From Characters to Words: Hierarchical Pre-trained Language Model for Open-vocabulary Language Understanding

Li Sun[1†], **Florian Luisier**[2], **Kayhan Batmanghelich**[1], **Dinei Florencio**[2], **Cha Zhang**[2]
[1]Boston University [2]Microsoft
[1]`{lisun,batman}@bu.edu`, [2]`{flluisie,dinei,chazhang}@microsoft.com`

Abstract

Current state-of-the-art models for natural language understanding require a preprocessing step to convert raw text into discrete tokens. This process known as tokenization relies on a pre-built vocabulary of words or sub-word morphemes. This fixed vocabulary limits the model's robustness to spelling errors and its capacity to adapt to new domains. In this work, we introduce a novel open-vocabulary language model that adopts a hierarchical two-level approach: one at the word level and another at the sequence level. Concretely, we design an intra-word module that uses a shallow Transformer architecture to learn word representations from their characters, and a deep inter-word Transformer module that contextualizes each word representation by attending to the entire word sequence. Our model thus directly operates on character sequences with explicit awareness of word boundaries, but without biased sub-word or word-level vocabulary. Experiments on various downstream tasks show that our method outperforms strong baselines. We also demonstrate that our hierarchical model is robust to textual corruption and domain shift.

1 Introduction

Pre-trained language models with Transformers have achieved breakthroughs in many natural language processing (NLP) tasks (Devlin et al., 2019; Liu et al., 2019). One of the key advantages of Transformers over traditional feature-engineered NLP pipelines is that Transformers enable end-to-end training from vast amount of data to automatically learn the optimal language representation (Mikolov et al., 2013b). However, most recent language models still require a separate preprocessing stage known as tokenization. Tokenization is a process that splits raw text parts into a list of discrete tokens from a fixed vocabulary. This pre-defined vocabulary remains as an important bottleneck preventing truly end-to-end training of language models (Tay et al., 2021; Islam et al., 2022).

	Word	BERT Tokens
Spelling Errors	changable (changeable)	ch, ##anga, ##ble (change, ##able)
	outragous (outrageous)	out, ##rag, ##ous (outrage, ##ous)
Domain Shift	reimbursement	re, ##im, ##bur, ##se, ##ment
	invoice	in, ##vo, ##ice

Table 1: Examples of sub-word tokenization of misspelled words (with their correct spellings in parentheses) and words from domain-specific corpus (e.g. business documents). Pre-trained sub-word tokenizer tends to over-segment words from these two categories, which produces less meaningful tokens.

Based on the granularity of the basic token units, tokenization methods can be divided into three categories: character-based, subword-based and word-based. A word-based tokenizer segments sentence into smaller chunks of words. Due to language complexity and memory limit, a word-based vocabulary can not represent all possible words. Word-level tokenization thus frequently runs into the issue of *out-of-vocabulary* words. A character-based tokenizer simply splits the text into a sequence of its characters. It is flexible to encode arbitrary words, but character-level tokenization produces long sequences, which is undesirable as the computational cost of Transformers grows quadratically with the sequence length. To strike a good balance between time and space complexity, most state-of-the-art pre-trained language models thus adopt sub-word tokenization. Data-driven sub-word tokenizers (Kudo and Richardson, 2018; Schuster and Nakajima, 2012; Kudo, 2018) are typically pre-trained on a general text corpus to learn a sub-word vocabulary based on the frequency of word fragments.

Despite their popularity, sub-word tokenizers

†Work done during internship at Microsoft.
PyTorch-like pseudocode is available in Appendix A.3.

limit the robustness and generalizability of the language models built upon them. First, sub-word tokenizers are sensitive to small textual perturbations (Xue et al., 2022). While humans can still comprehend text with subtle misspellings and capitalization variants (Rawlinson, 2007; Davis, 2003), these perturbations can drastically change the tokenization results, potentially leading to a sub-optimal text representation. Second, the sub-word vocabulary is pre-built and remains frozen during the language model pre-training and task-specific fine-tuning. Therefore, when adapting a pre-trained language model into a new language context (e.g. biomedical texts and business documents), the tokenizer is prone to excessive fragmentation of sub-word pieces (Yasunaga et al., 2022; Islam et al., 2022), as illustrated in Table 1. While this issue could be partially remedied by further task-specific pre-training or by collecting more fine-tuning data, such mitigation would be costly and not always possible.

We aim to bring the best of both character-based and word-based models to address the challenges discussed above. To this end, we propose a novel pre-trained language model with a hierarchical two-level architecture. At the word level, we split the text sequence by characters, and introduce an intra-word module that uses Transformers to learn a representation for each word in the sequence from the embeddings of their respective characters. At the sequence level, we introduce an inter-word module that contextualizes the embedding for every words in the text sequence. Our method does not require explicit sub-word or word-level vocabulary, and can thus be considered as an *open-vocabulary* approach (Mielke et al., 2021). By limiting the attention range to characters within the same word rather than the full sequence in the intra-word module, our model remains computationally efficient.

In order to validate our model, we comprehensively compare our method with various baseline methods, including the most popular sub-word based model BERT (Devlin et al., 2019), some state-of-the-art character-based models (Clark et al., 2022a; Boukkouri et al., 2020), and an hybrid character/sub-word model (Ma et al., 2020). Besides standard benchmarking, we also test the robustness of the various models in two ways: by introducing spelling noise into the validation set and by testing on cross-domain tasks.

Our contributions can be summarized as follows:

- We introduce a novel open-vocabulary pre-trained language model with a hierarchical two-level architecture. Our method does not rely on pre-defined word or sub-word vocabulary.

- We propose a novel adaptive and learnable aggregation method to summarize character-level features into word-level representations. An ablation study highlights its effectiveness.

- We show that our method outperforms strong baselines on multiple benchmarking datasets, while being computationally efficient.

- We perform quantitative experiments and a case study to show that our model is robust to textual corruption and domain shift.

2 Related Work

2.1 Word-level Models

Word embedding methods including Word2vec (Mikolov et al., 2013a) and GloVe (Pennington et al., 2014) led to many early NLP breakthroughs. These methods learn vector space representations of words from large-scale unlabeled corpora, and encode semantic relationships and meanings (Goldberg and Levy, 2014). In order to generalize to rare words, Bhatia et al. (2016) proposed to use LSTM to learn word embedding from both morphological structure and word distribution. While early methods only learned a context-independent word representation, ELMo (Peters et al., 2018) proposed to use a deep bidirectional language model to learn contextualized word representations. In more recent studies, Transformer-XL (Dai et al., 2019) enhanced the Transformer architecture with a recurrence mechanism to learn contextualized word embedding through language modeling. Despite the recent progress, word-level models still face the out-of-vocabulary challenge for noisy text and non-canonical word forms (Eisenstein, 2013).

2.2 Character-level Models

Character-level language models emerged in the early years thanks to their simplicity and ability to better address *out-of-vocabulary* words compared to word-level models (Elman, 1990; Graves, 2013; Kalchbrenner et al., 2016). While sub-word based approaches gained popularity in language modeling due to their superior performance, recent studies (Choe et al., 2019; Xue et al., 2022) show that character/byte-level models can match the performance of their sub-word counterpart when provided with sufficient parameter capacity. In addition, character-level models have been shown to be more robust to text corruptions (Tay et al., 2021), adversarial attacks, and domain shifts (Aguilar et al., 2020).

Character-level models also show promising results in multilingual settings. While sub-word or word tokenizers require a huge vocabulary to adequately cover various languages, multilingual character-based vocabulary can remain comprehensive and small. The text embedding layer does not eat up most of the model's parameter budget as in the multilingual BERT$_{Base}$ model for instance (up to 52%). More parameters can then be dedicated to the Transformer layers in character-based approaches. Character-level models have also been shown to perform better on low-resource languages (Islam et al., 2022).

An important drawback of character-level models is that they typically require more computations than sub-word and word-level models. This is because character-level tokenization produces longer token sequences compared to sub-word or word based approaches, and the computational and memory demands of the self-attention mechanism grow quadratically with the sequence length. In order to address this challenge, CANINE (Clark et al., 2022b) leverages strided convolution to downsample the character sequence, while Charformer (Tay et al., 2021) uses average pooling. Although these methods improve the computational efficiency of character-level models, they require a predefined static downsampling rate. Such downsampling operation often breaks the boundary of basic linguistic units, including morphemes and words.

2.3 Hybrid Models

Vanilla character-level models do not explicitly extract word or sub-word morpheme representations, which might negatively impact their performance on word-level downstream tasks, including named-entity recognition and extractive question answering. In order to address this issue, there have been efforts to combine character-level and word/sub-word level approaches to build hybrid models. These works propose to use information from character spelling to inform word representation. For example, Flair (Akbik et al., 2018) proposed to use the internal states of a pre-trained character language model to produce word-level embeddings. CharBERT (Ma et al., 2020) combined sub-word tokens and character tokens and fused their heterogeneous representations. CharacterBERT (Boukkouri et al., 2020) used a CNN to learn word-level representations from the embeddings of their characters, but still requires a word-level vocabulary for pretraining. Char2Subword (Aguilar et al., 2020) proposed a similar approach, where character embeddings are used to mimic pre-trained representation of sub-word tokens with Transformer encoder.

3 Method

Most character-level Transformer encoder models are sub-optimal for two reasons: (1) Dense self-attention on long character sequence is computationally expensive; (2) They do not leverage word boundary, which is an important inductive bias in linguistics. To overcome these challenges, we propose to decompose dense character-level Transformer encoder into two parts: intra-word Transformer encoder and inter-word Transformer encoder. Our hierarchical language model (HLM) adopts an hourglass structure (Nawrot et al., 2022) and contains three main components: (1) an intra-word module that learns word embeddings from their characters; (2) an inter-word module which contextualizes the word representations by attending to all words in the input sequence; (3) an intra-word prediction head for character-level pre-training. The overall architecture of our model is shown in Fig. 1. In the following sections, we discuss each component separately.

3.1 Intra-word Module

We aim to learn word-level representations from the embeddings of their characters. An ideal approach should be able to handle words of arbitrary lengths, attend to every character rather than a local window, and remain computationally efficient. Therefore, we choose a shallow (4 layers in our experiments) Transformer encoder to learn contextualized character embeddings, rather than a CNN or a LSTM used by previous methods (Boukkouri et al., 2020; Peters et al., 2018). Either average or max pooling (Boukkouri et al., 2020; Clark et al., 2022b) is commonly used to aggregate contextualized character embeddings and thus reduce the

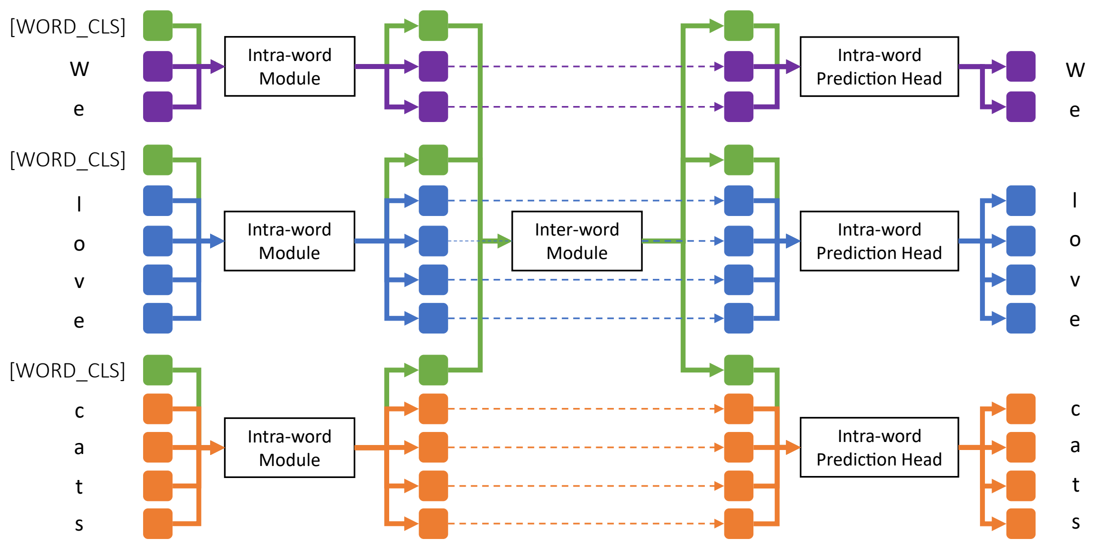

Figure 1: Overview of the proposed method. The intra-word module learns contextualized character embeddings by referring to characters from the same word. A [WORD_CLS] token is inserted at the beginning of each word to learn word-level representations. The inter-word module then learns contextualized word-level features by attending to all words in the sequence. Finally, the word-level and character-level embeddings are concatenated and fed to the intra-word prediction head for the pre-training task of masked character modeling. The prediction head is not used in downstream tasks.

sequence length. However, such simple pooling tends to wash out strong signals from particular morphemes (Fathi and Maleki Shoja, 2018). To address this challenge, we propose a novel adaptive and learnable aggregation method. Inspired by the approach of using the hidden state of the [CLS] token as the aggregate sequence-level representation, we insert a special [WORD_CLS] token at the beginning of every word. The embeddings of the [WORD_CLS] tokens are then used as word-level representations. Formally, for the i-th word of C_i characters in the sequence, we extract its word-level representation $\mathbf{r}^i$ as:

$$\mathbf{h}^i = f_\theta(\mathbf{e}_0^i \oplus \mathbf{e}_1^i \oplus \ldots \oplus \mathbf{e}_{C_i}^i)$$
$$\mathbf{r}^i = \mathbf{h}_0^i,$$

where f_θ is the intra-word Transformers that produces a contextualized representation $\mathbf{h}^i$ for each character of the i-th word, $\mathbf{e}_0^i$ is the embedding of the special [WORD_CLS] token, $\mathbf{e}_c^i$ is the c-th character embedding of the i-th word, and $\oplus$ denotes concatenation along the sequence dimension.

In Sec. 4.4, we conduct an ablation study to show that the proposed aggregation method outperforms the standard average or max pooling. By aggregating character-level tokens into word-level tokens, the token sequence length is greatly reduced for the subsequent inter-word module.

3.2 Inter-word Module

After obtaining word-level features, we apply an inter-word module consisting of deep transformer encoder layers to extract contextualized word-level representation by attending to all words in the sequences. Formally, the contextualized representation $\mathbf{w}^i$ of the i-th word of the sequence of N words is given as:

$$\mathbf{w}^i = f_\phi(\mathbf{r}^0 \oplus \ldots \oplus \mathbf{r}^{N-1}),$$

where f_ϕ denotes the inter-word Transformers.

We set the depth of the inter-word Transformer encoder stack to 12 in order to match the settings of BERT$_{Base}$ (Devlin et al., 2019) and CANINE (Clark et al., 2022b). The inter-word module contributes the most to the total model parameters.

3.3 Intra-word Prediction Head

Since we adopt an open-vocabulary approach, we propose to use character-level masked language modeling as pre-training task. To restore the character-level token sequence, we concatenate the contextualized character representations from the intra-word module (the initial [WORD_CLS] token is omitted) with the word-level features from the inter-word module along the sequence dimension. Finally, we apply a lightweight intra-word prediction head to get the posterior token probabilities.

Formally, the prediction of the C_i characters from the i-th word are given by:

$$\mathbf{c}^i = f_\sigma(\mathbf{w}^i \oplus \mathbf{h}_1^i \oplus \ldots \oplus \mathbf{h}_{C_i}^i),$$

where f_σ is the intra-word prediction head, consisting of a single Transformer layer, a fully-connected layer and a Softmax layer. Note that the intra-word prediction head is only used during pre-training for the masked character modeling task. During downstream fine-tuning, similar to CANINE, we concatenate initial word embedding $\mathbf{r}^i$ and contextualized word representation $\mathbf{w}^i$ along the feature dimension, and subsequently employ a small feed-forward network to integrate both low-level and high-level information for prediction.

3.4 Pre-training Task

Following the practice of BERT, we pre-train our model on English Wikipedia and BookCorpus dataset (19G) (Zhu et al., 2015). We pre-train the model for 3 epochs (3.9M steps with batch size set as 16) on a server with 8 NVIDIA Tesla V100 GPUs, and each epoch takes 137 hours. We adopt whole-word masked language modeling as pre-training task. In detail, we randomly select 15% of words from the input sequence, and mask every characters in the selected word. We replace the character tokens in 80% of the selected masked word with the `[MASK]` token. For 10% of the selected masked words, we replace their characters with randomly selected characters drawn from our character vocabulary. The remaining 10% words are unchanged. The three main components of our model are jointly trained in end-to-end fashion.

3.5 Implementation Details

We use spaCy (Honnibal et al., 2020) to split sentences into words, which is rule-based using space, punctuation and special rules (*e.g.* splitting *don't* into *do* and *n't*). We use a case-sensitive character vocabulary of size 1024, which consists of letters, digits and symbols. The maximum sequence length is set to 20 characters for the intra-word module and 512 words for the inter-word module. A `[CLS]` and a `[SEP]` token are inserted at the beginning and end of each sequence respectively. The hidden size is set to 768, the number of attention heads is set to 12, the feed-forward dimension in the Transformer encoder is set as 1536 and 3072 for intra-word and inter-word modules respectively. We leverage relative position (He et al., 2021) in our model, and we do not use token type embedding. GELU (Hendrycks and Gimpel, 2016) is used as activation function. Our model contains 125M parameters. We use the AdamW optimizer (Loshchilov and Hutter, 2018) for model pre-training and fine-tuning. For the pre-training, the weight decay is set to 0.01 and the number of warmup steps is set to 10,000. A linear learning rate decay schedule is used, starting at 5e-5. The dropout rate is set to 0.1. More algorithm details can be found in Appendix A.3.

4 Experiments

We evaluate the performance of our pre-trained model on a wide range of downstream tasks. We compare the performance of our pre-trained hierarchical language model (HLM) with various baseline methods, including the popular sub-word based BERT model, three recent byte/character-level models, as well as a hybrid model referred to as CharacterBERT. For BERT, we use the cased BERT$_{Base}$ model (108M parameters) to match our inter-word Transformers module setup. For CANINE, we adopt CANINE-C (132M) which also uses a character-level pre-training task. For CharacterBERT, we use the general version (105M) which is pre-trained on English Wikipedia and OpenWebText. For those baseline models, we use the pre-trained weights hosted on Huggingface[†] or released by the authors. For Charformer (203M) and Byte-level T5 (200M), we use results of the base version from the original paper as pre-trained weight is not available.

4.1 Evaluation on Standard Benchmarks

In order to assess our model's performance on general domain, we evaluate our methods on standard English NLP benchmarks, including Stanford Question Answering Dataset (SQuAD) task (Rajpurkar et al., 2016, 2018) and GLUE tasks (Wang et al., 2018). For the SQuAD task, we benchmark on both SQuAD 1.1 and 2.0 versions. SQuAD 1.1 dataset contains 100,000+ questions with associated context documents, and every question is answerable given the context. SQuAD 2.0 dataset contains an additional 50,000 unanswerable questions. We fine-tune the models for 2 epochs with a batch size of 16, and a learning rate of 3e-5. The evaluation on the validation set is shown in Table 2 (left). We use exact match (EM) and F1 scores as

[†]`https://huggingface.co/models`

Models	SQuAD 1.1 EM	SQuAD 1.1 F1	SQuAD 2.0 EM	SQuAD 2.0 F1	MRPC Acc	QNLI Acc	MNLI (m/mm) Acc
BERT (Devlin et al., 2019)	81.3	88.7	72.9	76.1	86.7	90.0	83.3/84.2
Byte-level T5† (Xue et al., 2022)	-	-	-	-	87.3	88.7	82.5/82.7
Charformer (Tay et al., 2021)	-	-	-	-	87.3	89.0	82.6/82.7
CANINE (Clark et al., 2022b)	72.9	82.1	66.6	70.3	84.8	84.6	76.9/78.2
CharacterBERT (Boukkouri et al., 2020)	79.9	87.5	71.5	74.6	84.1	89.9	81.9/82.6
CharBERT (Ma et al., 2020)	82.9	89.9	75.7	78.6	87.8	**91.7**	82.9/83.1
HLM (Ours)	**83.4**	**90.4**	**76.7**	**79.9**	**88.2**	90.8	**84.4/84.3**

- indicates not reported in the paper.

Table 2: Experimental results on the validation set of question answering and text classification tasks. We report exact match (EM) and F1 scores for SQuAD, and accuracy for text classification tasks.

the two evaluation metrics. Our method outperforms all the baseline methods on both SQuAD versions.

We also benchmark our model on three text classification tasks from the widely adopted GLUE tasks (Wang et al., 2018), including MNLI (Williams et al., 2018), MRPC (Dolan and Brockett, 2005) and QNLI (Rajpurkar et al., 2016). The MNLI dataset contains 393k training samples with textual entailment annotations. Given a sentence pair containing a premise and an hypothesis, the task is to predict whether the premise entails the hypothesis, contradicts the hypothesis, or neither. We conduct evaluation in both matched and mismatched settings. The MRPC dataset contains 3.7k of training sentence pairs, and the task is to predict whether the two sentences are semantically equivalent. The QNLI dataset contains 108k training samples of question-paragraph pairs, and the task is to predict whether the context sentence contains the answer to the question. We fine-tune the models on the datasets described above for 5 epochs, with a batch size of 16, and a learning rate of 2e-5. We use the accuracy as the evaluation metric. As shown in Table 2, our proposed method outperforms the baseline methods on all tasks.

In order to investigate the model's performance when the size is scaled up, we increase the size of our HLM to match BERT$_{Large}$ and benchmark the performance. Our HLM$_{Large}$ performs competitively with the baseline method given a limited computational budget. More details and preliminary experimental results can be found in Appendix A.2.

4.2 Robustness to Textual Corruptions

Humans are prone to making spelling mistakes. For example, 10-15% of web search queries contain misspellings (Dalianis, 2002; Cucerzan and Brill, 2004). In order to test our model's robustness to misspellings, we add synthetic noise to the fine-tuning and evaluation set of downstream tasks and re-evaluate all the models.

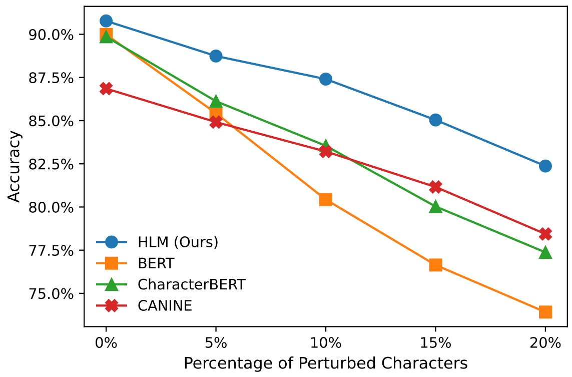

Figure 2: Evaluation results on QNLI under different amounts of noise. With increasing amount of perturbation, our HLM consistently outperforms baseline methods, and the performance drop is smaller than BERT.

Following the practice of Xue et al. (2022), we experiment with three types of noises: (1) *Random drop*: We randomly delete 10% of characters (spaces and punctuation are included) from the input sequence; (2) *Random repeat*: We randomly select 20% of characters, then append 1-3 repetitions (with equal probability) after the the selected original characters; (3) *Random case*: We randomly set the case for each character (upper or lower) in the input sequence.

We perform the perturbation experiments on two representative downstream tasks: text classification on MNLI dataset and question answering on SQuAD 2.0. For the MNLI dataset, we add noise to both premise and hypothesis sentences. For the SQuAD 2.0 dataset, we only apply the perturbations to the question sentence, but not to the context paragraph, in order to avoid copying corrupted answer from the context for extractive QA mod-

†Use results from Tay et al. (2021)

	Model	MNLI(m/mm)		SQuAD 2.0	
		Matched Acc	Mismatched Acc	EM	F1
Random drop	BERT	68.6(−14.7)	70.0(−14.2)	41.4(−30.6)	47.6(−27.6)
	CANINE	69.1(−7.8)	69.9(−8.3)	61.8(−4.7)	64.8(−5.4)
	CharacterBERT	70.1(−11.8)	71.1(−11.5)	61.5(−9.9)	64.4(−10.3)
	HLM (Ours)	**74.7**(−9.7)	**75.2**(−8.9)	**69.3**(−7.4)	**72.2**(−7.6)
Random repeat	BERT	72.3(−11.0)	75.4(−8.8)	35.2(−36.8)	37.2(−38.0)
	CANINE	76.3(−0.7)	77.1(−1.2)	65.3(−1.3)	68.5(−1.7)
	CharacterBERT	77.4(−4.4)	78.2(−4.4)	66.3(−5.1)	69.4(−5.3)
	HLM (Ours)	**83.1**(−1.3)	**82.8**(−1.4)	**75.8**(−0.9)	**78.9**(−1.0)
Random case	BERT	71.2(−12.1)	71.2(−13.0)	35.9(−36.2)	37.5(−37.7)
	CANINE	76.7(−0.2)	78.0(−0.2)	66.1(−0.5)	69.8(−0.5)
	HLM (Ours)	**83.5**(−0.9)	**83.5**(−0.7)	**76.3**(−0.4)	**79.3**(−0.5)

Table 3: Evaluation of models under various types of learnable noise. We apply the perturbations to both fine-tuning data and evaluation set. We report the performance value and degradation compared to the standard evaluation in parentheses. Bold face indicates the best absolute performance. We do not report results for randomly switching case for CharacterBERT as it is an uncased model.

els. The evaluation results are shown in Table 3. We found that BERT's performance significantly drops under perturbation, one explanation being that even subtle misspellings would greatly change the sub-word tokenization results. In comparison, character-level models including CANINE degrade less in the presence of noise. We also present the results for unseen perturbation setting in Appendix A.4. Overall, our proposed HLM is robust to different kinds of perturbation and achieves the best performance.

In order to access the model's robustness to various magnitude of perturbations, we add different amounts of noise to the QNLI dataset and perform the evaluation. In practice, we randomly sample 5%, 10%, 15%, 20% of characters for each example in the finetuning data and validation set. For each selected character, we either drop the character or repeat the character as mentioned above (equal probability). The accuracy on the validation set is shown in Fig. 2.

4.3 Robustness to Domain Shift

Most generic language models are pre-trained on web-crawled text corpora including Wikipedia and Common Crawl. But in real world deployments, models are often used in a different domain, an issue referred to as *domain shift*. In order to evaluate the robustness to domain shift, we fine-tune and evaluate the pre-trained models on downstream tasks from specialized domains including biomedicine and social media. For the biomedical field, we perform the evaluation on the NCBI-disease dataset (Crichton et al., 2017; Gu et al., 2021), which contains 7,287 sentences annotated with disease mentions from PubMed abstracts. The task is framed as a named entity recognition (NER) problem where the entities are the disease mentions. We fine-tune the models for 20 epochs, with a batch size of 16, and a learning rate of 2e-5. For the social media experiment, we leverage the W-NUT16 NER shared task (Strauss et al., 2016). This dataset contains 7,244 tweets annotated with 10 NER categories, including person, location, company and others. We fine-tune the models for 5 epochs. The evaluation results on the test sets are shown in Table 4. We use the F1 score as the evaluation metric. As observed, the proposed HLM outperforms the baseline methods, highlighting its higher robustness to domain shift.

Model	NCBI-disease (F1)	W-NUT16 (F1)
BERT	83.8	45.7
CANINE	75.2	32.0
CharacterBERT	84.7	34.0
HLM (Ours)	**86.4**	**47.9**

Table 4: Evaluation results on cross-domain tasks. We report F1 score on the test set as the evaluation metric.

Case study In order to understand the performance gain of our model over sub-word based BERT on cross-domain tasks, we look into the cases where BERT makes incorrect predictions. We found that many of these cases contain excessively fragmented words. Table 5 shows two examples from the NCBI-disease NER task. The word *fragility* in case 1 is segmented into *f, ##rag, ##ility*, and the word *rupture* in case 2 is segmented into *r, ##up, ##ture*. We think these tokenization results

Text	*Skin*	*fragility*	*in*	*most*	*cases*	*is*	*due*	*to*	*mutations*	*in*	*the*	*gene*	*encoding*	...
BERT tokens	Skin	f, ##rag, ##ility	in	most	cases	is	due	to	mutations	in	the	gene	encoding	
BERT	O	O	O	O	O	O	O	O	O	O	O	O	O	
HLM (Ours)	B	I	O	O	O	O	O	O	O	O	O	O	O	
Label	B	I	O	O	O	O	O	O	O	O	O	O	O	

Text	...	*a*	*disease*	*leading*	*to*	*aortic*	*rupture*	*in*	*early*	*adult*	*life*
BERT tokens		a	disease	leading	to	a, ##ort, ##ic	r, ##up, ##ture	in	early	adult	life
BERT		O	O	O	O	O	O	O	O	O	O
HLM (Ours)		O	O	O	O	B	I	O	O	O	O
Label		O	O	O	O	B	I	O	O	O	O

Table 5: Case study of two examples from NCBI-disease NER task. The tagging schema for disease entities are beginning (B), inside (I), and outside (O). Pink/green colors indicate incorrect/correct predictions respectively.

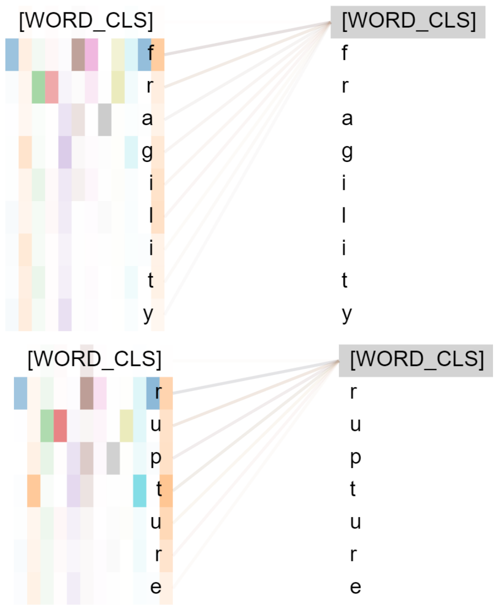

Figure 3: Visualization of the attention patterns at the last layer of our intra-word module. Colored rectangles indicate the 12 attention heads. Color brightness and line weight reflect the attention scores. In the two examples, the `[WORD_CLS]` token is mainly attended by the stems of the words, *fragil* and *rupt*, respectively.

are sub-optimal as they break word morphemes, which possibly explains BERT's mispredictions. In comparison, we use BertViz (Vig, 2019) to visualize the behavior of our HLM model. Specifically, we visualize the attention patterns of the `[WORD_CLS]` token of the last Transformer layer of our intra-word module. As shown in Fig. 3, the `[WORD_CLS]` token for the word *fragility* and *rupture* are primarily attended by the character string *fragil* and *rupt* respectively, which are the stems of the words.

Dataset	Average pooling	Max pooling	Ours
MRPC (Acc)	82.1	83.6	**86.0**
NCBI-disease (F1)	85.3	85.9	**86.6**

Table 6: Word-level aggregation comparisons. All models are pre-trained for 1.5 epochs.

Model	Throughput (sample/sec)
BERT	**93.8**
CANINE	44.3
CharacterBERT	78.4
HLM (Ours)	90.3

Table 7: Evaluation results on computational efficiency.

4.4 Ablation Study

In this section, we perform an ablation study to compare the effect of different word-level aggregation methods. Specifically, we replace the proposed special token learning-based aggregation with standard aggregation methods such as average pooling and max pooling. We did not implement the strided convolution proposed in CANINE as it can not handle the variable word lengths. We report the validation accuracy on MRPC and the test F1 score on NCBI-disease in Table 6. Our learned aggregation outperforms the standard pooling strategies. Note that average and max pooling are usually performed on a fixed-length window of characters in previous studies (Tay et al., 2021), not adaptively at the word-level as in our ablation study.

4.5 Computational Efficiency

In this section, we benchmark the computational efficiency of the proposed model. Specifically, we measure the inference throughput (number of processed samples per second) on the test set of the MRPC dataset, a sub-task of the GLUE benchmark.

We evaluate different models on the same server with one NVIDIA Tesla V100 GPU. The batch size is set to 32 and we use single precision. The evaluation results are shown in Table 7. While BERT is the most computationally efficient, our HLM also performs competitively, the performance gap being smaller compared to other character-level baseline models. We speculate that this performance gain comes from our hierarchical architecture. By aggregating character tokens into word-level tokens, the sequence length is drastically reduced for the inter-word module which has the deepest Transformer stack. We provide more analysis on the computational complexity in Appendix A.1.

5 Conclusion

In this work, we propose a novel hierarchical language model for open-vocabulary language understanding. Our method does not rely on explicit sub-word or word vocabulary. We demonstrate that our HLM model outperforms baseline methods on standard benchmarks, and highlight its robustness to spelling errors and domain shifts. In future work, we will expand our language support and explore incorporating a decoder for generative tasks.

Limitations

This work has two main limitations. First, we only consider baseline models with similar amount of parameters, and pre-trained on similar scale of text corpus for comparison. While we are aware of recent models including T5 (Raffel et al., 2020) and PaLM (Chowdhery et al., 2022), they either use huge corpus like C4 (745GB text) for pre-training or contain significantly more parameters than ours. In the future, we will try to find additional computational resources to scale up our model and pre-train on larger text corpus. Second, we leverage spaCy to segment sentences into words, which is rule-based using spaces, punctuations and other rules. This approach works well on English and many other common languages such as French, German and Spanish. But for a few languages that do not use spaces to split words (e.g. Chinese and Japanese), it will be challenging to retrieve word boundaries. To address this issue, we consider either falling back to character splitting for these languages (similar to multilingual BERT) or employing a more sophisticated word boundary detector in future work.

Acknowledgements

This work was partially supported by NIH Award Number 1R01HL141813-01 and the Pennsylvania Department of Health. We are grateful for the computational resources provided by Pittsburgh Super Computing grant number TGASC170024.

References

Gustavo Aguilar, Bryan McCann, Tong Niu, Nazneen Rajani, Nitish Keskar, and Thamar Solorio. 2020. Char2subword: Extending the subword embedding space using robust character compositionality. *arXiv preprint arXiv:2010.12730*.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Parminder Bhatia, Robert Guthrie, and Jacob Eisenstein. 2016. Morphological priors for probabilistic neural word embeddings. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 490–500, Austin, Texas. Association for Computational Linguistics.

Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, Hiroshi Noji, Pierre Zweigenbaum, and Junichi Tsujii. 2020. Characterbert: Reconciling elmo and bert for word-level open-vocabulary representations from characters. *arXiv preprint arXiv:2010.10392*.

Dokook Choe, Rami Al-Rfou, Mandy Guo, Heeyoung Lee, and Noah Constant. 2019. Bridging the gap for tokenizer-free language models. *arXiv preprint arXiv:1908.10322*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Jonathan H Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022a. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022b. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Gamal Crichton, Sampo Pyysalo, Billy Chiu, and Anna Korhonen. 2017. A neural network multi-task learning approach to biomedical named entity recognition. *BMC bioinformatics*, 18(1):1–14.

Silviu Cucerzan and Eric Brill. 2004. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 293–300, Barcelona, Spain. Association for Computational Linguistics.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Hercules Dalianis. 2002. Evaluating a spelling support in a search engine. In *International Conference on Application of Natural Language to Information Systems*, pages 183–190. Springer.

Matt Davis. 2003. Psycholinguistic evidence on scrambled letters in reading.

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.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*.

Jacob Eisenstein. 2013. What to do about bad language on the internet. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 359–369, Atlanta, Georgia. Association for Computational Linguistics.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Ehsan Fathi and Babak Maleki Shoja. 2018. Chapter 9 - deep neural networks for natural language processing. In Venkat N. Gudivada and C.R. Rao, editors, *Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications*, volume 38 of *Handbook of Statistics*, pages 229–316. Elsevier.

Yoav Goldberg and Omer Levy. 2014. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. 2021. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1):1–23.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spacy: Industrial-strength natural language processing in python.

Md Mofijul Islam, Gustavo Aguilar, Pragaash Ponnusamy, Clint Solomon Mathialagan, Chengyuan Ma, and Chenlei Guo. 2022. A vocabulary-free multilingual neural tokenizer for end-to-end task learning. *arXiv preprint arXiv:2204.10815*.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

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. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Wentao Ma, Yiming Cui, Chenglei Si, Ting Liu, Shijin Wang, and Guoping Hu. 2020. Charbert: character-aware pre-trained language model. *arXiv preprint arXiv:2011.01513*.

Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. 2021. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2022. Hierarchical transformers are more efficient language models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1559–1571, Seattle, United States. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Graham Rawlinson. 2007. The significance of letter position in word recognition. *IEEE Aerospace and Electronic Systems Magazine*, 22(1):26–27.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE.

Benjamin Strauss, Bethany Toma, Alan Ritter, Marie-Catherine de Marneffe, and Wei Xu. 2016. Results of the WNUT16 named entity recognition shared task. In *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, pages 138–144, Osaka, Japan. The COLING 2016 Organizing Committee.

Yi Tay, Vinh Q Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2021. Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672*.

Jesse Vig. 2019. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 37–42, Florence, Italy. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Michihiro Yasunaga, Jure Leskovec, and Percy Liang. 2022. LinkBERT: Pretraining language models with document links. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8003–8016, Dublin, Ireland. Association for Computational Linguistics.

Zhanpeng Zeng, Yunyang Xiong, Sathya Ravi, Shailesh Acharya, Glenn M Fung, and Vikas Singh. 2021. You only sample (almost) once: Linear cost self-attention via bernoulli sampling. In *International conference on machine learning*, pages 12321–12332. PMLR.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

A Appendix

A.1 Analysis on Computational Complexity

Let N denotes the character length of input sequence. Without loss of generality, we assume the words in the sequence are of the same length M. The multi-head self-attention module is the major component of Transformer. While it provides global receptive field, the computational cost and memory footprint grow quadratically with input sequence length (Zeng et al., 2021). Therefore, for a vanilla character-based Transformers with dense self-attention, the computational and space complexity is $O(N^2)$.

For our proposed HLM, the input sequence is still at the character level. But we sparsity the dense self-attention by introducing a hierarchical architecture. For the intra-word module, each character token only attends to characters from the same word. Since there are $\frac{N}{M}$ words in the sequence, the computational and space complexity of the intra-word module is

$$O\left(\frac{N}{M}\cdot M^2\right) = O(NM) \tag{1}$$

For the inter-word module, since it only operates on word-level tokens, the computational and space complexity is

$$O\left(\frac{N^2}{M^2}\right) \tag{2}$$

Since typically $N \gg M$, and we have a shallow intra-word module and a deeper inter-word module, Eq. 2 dominates the computational and space complexity of the full model, which is significantly lower than the vanilla character-level model.

In comparison to sub-word based models like BERT, our inter-word module operates on word-level token sequence, which is always equal or shorter than sub-word level token sequence. Therefore, despite our model has an extra intra-word module, we empirically observe in Table 7 that our HLM is competitive in terms of computational efficiency comparing to sub-word based models.

A.2 Preliminary Evaluation of Scaled Model

In this section, we scale up our model size and benchmark the performance. In order to match BERT$_{Large}$, we set the number of layers in the inter-word Transformer encoder to 24 and the feed-forward dimension of Transformer encoder is set as 2048 and 4096 for intra-word and inter-word modules respectively. We set the number of attention heads as 16 and the hidden size as 1024. The batch size is set as 128. Other hyperparameters are set as the same as HLM$_{Base}$, described in Section 3. Due to limited access to computational resources, we could only pre-train the model for 370k steps at the camera-ready deadline. In comparison, BERT$_{Large}$ was pretrained for 1M steps with a batch size of 256. Therefore, our computational budget is about 1/6 of BERT's. We benchmark our model's performance on SQuAD 1.1 and 2.0 datasets. The evaluation results on the validation set are shown in Table 8. We use exact match (EM) and F1 scores as the two evaluation metrics. Our model performs competitively compared with BERT$_{Large}$, despite that our HLM$_{Large}$ has significantly less computational budget for pre-training.

	SQuAD 1.1		SQuAD 2.0	
Models	EM	F1	EM	F1
BERT$_{Large}$	84.1	90.9	78.7	81.9
HLM$_{Large}$ (370k steps)	83.4	90.2	78.2	81.3

Table 8: Experimental results on the validation set of question answering tasks. We report exact match (EM) and F1 scores for SQuAD.

A.3 Algorithm Details

In this section, we provide algorithm details for our input pre-processing and model algorithm. Our pre-processing consists the following steps. First, we split each sentence into a list of words. Next, we map characters to codepoint indexes using a character-level vocabulary, and insert `[WORD_CLS]` token at the start of each word. Next, we insert a `[CLS]` token at the start, and a `[SEP]` token at the end for each sequence. Then we truncate the token sequence based on both character-level (20 characters for each word) and word-level (512 words per sentence) limits. Next, we compute the maximum number of characters for words in the batch, and pad all words to this length. We also determine the maximum number of words in the sequence batch, and pad all sequences to this length. The pre-processed batch can then be represented as a matrix of shape `[batch_size, max_num_word, max_num_char]`. Our unique representation of text batch enables us to efficiently switch between performing intra-word self-attention and inter-word self-attention by simply reshaping, which is shown in Algorithm 1.

We provide pseudocode for pre-training of our HLM in Algorithm 1. For better readability, we omit implementation details including utilizing attention mask which avoids performing attention on the [PAD] tokens and handling for padding words. We recommend padding the input matrix to multiples of 8 for better acceleration on GPU. We also found a residual connection between initial word embedding $\mathbf{r}^i$ and contextualized word embedding $\mathbf{w}^i$ improves the performance in a subsequent study.

A.4 Robustness to Unseen Perturbations

In this section, we benchmark the model's robustness to *unseen noise*. Specifically, we only add noise to the evaluation set, while using the original fine-tuning data. We experiment with three types of perturbation as introduced in Section 4.2. The results are shown in Table 9. In all three scenarios, our proposed HLM outperforms baseline methods, showing better robustness.

Algorithm 1 Pseudocode for HLM, PyTorch-like

```
# embeddings: character-level embedding lookup table
# intra_word_encoder: Intra-word Transformer encoder
# inter_word_encoder: Inter-word Transformer encoder
# intra_word_head: Intra-word prediction head

for input_ids, labels in loader: # load a minibatch with n samples
    input_embeds = embeddings(input_ids)
    batch_size, num_word, num_char, hidden_size = input_embeds.shape

    # reshape to let Transformers attend to intra-word tokens rather than full sequence
    input_embeds = input_embeds.reshape((batch_size*num_word, num_char, hidden_size))
    initial_embeds = intra_word_encoder(input_embeds)

    # extract embedding for [WORD_CLS] token, which is always at the beginning of each word
    word_embeds = initial_embeds[:,0,:]

    # reshape and extract contextualized inter-word representation
    word_embeds = word_embeds.reshape((batch_size, num_word, hidden_size))
    word_embeds = inter_word_encoder(word_embeds)
    word_embeds = word_embeds.reshape((batch_size*num_word, 1, hidden_size))

    # concatenate to restore the character-level token sequence
    char_embeds = concatenate([word_embeds, initial_embeds[:,1:,:]], axis=1)
    char_logits = intra_word_head(char_embeds)
    char_logits = char_logits.reshape((batch_size, num_word, num_char, -1))

    loss = CrossEntropyLoss(char_logits, labels) # masked character modeling loss
    loss.backward() # back-propagate
    # AdamW update
    update(embeddings, intra_word_encoder, inter_word_encoder, intra_word_head)
```

		MNLI(m/mm)		**SQuAD 2.0**	
	Model	Matched Acc	Mismatched Acc	EM	F1
Random drop	BERT	57.5(−25.8)	57.9(−26.3)	53.1(−19.0)	55.6(−19.6)
	CANINE	57.7(−19.2)	58.2(−20.1)	57.1(−9.5)	59.0(−11.3)
	CharacterBERT	55.9(−26.0)	56.0(−26.6)	52.0(−19.5)	55.1(−19.5)
	HLM (Ours)	**59.7**(−24.7)	**61.0**(−23.2)	**58.3**(−18.4)	**60.3**(−19.5)
Random repeat	BERT	52.4(−30.9)	53.5(−30.7)	51.4(−20.7)	52.7(−22.5)
	CANINE	56.2(−20.7)	57.4(−20.8)	53.8(−12.8)	56.1(−14.2)
	CharacterBERT	54.5(−27.4)	55.2(−27.4)	49.4(−22.1)	52.6(−22.0)
	HLM (Ours)	**58.5**(−25.9)	**58.3**(−25.9)	**57.7**(−19.1)	**59.2**(−20.7)
Random case	BERT	43.8(−39.5)	44.1(−40.2)	48.1(−23.9)	48.4(−26.8)
	CANINE	72.7(−4.2)	73.2(−5.1)	65.3(−1.3)	68.6(−1.6)
	HLM (Ours)	**73.5**(−10.9)	**74.5**(−9.6)	**70.2**(−6.5)	**73.1**(−6.8)

Table 9: Evaluation of the models under various types of unseen noise. The perturbations are only applied to the evaluation sets, while the fine-tuning data is left untouched. We report the performance value and degradation compared to the standard evaluation (no perturbation) in parentheses. Bold face indicates the best absolute performance. We do not report results for randomly switching case for CharacterBERT as it is an uncased model.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
We discuss the limitations in Section 7.

☒ A2. Did you discuss any potential risks of your work?
We think there is a minimal potential risk in our work, since our model is not a generative language model.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
We summarize the main claims in Section 1.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
Not applicable. Left blank.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Not applicable. Left blank.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Not applicable. Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Not applicable. Left blank.

C ☑ Did you run computational experiments?

In section 4.

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
In section 3.5.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
In section 3.5.

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
In section 4.

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
In section 3.5.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?
Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.