LANGUAGE MODELING WITH PIXELS

Fatemeh Saberi Khomami

OUTLINE

- Language models challenges
- A proposed solution
- What is a PIXEL ?
- How does a PIXEL work?
- Image reconstruction after pretraining
- Finetuning PIXEL
- PIXEL - syntactic task
- PIXEL - semantic task
- Robustness of PIXEL
- Conclusion
- References

LANGUAGE MODELS CHALLENGES

- Word-based vocabulary has the following issues:
  - it is not possible to encode out-of-vocabulary words and we loss information
  - too many parameters in the word embedding layer
  - estimating the probability distribution over the vocabulary is an expensive computation
- Character-based vocabulary has the following issue:
  - causes increased sequence lengths
- Subword-based vocabulary have solved the above problems for the monolingual context
- Subword-based vocabulary has a vocabulary bottle-neck when used for multilingual context
- Tackling the above issue creates two more challenges, a trade off between what can be represented in the embedding matrix and computational issues in the output layer

A PROPOSED SOLUTION

- Consider language modeling as a visual reconstruction task
- This will free us from a vocabulary and its corresponding issues
- The mentioned technique is called a PIXEL model

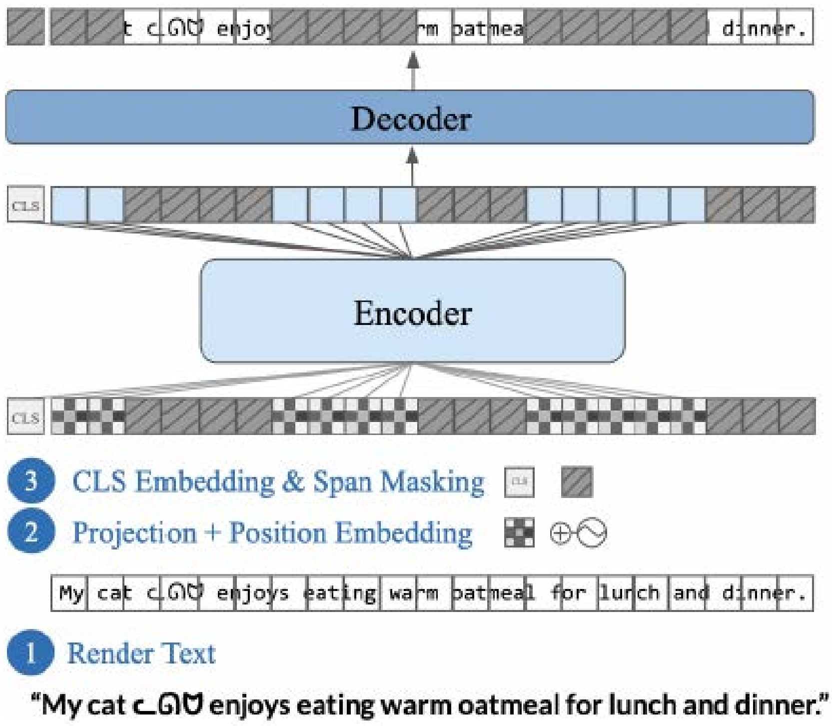
WHAT IS A PIXEL ?

- PIXEL is a Pixel-based Encoder of Language. It is build on the Masked Autoencoding Visual Transformers (ViT-MAE)
- PIXEL is a new type of language model that can theoretically support any language that can be typeset by a modern computer.
- PIXEL does not have a vocabulary embedding layer, instead, it renders text as a sequence of fixed-sized patches and processes the patches using a Vision Transformer encoder.
- PIXEL also does not have a computationally expensive output layer when it reconstructs the pixels of the masked patches.

HOW DOES A PIXEL WORK ?

PIXEL consists of three major components:

1. A text renderer
   draws text as an image
2. An encoder
   encodes the unmasked regions of the rendered image
3. A decoder
   reconstructs the masked regions at the pixel level



HOW DOES A PIXEL WORK ? -TEXT RENDERER

- The key component of PIXEL is the text renderer that converts sequences of texts into patches of consecutively 16*16 RGB photos. (useful to accurately represent colour emoji)
- Uses a black 16*16 patch to serve as a separator and an end-of-sequence (EOS) marker
- Sequences longer than the maximum length are either truncated or split into multiple sequences
- The renderer supports:
  - color emoji
  - hieroglyphs scripts
  - left-to-right and right-to-left writing systems

HOW DOES A PIXEL WORK ? -ARCHITECTURE

- PIXEL-base is a 112M parameter ViT-MAE architecture
  - with a12-layer ViT encoder : with 86M parameters
  - and an 8-layer Transformer decoder : with 26M parameters

- PIXEL-base is pretrained on a rendered version of the English Wikipedia and the Bookcorpus. In total they have 3.1B words which will be rendered into 16.4M examples. The batch size is 256, hence the model will take 64,062 steps per epoch. The pretraining is 16 epochs long.

- Training PIXEL took 8 days on 8*40GB Nvidia A100 GPUs

HOW DOES A PIXEL WORK ? -ARCHITECTURE

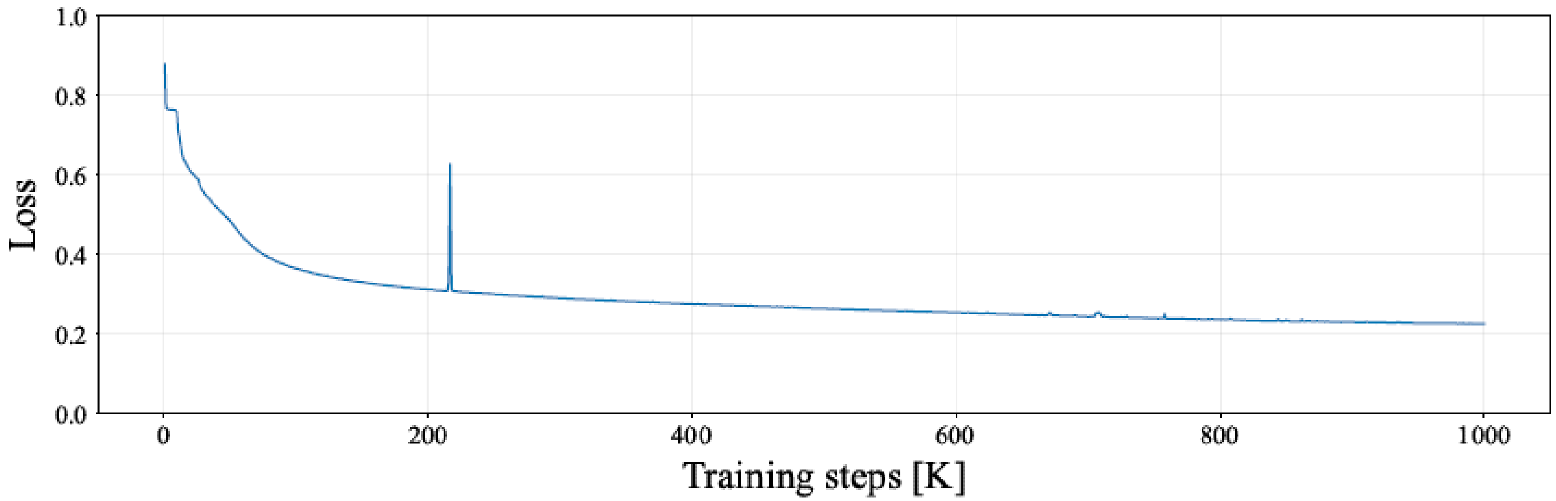


Figure 7: PIXEL pretraining loss curve

HOW DOES A PIXEL WORK ? - PATCH EMBEDDING

- The images produced by the text renderer projected to obtain a sequence of patch embeddings with a 16 * 16 pixel resolution, to which fixed sinusoidal position embeddings are added.

- Fixed sinusoidal position embeddings are added to inject order information.

HOW DOES A PIXEL WORK ? - PATCH SPAN MASKING

- In ViT-MAE we should mask some of the patches.
- The span masking masks more meaningful units of text
- We found that 25% masking ratio works well for PIXEL-base

Algorithm 1 PIXEL Span Masking

Input: #Image patches N, masking ratio R,
maximum masked span length S, span length
cumulative weights $W = \{w_1, \ldots, w_S\}$
Output: Masked patches $\mathcal{M}$

$\mathcal{M} \leftarrow \emptyset$
repeat
  $s \leftarrow \text{randchoice}(\{1, \ldots, S\}, W)$
  $l \leftarrow \text{randint}(0, \max(0, N - s))$
  $r \leftarrow l + s$
  if $\mathcal{M} \cap \{l - s, \ldots, l - 1\} = \emptyset$ **and**
   $\mathcal{M} \cap \{r + 1, \ldots, r + s\} = \emptyset$ **then**
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{l, \ldots, r\}$
  end if
until $|\mathcal{M}| > R \cdot N$
return $\mathcal{M}$

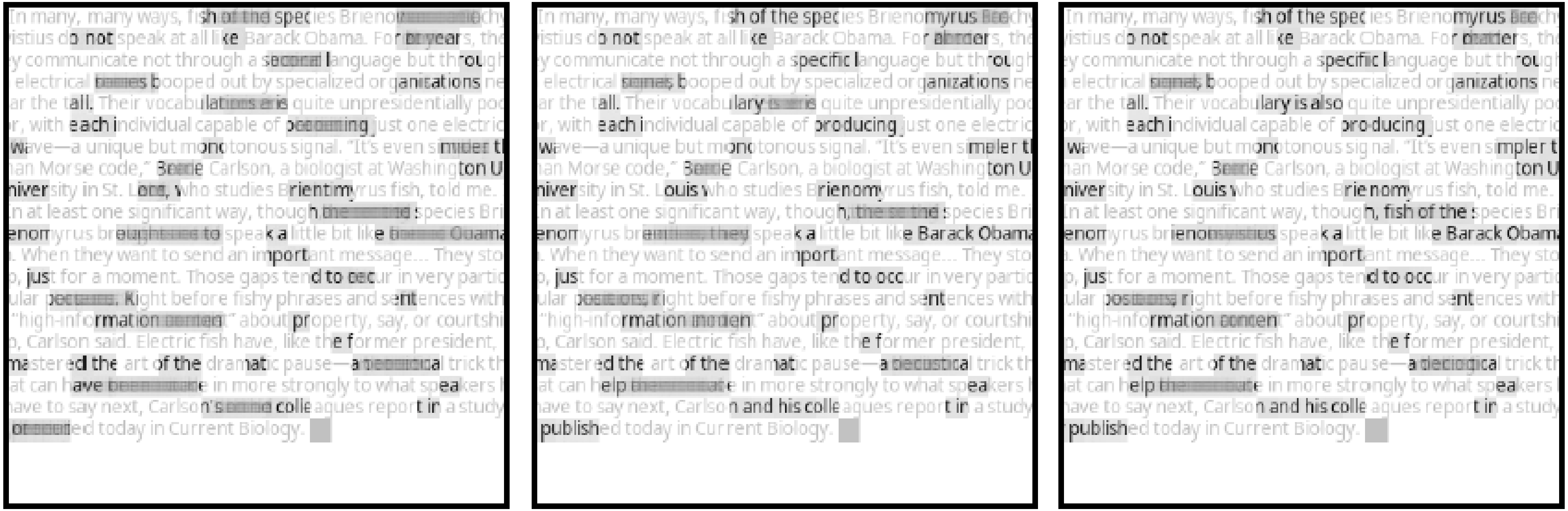
HOW DOES A PIXEL WORK ? - ENCODER

- The encoder only processes unmasked patches
- By not using masked patches, encoder is avoiding a mismatch between pretraining and finetuning
- We also prepend the special CLS embedding to the unmasked patches.
- The resulting CLS and unmasked patches are processed by a 12-layer Transformer encoder to produce a sequence of encoder output representations.

HOW DOES A PIXEL WORK ? - DECODER

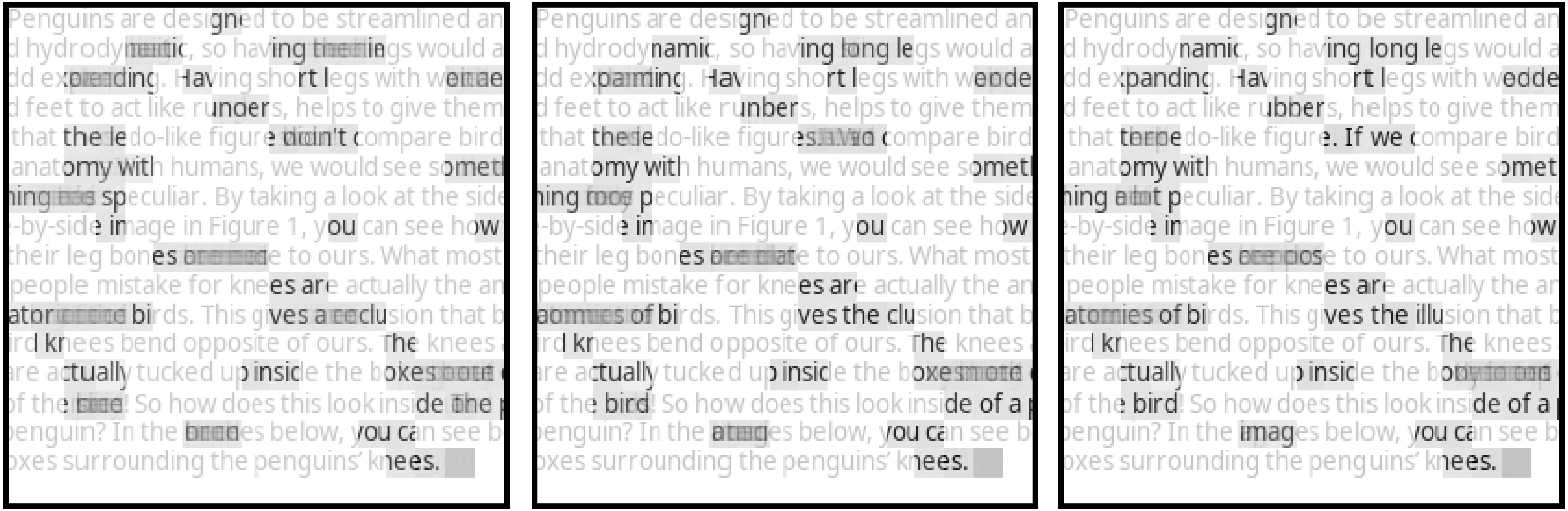
- The PIXEL decoder first projects the encoder outputs into the same space as the decoder model hidden size
- It then inserts learnable mask embeddings at the masked positions, these are what PIXEL tries to reconstruct at the pixel level
- Fixed sinusoidal position embeddings are added to inject order information
- The result of the above steps will be processed via 8 Transformer layers
- The encoder circumvents the question of whether to tie the subword embedding weights
- PIXEL measures the discrepancy between target image patches and reconstructed patches by measuring the MSE
- This loss is only computed for masked, non-blank (text) patches.

IMAGE RECONSTRUCTION AFTER PRETRAINING



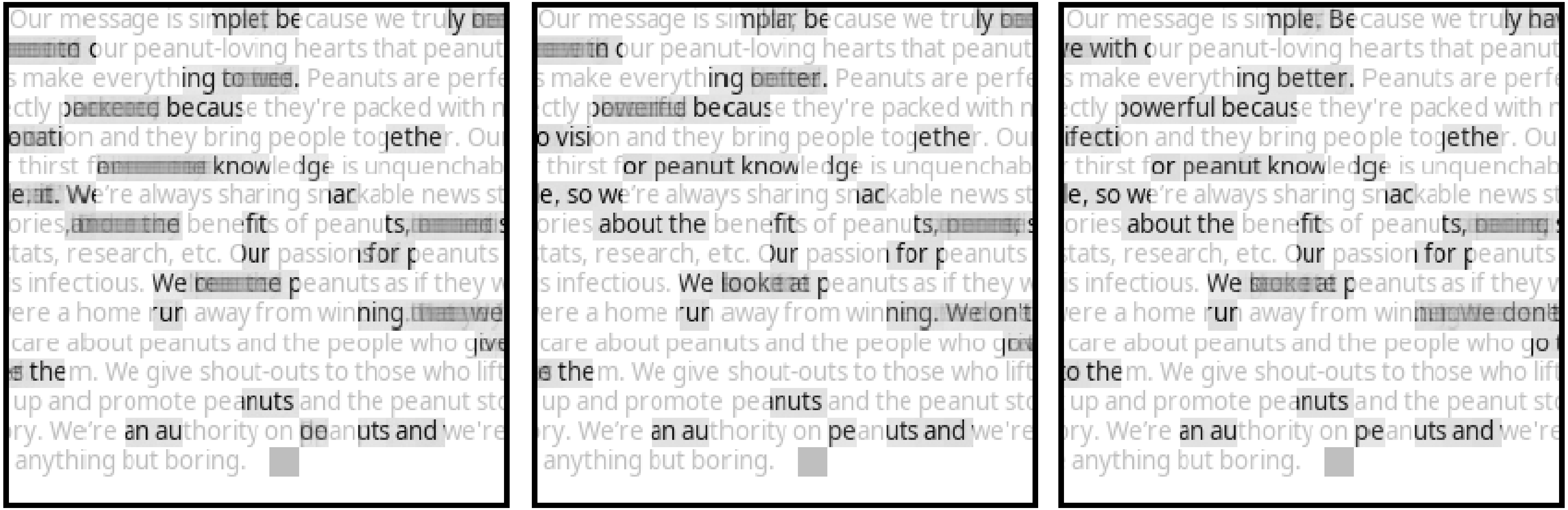
(a) 100k steps (b) 500k steps (c) 1M steps

IMAGE RECONSTRUCTION AFTER PRETRAINING



(a) 100k steps (b) 500k steps (c) 1M steps

IMAGE RECONSTRUCTION AFTER PRETRAINING



(a) 100k steps (b) 500k steps (c) 1M steps

FINETUNING PIXEL

- PIXEL can be finetuned for downstream NLP by replacing the PIXEL decoder with a suitable classification head.
- By truncating or interpolating the sinusoidal position embeddings, we can finetune with sequences shorter or longer than 529 patches, respectively.

FINETUNING PIXEL - WORD CLASSIFICATION

- For word-level tasks like part-of-speech (POS) tagging and named entity recognition (NER), we render each word to the start of a new image patch so that we can create a bijective mapping between words and patches

(c) Word-level rendering (Amharic)

- To finetune PIXEL on these images, we add a linear classifier with dropout. We assign the label of a word only to its first corresponding image patch and compute a cross-entropy loss with SoftMax.

FINETUNING PIXEL - DEPENDENCY PARSING

For dependency parsing, we render text as above but obtain word-level representations by mean pooling over all corresponding image patches of a word

FINETUNING PIXEL - EXTRACTIVE QUESTION ANSWERING

We use a linear classifier to predict the start and end patches of the span containing the answer.

PIXEL - SYNTACTIC TASK

	$\|\theta\|$	ENG	ARA	COP	HIN	JPN	KOR	TAM	VIE	ZHO
POS Tagging (Accuracy)										
BERT	110M	**97.2**	95.4	26.5	86.4	87.9	60.0	45.4	84.5	58.6
PIXEL	86M	96.7	**95.7**	**96.0**	**96.3**	**97.2**	**94.2**	**81.0**	**85.7**	**92.8**
Dependency Parsing (LAS)										
BERT	110M	**90.6**	**77.7**	13.0	75.9	73.8	30.2	15.2	49.4	28.8
PIXEL	86M	88.7	77.3	**83.5**	**89.2**	**90.7**	**78.5**	**52.6**	**50.5**	**73.7**

	[UNK]%	Fertility
ENG	0	1.2
ARA	1.8	3.7
COP	93.6	1.0
HIN	32.6	2.7
JPN	45.5	1.5
KOR	84.7	1.0
TAM	82.3	1.3
VIE	4.5	2.5
ZHO	73.2	1.5

Table 1: Results for PIXEL and BERT finetuned for POS tagging and dependency parsing on various Universal Dependencies treebanks. We report test set results averaged over 5 runs each. $|\theta|$ denotes the number of model parameters. The table on the right shows BERT's proportion of [UNK]s as a measure of (inverse) vocabulary coverage and fertility (i.e., number of subwords per tokenized word; Ács, 2019; Rust et al., 2021) as a measure of over-segmentation in respective UD treebanks.

PIXEL - SEMANTIC TASK

	#L	$\|\theta\|$	TyDiQA-GoldP ENG	ARA	BEN	FIN	IND	KOR	RUS	SWA	TEL	AVG	SQuAD ENG	KorQuAD KOR	JaQuAD JPN
MBERT	104	179M	75.6	78.1	74.7	75.5	84.3	64.8	74.9	83.1	81.6	77.1	88.6	90.0	76.4
BERT	1	110M	**68.5**	**58.0**	**43.2**	**58.3**	**67.1**	12.4	**53.2**	**71.3**	48.2	51.5	**88.2**	14.9	28.8
PIXEL	1	86M	59.6	57.3	36.3	57.1	63.6	**26.1**	50.5	65.9	**61.7**	**52.3**	81.4	**78.0**	**34.1**

Table 4: Results for PIXEL and BERT finetuned on extractive QA datasets. We report validation set F_1 scores averaged over 5 runs each. Average (AVG) scores for TyDiQA-GoldP exclude ENG as customary (Clark et al., 2020). While BERT clearly outperforms PIXEL in ENG, PIXEL is much better in KOR, TEL, and JPN—a consequence of the vocabulary bottleneck in BERT—thereby gaining an edge on average. PIXEL strongly benefits from larger finetuning corpora as seen for KOR (1.6k training examples in TyDi versus 60k in KorQuAD). In some languages, PIXEL's performance suffers due to how we currently extract answer spans from predicted patches (see §3.3 for an explanation), yielding suboptimal text outputs in spite of correct start and end patch predictions.

ROBUSTNESS OF PIXEL

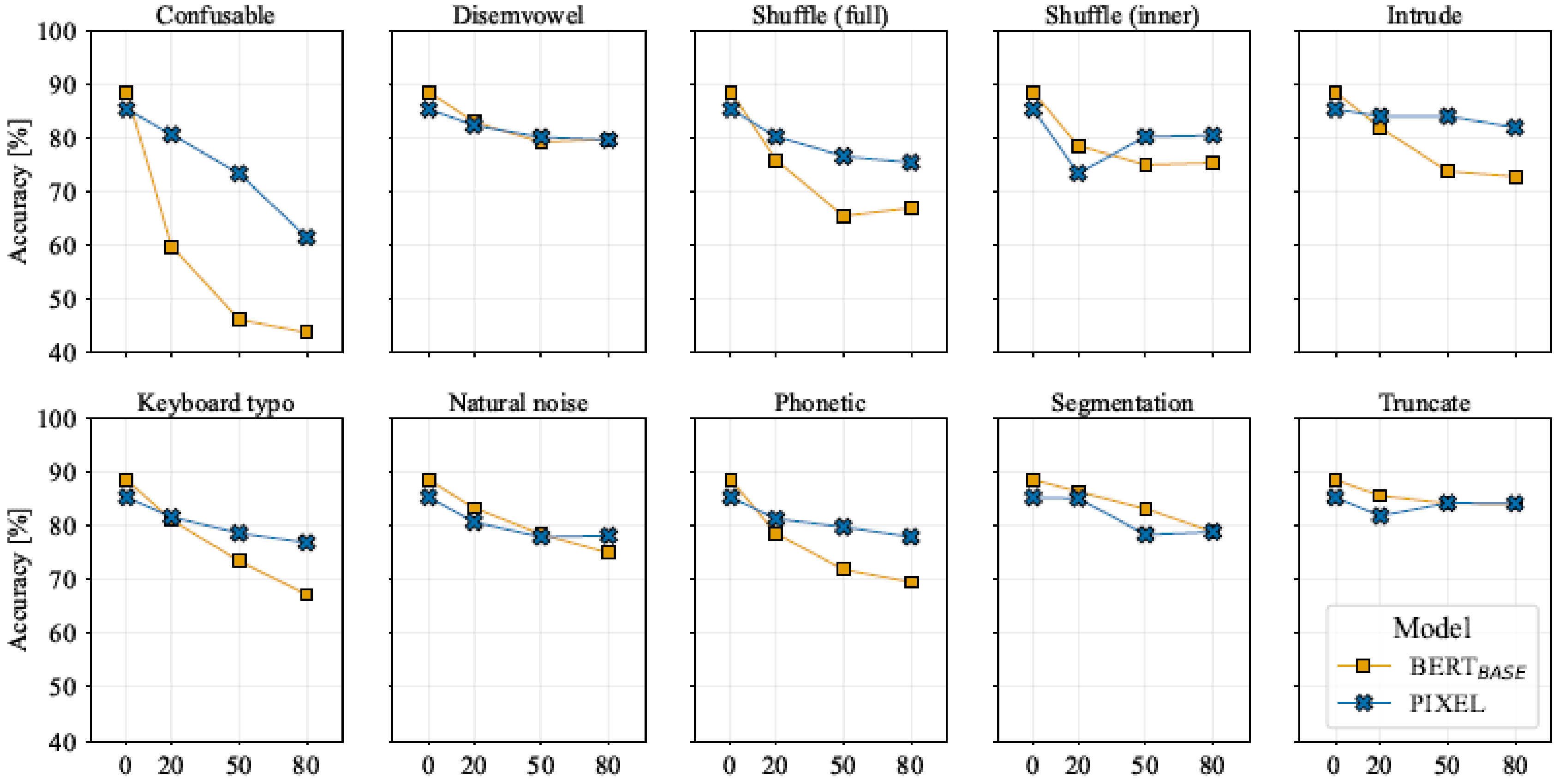
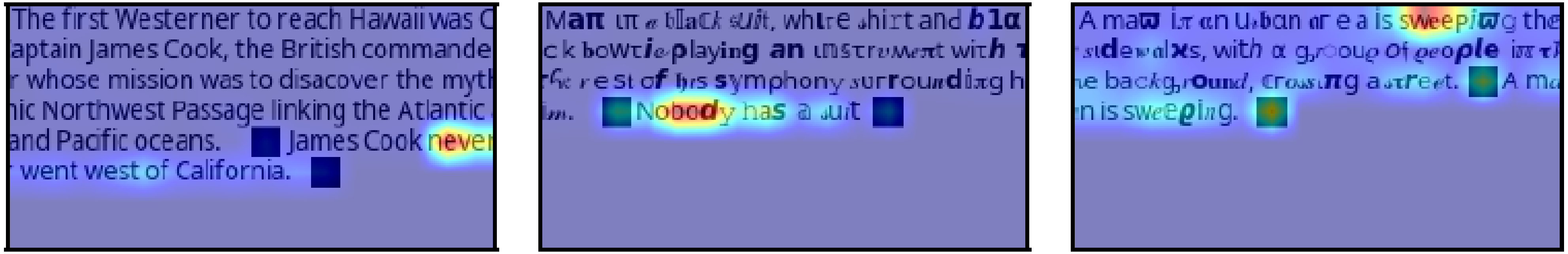


Figure 4: Test set accuracy for a single run of PIXEL and BERT across different levels of noise introduced through various orthographic attacks in SNLI. The results show that PIXEL is more robust than BERT to most of these attacks.

ROBUSTNESS OF PIXEL



(a) 0%, *contradiction* (b) 80%, *contradiction* (c) 80%, *entailment*

Figure 5: Visual explanations of correct predictions (for the classes *contradiction* and *entailment*) made by PIXEL for different NLI examples at 0% and 80% CONFUSABLE character substitutions using method by Chefer et al. (2021), providing qualitative evidence for PIXEL's robustness to character-level noise and the interpretability of its predictions. The images are cropped in half for presentation purposes. Red heatmap regions represent high relevancy.

CONCLUSION

- This paper introduces a new approach to processing written language as images, which removes the need for a finite vocabulary, providing a solution to the vocabulary bottleneck
- PIXEL cannot be used for language generation tasks because it is not possible to produce discrete words from the pretrained decoder

REFERENCES

1) Rust, et al., “Language Modelling with Pixels,” arXiv:2207.06991v1

Thank you