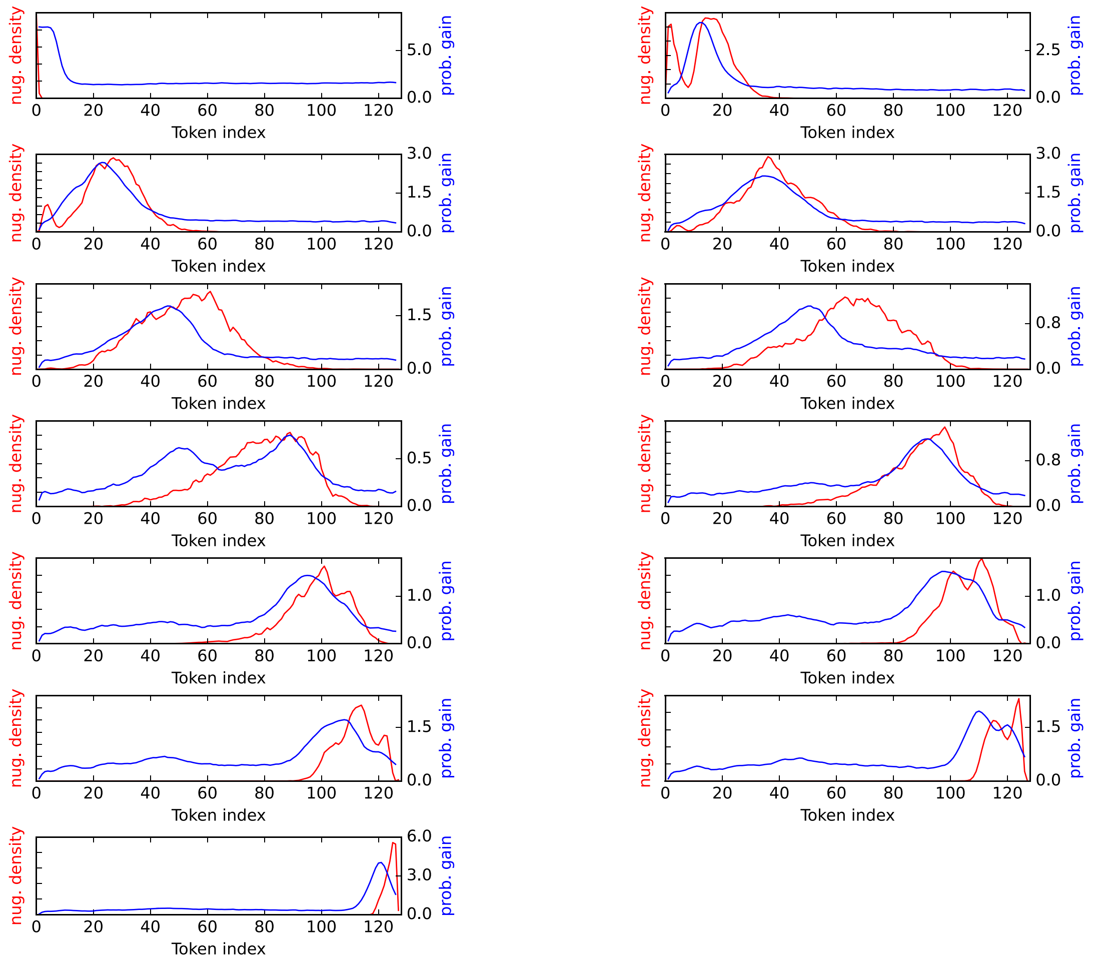

Figure 11. The probability gain on individual tokens vs the nugget location. We use the same notation as that in fig. 7. Each graph corresponds to one nugget in the texts, where nuggets are ordered by their indices on the original documents. Results are averaged over 10k documents, and only the first 13 nuggets in each document are considered.

B.2. Language model training

To be fair, each language model is initialized from a checkpoint of the AE model, even if they do not require the input of history nuggets. In practice, the transformer and the compressive transformer baselines are initialized from the AE model with $r = 0.1$. Thus, all models have the same number of parameters in the self-attention module, while the baseline models do not utilize the cross-attention part.

The WikiText-103 data are segmented into chunks of 128 tokens, and the model is trained to predict each segment based on a certain amount of history information. Note that during training, the training loss is calculated for all tokens in a segment in parallel, while during inference we input the model with as many preceding tokens as possible in the current segment to provide sufficient context, up to 128 tokens.

All models are trained with 4 NVIDIA RTX 6000 GPU cards with 24×4 GB GPU memories. Adam (Kingma & Ba, 2015) is used and is configured with a learning rate of 5×10^{-5}. It takes around 48 hours for a model with nuggets to converge. The model without nuggets, including the TSDAE baseline, can be faster to converge, taking around 24 hours.

C. Analysis of NUGGET encoding: Complete results

In this section, we show a complete version of fig. 7. We collect the first 13 nuggets in each document. Results are shown in fig. 11. Please refer to section 5.3 for a description of the experiments.

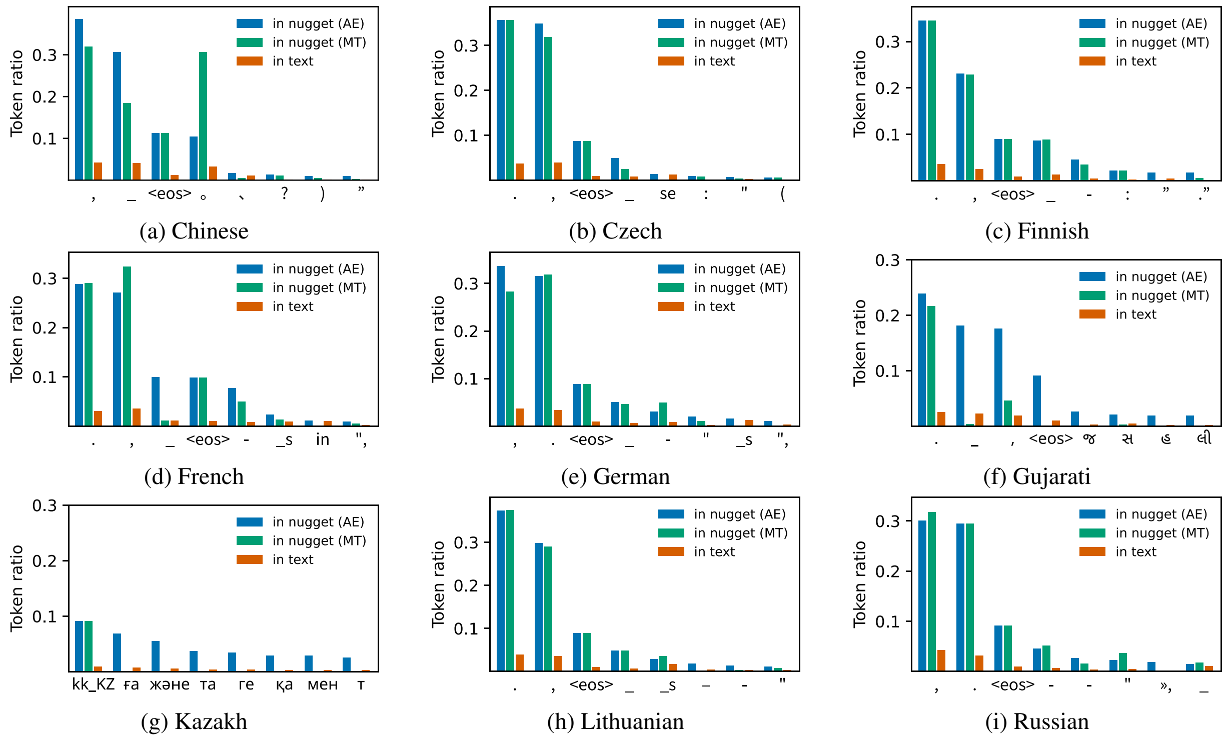

Figure 12. The token frequency in text and nuggets with training objectives of autoencoding (AE) and machine translation (MT). The experiments inherit the settings in fig. 5 and section 5.2 and are conducted in 9 other languages.

D. Nugget token distribution in languages other than English

In this section, we show the results of fig. 5 in languages other than English. We use all of the 9 languages from WMT19 (Barrault et al., 2019): Chinese, Czech, Finnish, French, German, Gujarati, Kazakh, Lithuanian, and Russian. Except that French is translated into German, other languages are all translated into English when the training objectives are set as machine translation. Note that Kazakh and Gujarati have much less training data than other languages and the training on them quickly stops. Results are shown in fig. 12. Please refer to section 5.2 for a description of the experiments.