PIXAR: Auto-Regressive Language Modeling in Pixel Space

Anonymous ACL submission

Abstract

Recent work showed the possibility of building open-vocabulary large language models (LLMs) that directly operate on pixel representations. These models are implemented as autoencoders that reconstruct masked patches of rendered text. However, these pixel-based LLMs are limited to discriminative tasks (e.g., classification) and, similar to BERT, cannot be used to *generate text*. Therefore, they cannot be used for generative tasks such as free-form question answering. In this work, we introduce PIXAR, the first pixel-based autoregressive LLM that performs text generation. Consisting of only a decoder, PIXAR can perform free-form generative tasks while keeping the number of parameters on par with previous encoder-decoder models. Furthermore, we highlight the challenges of generating text as non-noisy images and show this is due to using a maximum likelihood objective. To overcome this problem, we propose an adversarial pretraining stage that improves the readability and accuracy of PIXAR by 8.1 on LAMBADA and 8.5 on bAbI— making it comparable to GPT-2 on text generation tasks. This paves the way to build open-vocabulary LLMs that operate on perceptual input only and calls into question the necessity of the usual symbolic input representation, i.e., text as (sub)tokens.

1 Introduction

In natural language processing (NLP) pipelines, tokenizers are an essential ingredient used to divide the raw text into a sequence of sub-units, such as sub-words (Wu et al., 2016), characters (Cui et al., 2020), sentence pieces (Kudo and Richardson, 2018), and bytes (Sennrich et al., 2016). Normally, NLP models represent these sub-units as *symbols* within an ID-based categorical vocabulary.

This categorical representation highlights the following weaknesses of traditional NLP systems. First, to encode each item in the vocabulary, NLP systems have to allocate an embedding matrix that grows linearly with vocabulary size. Modern large language models (LLMs) allocate millions of parameters just for this.[1] Additionally, fixing a vocabulary a priori can lead to performance degradation due to unseen out-of-vocabulary (OOV) words (Kaddour et al., 2023). Tokenizers with smaller granularities, such as characters and bytes, can alleviate the OOV issue but are still brittle as they can suffer from orthographic attacks (Eger et al., 2020). On the other hand, humans are incredibly robust to a variety of text permutations (Rayner et al., 2006) because they leverage the graphical information in text (Sun et al., 2021).

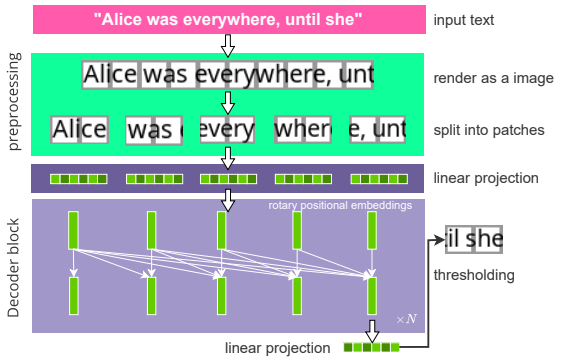

Figure 1: **PIXAR is the first generative language model operating on pixels only.** PIXAR accepts texts as images and also generates texts in image patches autoregressively, a challenging task.

To tackle these problems, Rust et al. (2023) proposed PIXEL, a pixel-based LLM that treats text as images. Pixel-based embeddings remove the need for a finite vocabulary and keep the visual information of text, *questioning whether we need symbolic representations of text as input at all*, or if an LLM can learn symbols implicitly. PIXEL achieved comparable performance with BERT (De-

[1] The embedding matrix of Vicuna-7B (∼130M) is comparable to the size of `bert-base-uncased` (∼109M).

vlin et al., 2019) in a range of downstream classification and regression NLP tasks while being robust to character-level visual attacks (Eger et al., 2020). However, because of its close architectural similarities with BERT, PIXEL cannot deal with free-form generative tasks, such as generative question answering (Lawrence et al., 2019).

To fill this gap, we present PIXAR[2], the first pixel-based autoregressive LLM that can generate short sequences of text as images. PIXAR is to GPT-like architectures as PIXEL is to BERT-like architectures: it consists of a Transformer decoder (Radford et al., 2019) that autoregressively generates text image patches as output. *Generating new text as pixels starting from pixels only* is, however, more challenging than selecting symbolic tokens from a vocabulary (as GPT-like models) or reconstructing masked image patches (as in PIXEL). This is because the model has to learn to generate longer sequences of pixels. To this end, we introduce a two-stage pretraining strategy for PIXAR. First, following previous work on autoregressive LLMs (Radford et al., 2019) and image generation models (Chen et al., 2020a), PIXAR is trained by reconstructing the next patch of pixels derived from a large-scale corpus of rendered text using teacher-forcing. This maximum-likelihood approach, however, can generate image patches containing noisy text. To mitigate this problem, we proposed a second pretraining stage, where PIXAR is trained with an additional adversarial loss.

Our experiments in Section 4 show that 200 steps of stage 2 pretraining improve the readability of generated text significantly and achieve comparable performance with GPT-2 (Radford et al., 2019) on open-answer short generative tasks such as bAbI (Weston et al., 2015) and LAMBADA (Paperno et al., 2016). Additionally, PIXAR achieves better performance than PIXEL on the GLUE benchmark (Wang et al., 2018) while using a computational budget and a number of model parameters equivalent to the encoder part of PIXEL.

2 Beyond token-based LLMs

The idea of using pixel-based representations of text has been applied in various NLP tasks. For instance, Liu et al. (2017) used a CNN-based block to extract character-level visual representations of Chinese writing and similarly, Sun et al. (2018) for text classification. Graphical features of Chinese characters are a good example where canonical symbolic tokenizers miss important information, as highlighted in Meng et al. (2020) exploiting the Tianzige feature of Chinese characters or Dai and Cai (2017) and ChineseBERT (Sun et al., 2021) integrating character-level visual features into embedding vectors for BERT-like models.

Sub-word tokenization as in character-level or byte-level tokens is possible and partially solves some issues.[3] However, these approaches still struggle with a medium that is inherently visual. This is the case if we consider certain writing systems that are (partially) logographic such as in Chinese, or when text contains emojis.

Pixel-based representations mitigate these issues. Salesky et al. (2021) built machine translation models using visual information as input. However, their output layers still rely on embeddings over a fixed vocabulary. Inspired by Salesky et al. (2021), PIXEL (Rust et al., 2023) pretrained a Masked AutoEncoder (MAE) (He et al., 2021) with a large corpus of rendered text using a masked reconstruction objective. This can be considered the first *purely pixel-based LLM* that can be applied to a variety of downstream tasks such as POS tagging and extractive question answering. This approach was extended by Salesky et al. (2023) to deal with multiple languages as well. Moreover, visual text representation can be utilized in multimodal models to build a unified representation for both image and text modalities. Specifically, CLIPPO (Tschannen et al., 2023b) replaces the ID-based text encoder of CLIP (Radford et al., 2021) with a single pixel-based encoder to process both regular images and rendered text for visual QA. Due to the encoder-only architecture of PIXEL and CLIPPO, they still cannot be used to generate text. GlyphDiffusion (Li et al., 2023a) made attempts to use a diffusion model to generate new texts as images from noise and conditioned on encoded text features. However, its encoder still relies on symbolic embeddings. On top of the benefits of purely pixel-based LLMs, we are interested in *understanding if a deep learning model can learn symbolic representation from perceptual information only*, a very challenging task, as discussed next.

3 PIXAR

In contrast with the aforementioned approaches, we design PIXAR as a PIXel-based AutoRegres-

[2]The code is submitted with this submission.

[3]We discuss more related works in Appendix B.

sive LLM whose inputs and outputs are only pixels representing textual information. PIXAR abandons the MAE architecture of PIXEL and instead applies the design choices of other generative LLMs like GPT-2 and LLaMA-2 (Touvron et al., 2023a,b). PIXAR is pretrained to predict the next patches of pixels, conditioning only on the previous patches of rendered text. While this allows PIXAR to tackle free-form generative tasks, it presents several challenges. We start by reviewing its architecture.

3.1 Model Architecture

We design PIXAR as a decoder-only model with a stack of $N = 12$ Transformer layers. Specifically, we extend the Transformer (Vaswani et al., 2017) by applying some of the improvements introduced with LLaMA-2: pre-normalization using RMSNorm (Zhang and Sennrich, 2019), SwiGLU activation functions (Shazeer, 2020), and rotary positional embeddings (Su et al., 2023). Each Transformer layer in PIXAR generates $h^{\text{out}} \in \mathbb{R}^d$ hidden states as output. Further details are in Appendix G.

Similarly to PIXEL, we represent the input text as a single (long) image containing several non-overlapping patches (Figure 1). Following He et al. (2021), each image patch $\boldsymbol{x} \in \mathbb{R}^{H \times W \times C}$—where H and W are the patch height and width and C the number of channels—is then flattened into a vector, and then projected into a hidden embedding $h^{\text{in}} \in \mathbb{R}^d$. The resulting sequence of patch embeddings $\{h^{\text{in}}_1, \ldots, h^{\text{in}}_{\text{eos}}\}$ is used as input to the Transformer backbone. We experiment with both RGB images (i.e., $\boldsymbol{x} \in [0,1]^{H \times W \times 3}$) and binary images ($C = 1$ and $\boldsymbol{x} \in \{0,1\}^{H \times W}$), finding the latter to provide equivalently good downstream performance while simplifying the learning problem (Section 4).

A linear layer after the last Transformer layer projects the embedding h^{out}_N back to pixel space as a vector $\tilde{\boldsymbol{x}}$ representing a linearized $H \times W \times C$ patch (with $C = 1$ for binary images). We interpret $\tilde{\boldsymbol{x}}$ in the following way. For text rendered as binary images, $\tilde{\boldsymbol{x}}$ are the logits that then are squashed by an element-wise sigmoid with temperature T ($T = 1$). To recover a hard binary vector from these probabilities, we apply a threshold θ ($\theta = 0.5$).

When dealing with RGB images, we first clip $\tilde{\boldsymbol{x}}$ element-wise to be within $[0,1]$, then linearly map each channel to {0-255} to construct RGB patches.

3.2 Stage 1 training: MLE

We train PIXAR by maximum likelihood estimation (MLE) by minimizing the negative log-likelihood of L ground truth pixel patches $\boldsymbol{x}_{i:i+L-1}$ conditioned on a sequence of observed (gold) patches $\boldsymbol{x}_{1:i-1}$ which is known as "teacher forcing" (Williams and Zipser, 1989). We assume every pixel in $\boldsymbol{x}_{i:i+L}$ to be conditionally independent given the last layer embedding h^{out}_N. This yields to minimize a reconstruction loss $\mathcal{L}_{\text{rec}}$ over $\boldsymbol{x}_{i:i+L-1}$ that ends up being the usual pixel-wise binary cross-entropy loss for binary images, and the MSE loss for RGB images under our assumptions (Kingma and Welling, 2013; Ghosh et al., 2020).

Figure 2: **A second stage adversarial training can improve the readability of text generated** by PIXAR when compared to noisy patches generated by MLE only. The prompt is from the LAMBADA test set, rendered as a binary image.

This sequential prediction task is quite challenging, as we will need to predict $H \times W \times C \times L$ variables. Having text rendered as binary images simplifies learning, but we observed that PIXAR can easily get stuck in local optima that yield noisy generation: especially for $L > 1$, see Figure 2. This is somehow expected and linked to the tendency of MLE to put probability mass among possible modes (Theis et al., 2015) i.e., readable patch configurations in our case. In Section 3.3 we introduce a way to quantify "readability" of the generated patches, while we discuss next a practical solution to avoid the pitfalls of MLE training.

3.3 Stage 2 training: Adversarial

Our solution to the noisy generated patches is an adversarial loss to be used in conjunction with $\mathcal{L}_{\text{rec}}$ in a second stage training. We name it patch-wise context-aware adversarial (PCAA) loss, and noticed that only 200 steps of minimizing it in our experiments (Section 4.3) can greatly boost the readability and generation performance of PIXAR.

Context-aware adversarial loss. Following previous work on GANs (Goodfellow et al., 2014), an adversarial training regime consists of a generator and a discriminator. We consider PIXAR as our generator. When designing our discriminator, we tried to follow previous work that uses patch-

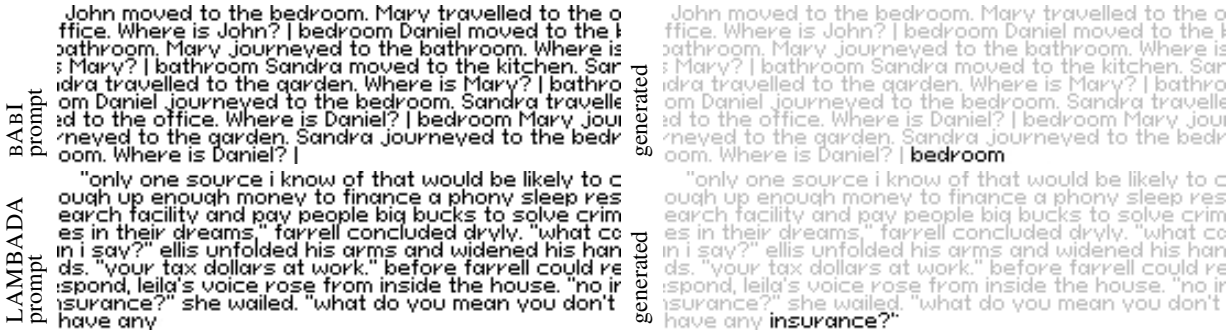

Figure 3: **PIXAR can generate readable and correct texts** according to the prompt (darker) on both bAbI (top) and LAMBADA (bottom). We folded images into rectangles for better visibility.

wise discriminators (Esser et al., 2021; Rombach et al., 2021). However, in preliminary experiments, we find that these solutions lead to catastrophic mode collapse because they only consider the predicted patch as input and ignore previously predicted patches, i.e., the "context".

Our solution is the PCAA loss: $\mathcal{L}_{\text{PCAA}} = \mathbb{E}_{\tilde{\boldsymbol{x}}_{i:i+L-1}}[-\log(D(\tilde{\boldsymbol{x}}_{i:i+L-1}|\boldsymbol{x}_{1:i-1}))]$, which represents how much the generator can "fool" the discriminator a generated patch $\tilde{\boldsymbol{x}}_i$ is real. To compute the PCAA loss, we use a context-aware discriminator implemented by appending a patch-wise classification head to a copy of the pretrained PIXAR (stage 1). Then, for a sequence, the discriminator is trained to decide whether a patch $\boldsymbol{x}_i$ is real or fake (i.e., "readable text" vs "noisy" one) given real previous patches $\boldsymbol{x}_{1:i-1}$.

Due to the computational overhead of the Transformer layers in PIXAR, we developed a patch-sampling algorithm to effectively calculate the PCAA loss. For a sequence of patches, we first generate fake patches and calculate the reconstruction loss using the generator. Then we use the discriminator to calculate key and value vectors for each real patch and calculate the PCAA loss of fake patches. To accelerate the training, we only calculate the PCAA loss of 30 uniformly sampled fake patches for one sequence. We report additional implementation details in the Appendix K.

Balancing MLE and PCAA. To mitigate the notorious instability of training a GAN, we follow Esser et al. (2021) and mix both the MLE loss $\mathcal{L}_{\text{rec}}$ and our PCAA loss $\mathcal{L}_{\text{PCAA}}$. We do so by computing

$$\mathcal{L}_{\text{stage2}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{m}} \cdot \lambda_{\text{auto}} \cdot \mathcal{L}_{\text{PCAA}},$$

where λ_{m} is a tunable hyperparameter, and λ_{auto} is a value automatically computed as $\lambda_{\text{auto}} = \nabla_{G_L}[\mathcal{L}_{\text{rec}}]/(\nabla_{G_L}[\mathcal{L}_{\text{PCAA}}] + \delta)$ where $\nabla_{G_L}[\cdot]$ represents the scale of gradients[4] of the generator w.r.t. its last layer N, and $\delta = 1e^{-8}$ is added to avoid division by zero.

3.4 PIXAR in action

As previously mentioned, PIXAR is able to tackle generative tasks that are out of the scope of PIXEL. At the same time, it can handle the discriminative tasks used in Rust et al. (2023). Depending on the task, we use PIXAR's learned representations in different ways, as discussed next.

Discriminative tasks. Following previous work on downstream discriminative tasks (e.g., Devlin et al. (2019)), we add a lightweight prediction head to PIXAR to make it output classification labels or regression scores. Because of the causal masking strategy of PIXAR (conditioning on patches from left to right), only the last hidden state $h^{\text{out}}_{\text{eos}}$ associated with the EOS patch can attend over all the patches in the input sequence. Therefore, our added head linearly projects $h^{\text{out}}_{\text{eos}}$ to logits then followed by a softmax to output the probability of each class, or used as is for regression tasks.

Generative tasks. PIXAR generates L image patches at a time, containing newly rendered words that complete a given prompt. We can use PIXAR in generative tasks such as language modeling where, given a text passage rendered as a sequence of image patches, the model generates the next word (Paperno et al., 2016). Similarly, we can also perform few-shot learning experiments for QA by providing pairs of examples and solutions as rendered prompts (Weston et al., 2015). We use rendered pipe characters as visual delimiters to separate prompts and generated text (see Figure 3 and

[4]The scale is the mean of the element-wise absolute gradients of the output linear projection layer. Because we accumulate the gradients of a batch from mini-batches, we calculate the scale of every mini-batch and then average them.

Section 4.3). Note that common generative NLP benchmarks expect predictions as (sequences of) symbolic tokens. We discuss next how we can extract text from generated pixel patches, and quantify the readability issue linked to MLE.

Text recognition. We concatenate the generated patches into one image and use OCR software to recognize the text within it. We found, however, that common OCR tools expect higher resolution images to be accurate, and furthermore, they do not work well with binary images, which leads to incorrect recognition even if they are readable by humans. To improve the OCR accuracy, we scale the generated image patches by a factor of 3 and place them in the middle of a square white image background. Also, we combine the results of PaddleOCR[5] and Tesseract OCR[6]. If the recognized word from any of the OCRs matches the target word, we count it as a correct prediction.

Readability metric. As discussed in Section 3.3, some of the generated patches can be noisy and unreadable by both humans and OCR tools. To quantify the quality of the generated text by PIXAR, we define readability as the ratio of generated image patches that contain at least one leading "legal" English word, i.e., a word that could be recognized by the OCR software and that appears in a reference vocabulary built by using 333k most common words from the English Word Frequency dataset[7]. As expected, this metric is correlated to higher prediction accuracy (Table 2).

4 Experiments

We aim to answer the following research questions: **RQ1)** how does PIXAR compare w.r.t., PIXEL on discriminative tasks? **RQ2)** how does it perform as a generative model? **RQ3)** how robust is generated text to orthographic attacks? and **RQ4)** what does PIXAR attend to in a prompt if no symbolic tokens are provided to it as in classical LLMs?

4.1 Experimental setup

Data. PIXAR is trained using the same pretraining dataset as PIXEL, consisting of Bookcorpus (Zhu et al., 2015) and English Wikipedia (see details in Rust et al. (2023)). We follow their preprocessing as described in Appendix C.

Text rendering. We use the same PangoCairo render of PIXEL, but using $H = W = 8$ for the division in patches. For better readability under this low resolution, we use the pixel-style font "Pixeloid Sans".[8] In initial experiments, we found this to be performance-neutral compared with Noto font used by PIXEL. For binary images, we render each pixel to a gray-scale value first and then binarize each pixel value by applying a threshold $\theta = 0.5$. For RGB images, we map each pixel value to the [0, 1] interval. We follow PIXEL's implementation that uses a black patch to represent special EOS tokens, which we use also for delimiting pairs of sentences.

Design choices and ablations. Due to computational constraints, we run preliminary experiments to determine **a)** the number L of patches to predict during training, **b)** whether to use a linear projection layer to map from and to pixel space or perform inference in latent space (Rombach et al., 2022), and **c)** RGB or binary encoding for images. We found that $L = 2$ in binary (linear) pixel space yields the best results and we adopt it for the rest of the experiments. Appendix E details this process.

Stage 1 pretraining. In stage 1, PIXAR is optimized by the AdamW optimizer (Loshchilov and Hutter, 2019) for 1M steps with batch size set to 384. We linearly warmed up the learning rate to 3e-4 in the first 2000 steps and then annealed it to 3e-6 using a cosine scheduler (Loshchilov and Hutter, 2017). For discriminative tasks, we pretrained a PIXAR with 85M parameters, which is equivalent to the size of the encoder of the PIXEL. For generative tasks, we used a PIXAR with 113M parameters instead to have a comparable size with GPT-2. We report details of pretraining and model architecture hyperparameters in Appendix G.

Stage 2. We experimented with different values of λ_{m}. Specifically, we trained the stage 1 model for 200 steps using different λ_{m} from 0.1 to 15, selecting the one with best validation performance.

4.2 RQ1) Discriminative Tasks

We follow Rust et al. (2023) and evaluate the language understanding ability of PIXAR using the GLUE benchmark. GLUE consists of 8 classification tasks and 1 regression task. For each task, we finetune the pretrained PIXAR with a newly initialized prediction head using rendered data. The rendering configuration is the same as the pretraining. In some tasks, an example consists of a pair of sentences. To delimit the two sentences, we inserted a black patch between them. As discussed in

[5]PaddleOCR is available here

[6]Tesseract OCR is available here

[7]The vocabulary file could be downloaded here

[8]The font file is available at the following link.

model	$\|\theta\|$	MNLI-m/mm 392k	QQP 363k	QNLI 108k	SST-2 67k	COLA 8.5k	STSB 5.7k	MRPC 3.5k	RTE 2.5k	WNLI 635	AVG
BERT	110M	84.0/84.2	87.6	91.0	92.6	60.3	88.8	90.2	69.5	51.8	80.0
GPT-2	126M	81.0	89.4	87.7	92.5	77.0	74.9	71.5	52.0	54.9	75.6
PIXEL	86M	78.1/78.9	84.5	**87.8**	**89.6**	38.4	81.1	**88.2**	**60.5**	53.8	74.1
PIXAR (stage 1)	85M	78.4/78.6	85.6	85.7	89.0	**39.9**	81.7	83.3	58.5	59.2	74.0
PIXAR (stage 2)	85M	**79.7/80.1**	**86.3**	85.7	89.3	37.0	**82.4**	82.8	57.7	**60.6**	**74.2**

Table 1: **PIXAR achieves on-par performance with PIXEL on GLUE** with a simpler structure. We report the F1 score for MRPC and QQP, Matthew's correlation for COLA, Spearman's ρ for STSB, and accuracy for other tasks. We report BERT performance here as a reference. Scores of BERT and PIXEL are originally reported in Rust et al. (2023). GPT-2 scores are reported from Huggingface. The best performance of each task is marked in bold.

Section 3.4, we used the embedding from the last black patch as the input of the task head.

We adopt the same hyperparameters for fine-tuning used by Rust et al. (2023). Additionally, during training, we use an early-stopping strategy based on the validation set of each corresponding dataset. For tasks with more than 300k samples, e.g. MNLI and QQP, we set the maximum training step as 8000 and batch size as 256. We found that PIXAR is robust to hyperparameters on tasks with more data and sensitive to batch size or learning rate for smaller datasets (see Appendix H).

As shown in Table 1, PIXAR achieves on-par average performance with PIXEL (74.0 vs 74.1) with a substantially simpler architecture, and scores very closely to GPT-2 (75.6). Note that PIXEL is a 112M encoder-decoder model but only the 86M encoder is used in this experiment and GPT-2 uses even more parameters (126M). We also highlight the stark improvement of 5.4 points on WNLI compared to PIXEL. Considering that WNLI is based on fictional books, the improvement in performance on WNLI might be associated with PIXAR's ability to learn better representations from the pretraining dataset that includes books (i.e., BookCorpus). We also note that stage-2 pretraining slightly increases performance for discriminative tasks. According to our intuition, the adversarial training improves PIXAR's hidden states representations making them more suitable for discriminative tasks.

4.3 RQ2) Generative Tasks

For generative tasks, we render the prompt into an image and insert a white space patch of 3 pixels long to start the generation—intended as a reasonable space to delimit the beginning of a new word. Then PIXAR generates the next image patches autoregressively from there. We use

Model	$\|\theta\|$	LAMBADA	bAbI
GPT-2	124M	17.1	26.8
PIXAR$_{(\text{stage1})}$	113M	5.7 (54.8)	11.1 (63.2)
PIXAR$_{(\text{stage2})}$	113M	13.8 (82.2)	19.6 (77.0)

Table 2: **PIXAR with a 2-stage pretraining is comparable with GPT-2 on short generative tasks.** We report the zero-shot last word prediction accuracy and readability ratio (in brackets) of the LAMBADA test set and the few-shot accuracy of 10K synthesized samples from bAbI.

two datasets for the generative tasks evaluation: **LAMBADA**, a benchmark designed to evaluate the text-understanding capability of LLMs where models have to predict the last word of a sentence in a given context, and **bAbI** a QA task that evaluates the model's reading comprehension ability on some given facts. For bAbI, since PIXAR and GPT-2 are not directly trained on QA data, we show each model 4 examples in the prompt and split the question and answer using the "|" symbol (see Figure 3 and Section 3.4).

In LAMBADA the model needs to capture long-range dependencies in a context that is exposed only to the models during evaluation. From Table 2, we can observe that after only 200 steps of the second stage of pretraining, we can boost the performance of PIXAR and its overall generation readability (Section 3.4) by 8.1 points. Although PIXAR cannot beat GPT-2, it is almost in the same ballpark (13.8 versus 17.1), but it is 27% smaller than GPT-2 and our automatic OCR pipeline might not perfectly recognize some differently rendered words. For bAbI, we confirm the benefit of stage 2 pretraining (as we can also generate 77% of "readable" answers), but PIXAR's final performance is a bit further away from GPT-2 (19.6 vs 26.8), hinting

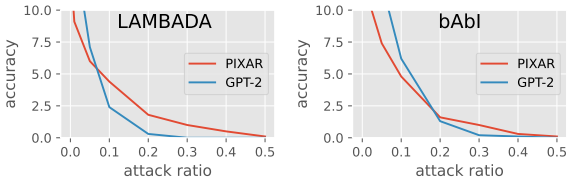

Figure 4: **PIXAR is more robust than GPT-2 under high visual attack ratios**, especially on LAMBADA. We measured zero-shot accuracy on LAMBADA and few-shot accuracy on bAbI.

that it is a more challenging task than LAMBADA. Nevertheless, our results on bAbI provide a promising signal that few-shot learning is also possible with pixel-based generative models such as PIXAR. We complement this experiment by showing more generation examples in Figures 3 and 8 and in our analysis of attention in Section 4.5.

4.4 RQ3) Robustness to orthographic attacks

Following Rust et al. (2023), we use the procedure used in Eger and Benz (2020) to evaluate whether PIXAR can be more robust to "visual attacks" than GPT-2 when performing generative tasks such as LAMBADA and bAbI. First, we manually select a subset of look-alike characters for every English letter from the visually confusable characters of Unicode Technical Standard #39 set [9]. In this way, we make sure that the characters used in the visual attacks can be displayed with the "Pixeloid Sans" font. During the evaluation, we randomly replaced a ratio of letters in a prompt with a random look-alike character and measured the accuracy of the generated word. The results in Figure 4 demonstrate that although both models suffer from the visual attack, PIXAR showcased higher robustness under higher visual attack ratios. We report details of the performance under visual attack in Table 7.

4.5 RQ4) What is PIXAR looking at?

Modeling text purely from perceptual information raises the question of how PIXAR processes such information to generate configurations of pixels that resemble symbolic information (words), some of which might not have been seen during training. This question is vast (and challenging!) and, in this work, we attempt to answer it by looking at which parts of the rendered pixels PIXAR attends to. To do so, we take inspiration from the attention map analysis of Rust et al. (2023), and inspect **RQ4.1)** how attention changes through the Transformers layers, **RQ4.2)**, what PIXAR is looking at when generating one patch at a time, and **RQ4.3)** if there is any correlation with (sub-)words that are semantically meaningful for the generative task at hand. For all the above questions, we examine the attention weights when generating image patches given prompts from the LAMBADA task. For each generated image patch, we calculate the average attention weights across all attention heads of all layers (RQ4.1) or just the last layer (RQ4.2,3). We then plot these averaged weights as heatmaps, which allow us to observe the areas of focus during the generation process. We provide some examples in Figures 5 to 7, and more in Appendix J.

RQ4.1) We visualize the attention weights of all layers when generating the first image patch (Figure 5). A consistent pattern we observe is that the bottom layers (e.g. first and third layers) tend to focus on many parts of the input prompt at once. Layers closer to the final one instead attend more intensely to sub-words, filtering out irrelevant information. This hints at a possible refinement process in PIXAR's attention mechanism as the information propagates throughout layers and focuses on patches representing potentially meaningful words.

RQ4.2) Similarly, looking at attended pixels before and after generating the first patch of a word (Figure 6), suggests that PIXAR has two distinct modes when generating a long word spanning multiple patches, and it distributes attention differently depending on the stage of generation. Specifically, when generating the first patch of a long word, PIXAR tends to focus its attention on previous patches, suggesting a reliance on contextual information to decide which word to generate. Then, after it commits to that word, long contexts become less relevant and PIXAR systematically looks more at patches closer to the generated word. This second mode suggests a tendency of PIXAR to preserve consistency and coherence within the ongoing word generation process.

RQ4.3) Finally, we visualize attention when PIXAR generates the correct or incorrect answer. We notice that when the answer is present in the prompt, PIXAR correctly attends to it with higher importance. This is the case for correctly predicted and highlighted "rope" and "Leo", but also the not predicted but still highlighted "bob" in Figure 7. Even when wrongly predicting "water" (the correct answer is "coffee"), the model attends to "water" and other properties related to it in the prompt.

[9] Confusables characters can be found here.

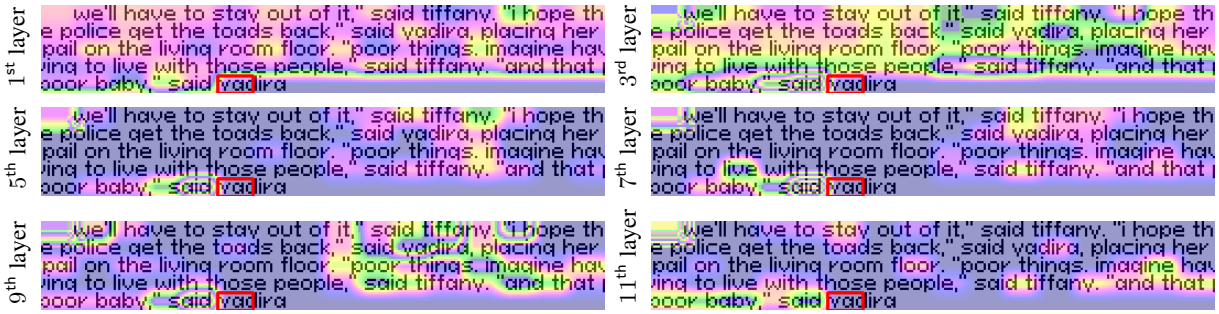

Figure 5: **PIXAR looks at longer patch sequences in the first layers and then focuses on specific word-like sequences** as shown by the above heatmaps of the attention weights for the first generated patch of "yadira". The same pattern is visible in many more examples in Figure 11.

Figure 6: **PIXAR focus on more context patch sequences when generating the first patch of a word, then narrowing its focus**, as shown by the above heatmaps of the attention weights for two consecutively generated patches of the correctly predicted "hazard". Figure 9 shows more examples of the same pattern .

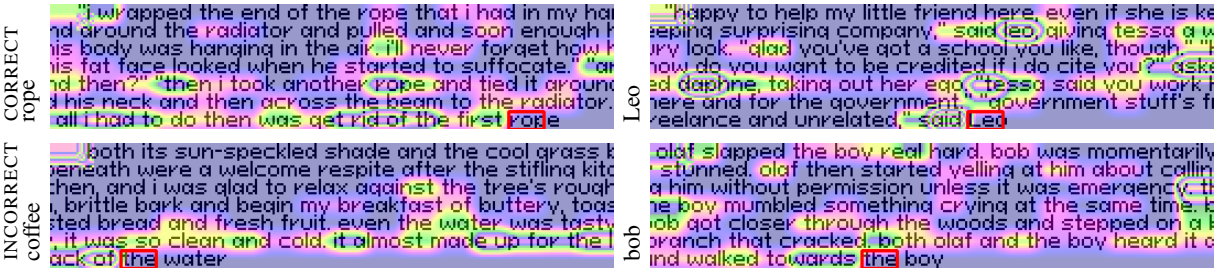

Figure 7: **PIXAR attends to sub-words that can be semantically relevant to generate correct answers** (top), even when the generated answer is wrong (bottom). Correct answers are on the left of each figure. Figure 10 contains more examples.

More examples in Figure 10 showcase similar patterns. While not easy to quantify, this hints at the promising ability of PIXAR to be able to extract, perhaps not correct but still semantically meaningful, symbolic information from perceptual one.

5 Conclusion

We introduced PIXAR, an autoregressive pixel-based LLM that inherits the benefits of the first purely pixel-based LLM PIXEL but, differently from it, can support generative language tasks. We showcased the capabilities of PIXAR on discriminative tasks such as GLUE where it achieves comparable (or superior in some cases) performance w.r.t. PIXEL, and on generative free-text QA tasks, such as LAMBADA and bAbI, where it gets closer to GPT-2 while being more robust to orthographic attacks. While these results are encouraging, PIXAR performance is still behind more sophisticated LLMs that operate on symbolic tokens. This is expected, as reading and generating text as pure pixels is a much more challenging task, and we expect to improve on PIXAR's architecture and performance in future work, e.g., by training larger variants of PIXAR, on more text data rendered as images, and with more language variants. Furthermore, we highlighted how an adversarial loss can greatly improve the quality and accuracy of generated text. Overall, our work shed light on the possibility of treating texts as image-domain data paving the way towards more expressive language models that can efficiently generalise across languages and cultures (Liu et al., 2021).

References

Armen Aghajanyan, Bernie Huang, Candace Ross, Vladimir Karpukhin, Hu Xu, Naman Goyal, Dmytro Okhonko, Mandar Joshi, Gargi Ghosh, Mike Lewis, and Luke Zettlemoyer. 2022. Cm3: A causal masked multimodal model of the internet.

Armen Aghajanyan, Lili Yu, Alexis Conneau, Wei-Ning Hsu, Karen Hambardzumyan, Susan Zhang, Stephen Roller, Naman Goyal, Omer Levy, and Luke Zettlemoyer. 2023. Scaling laws for generative mixed-modal language models.

Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. 2022. Beit: Bert pre-training of image transformers.

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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Huiwen Chang, Han Zhang, Jarred Barber, AJ Maschinot, Jose Lezama, Lu Jiang, Ming-Hsuan Yang, Kevin Murphy, William T. Freeman, Michael Rubinstein, Yuanzhen Li, and Dilip Krishnan. 2023. Muse: Text-to-image generation via masked generative transformers.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. 2022. Maskgit: Masked generative image transformer.

Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. 2020a. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR.

Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. 2020b. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. 2020. Revisiting pre-trained models for chinese natural language processing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics.

Falcon Dai and Zheng Cai. 2017. Glyph-aware embedding of Chinese characters. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 64–69, Copenhagen, Denmark. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Steffen Eger and Yannik Benz. 2020. From hero to zéroe: A benchmark of low-level adversarial attacks. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 786–803, Suzhou, China. Association for Computational Linguistics.

Steffen Eger, Gözde Gül Şahin, Andreas Rücklé, Ji-Ung Lee, Claudia Schulz, Mohsen Mesgar, Krishnkant Swarnkar, Edwin Simpson, and Iryna Gurevych. 2020. Text processing like humans do: Visually attacking and shielding nlp systems.

Patrick Esser, Robin Rombach, and Björn Ommer. 2021. Taming transformers for high-resolution image synthesis.

Partha Ghosh, Mehdi SM Sajjadi, Antonio Vergari, Michael J Black, and Bernhard Schölkopf. 2020. From variational to deterministic autoencoders. In *Eight International Conference on Learning Representations (ICLR 2020)*. OpenReview. net.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2021. Masked autoencoders are scalable vision learners.

Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. 2021. Perceiver: General perception with iterative attention.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

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

Carolin Lawrence, Bhushan Kotnis, and Mathias Niepert. 2019. Attending to future tokens for bidirectional sequence generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1–10, Hong Kong, China. Association for Computational Linguistics.

Junyi Li, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023a. Glyphdiffusion: Text generation as image generation.

Tianhong Li, Huiwen Chang, Shlok Kumar Mishra, Han Zhang, Dina Katabi, and Dilip Krishnan. 2023b. Mage: Masked generative encoder to unify representation learning and image synthesis.

Fangyu Liu, Emanuele Bugliarello, Edoardo Maria Ponti, Siva Reddy, Nigel Collier, and Desmond Elliott. 2021. Visually grounded reasoning across languages and cultures. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10467–10485, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Frederick Liu, Han Lu, Chieh Lo, and Graham Neubig. 2017. Learning character-level compositionality with visual features. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2059–2068, Vancouver, Canada. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2017. Sgdr: Stochastic gradient descent with warm restarts.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Yuxian Meng, Wei Wu, Fei Wang, Xiaoya Li, Ping Nie, Fan Yin, Muyu Li, Qinghong Han, Xiaofei Sun, and Jiwei Li. 2020. Glyce: Glyph-vectors for chinese character representations.

Jacob Menick and Nal Kalchbrenner. 2018. Generating high fidelity images with subscale pixel networks and multidimensional upscaling.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision.

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

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents.

Keith Rayner, Sarah J White, Rebecca L Johnson, and Simon P Liversedge. 2006. Raeding wrods with jubmled lettres: there is a cost. *Psychological science*, 17(3):192–193.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2021. High-resolution image synthesis with latent diffusion models. *CoRR*, abs/2112.10752.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models.

Phillip Rust, Jonas F. Lotz, Emanuele Bugliarello, Elizabeth Salesky, Miryam de Lhoneux, and Desmond Elliott. 2023. Language modelling with pixels.

Elizabeth Salesky, David Etter, and Matt Post. 2021. Robust open-vocabulary translation from visual text representations.

Elizabeth Salesky, Neha Verma, Philipp Koehn, and Matt Post. 2023. Multilingual pixel representations for translation and effective cross-lingual transfer.

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. 2017. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units.

Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Noam Shazeer. 2020. Glu variants improve transformer.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding.

Baohua Sun, Lin Yang, Patrick Dong, Wenhan Zhang, Jason Dong, and Charles Young. 2018. Super characters: A conversion from sentiment classification to image classification. In *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 309–315, Brussels, Belgium. Association for Computational Linguistics.

Zijun Sun, Xiaoya Li, Xiaofei Sun, Yuxian Meng, Xiang Ao, Qing He, Fei Wu, and Jiwei Li. 2021. Chinese-BERT: Chinese pretraining enhanced by glyph and Pinyin information. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2065–2075, Online. Association for Computational Linguistics.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. 2015. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Michael Tschannen, Cian Eastwood, and Fabian Mentzer. 2023a. Givt: Generative infinite-vocabulary transformers.

Michael Tschannen, Basil Mustafa, and Neil Houlsby. 2023b. Clippo: Image-and-language understanding from pixels only.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. 2016. Pixel recurrent neural networks.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2018. Neural discrete representation learning.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. 2024. Mambabyte: Token-free selective state space model. *arXiv preprint arXiv:2401.13660*.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks.

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

BigScience Workshop, :, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, Dragomir Radev, Eduardo González Ponferrada, Efrat Levkovizh, Ethan Kim, Eyal Bar Natan, Francesco De Toni, Gérard Dupont, Germán Kruszewski, Giada Pistilli, Hady Elsahar, Hamza Benyamina, Hieu Tran, Ian Yu, Idris Abdulmumin, Isaac Johnson, Itziar Gonzalez-Dios, Javier de la Rosa, Jenny Chim, Jesse Dodge, Jian Zhu, Jonathan Chang, Jörg Frohberg, Joseph Tobing, Joydeep Bhattacharjee, Khalid Almubarak, Kimbo Chen, Kyle Lo, Leandro Von Werra, Leon Weber, Long Phan, Loubna Ben allal, Ludovic Tanguy, Manan Dey, Manuel Romero Muñoz, Maraim Masoud, María Grandury, Mario Šaško, Max Huang, Maximin Coavoux, Mayank Singh, Mike Tian-Jian Jiang, Minh Chien Vu, Mohammad A. Jauhar, Mustafa Ghaleb, Nishant Subramani, Nora Kassner, Nurulaqilla Khamis, Olivier Nguyen, Omar Espejel, Ona de Gibert, Paulo Villegas, Peter Henderson, Pierre Colombo, Priscilla Amuok, Quentin Lhoest, Rheza Harliman, Rishi Bommasani, Roberto Luis López, Rui Ribeiro, Salomey Osei, Sampo Pyysalo, Sebastian Nagel, Shamik Bose, Shamsuddeen Hassan Muhammad, Shanya Sharma, Shayne Longpre, Somaieh Nikpoor, Stanislav Silberberg, Suhas Pai, Sydney Zink, Tiago Timponi Torrent, Timo Schick, Tristan Thrush, Valentin Danchev,

Vassilina Nikoulina, Veronika Laippala, Violette Lepercq, Vrinda Prabhu, Zaid Alyafeai, Zeerak Talat, Arun Raja, Benjamin Heinzerling, Chenglei Si, Davut Emre Taşar, Elizabeth Salesky, Sabrina J. Mielke, Wilson Y. Lee, Abheesht Sharma, Andrea Santilli, Antoine Chaffin, Arnaud Stiegler, Debajyoti Datta, Eliza Szczechla, Gunjan Chhablani, Han Wang, Harshit Pandey, Hendrik Strobelt, Jason Alan Fries, Jos Rozen, Leo Gao, Lintang Sutawika, M Saiful Bari, Maged S. Al-shaibani, Matteo Manica, Nihal Nayak, Ryan Teehan, Samuel Albanie, Sheng Shen, Srulik Ben-David, Stephen H. Bach, Taewoon Kim, Tali Bers, Thibault Fevry, Trishala Neeraj, Urmish Thakker, Vikas Raunak, Xiangru Tang, Zheng-Xin Yong, Zhiqing Sun, Shaked Brody, Yallow Uri, Hadar Tojarieh, Adam Roberts, Hyung Won Chung, Jaesung Tae, Jason Phang, Ofir Press, Conglong Li, Deepak Narayanan, Hatim Bourfoune, Jared Casper, Jeff Rasley, Max Ryabinin, Mayank Mishra, Minjia Zhang, Mohammad Shoeybi, Myriam Peyrounette, Nicolas Patry, Nouamane Tazi, Omar Sanseviero, Patrick von Platen, Pierre Cornette, Pierre François Lavallée, Rémi Lacroix, Samyam Rajbhandari, Sanchit Gandhi, Shaden Smith, Stéphane Requena, Suraj Patil, Tim Dettmers, Ahmed Baruwa, Amanpreet Singh, Anastasia Cheveleva, Anne-Laure Ligozat, Arjun Subramonian, Aurélie Névéol, Charles Lovering, Dan Garrette, Deepak Tunuguntla, Ehud Reiter, Ekaterina Taktasheva, Ekaterina Voloshina, Eli Bogdanov, Genta Indra Winata, Hailey Schoelkopf, Jan-Christoph Kalo, Jekaterina Novikova, Jessica Zosa Forde, Jordan Clive, Jungo Kasai, Ken Kawamura, Liam Hazan, Marine Carpuat, Miruna Clinciu, Najoung Kim, Newton Cheng, Oleg Serikov, Omer Antverg, Oskar van der Wal, Rui Zhang, Ruochen Zhang, Sebastian Gehrmann, Shachar Mirkin, Shani Pais, Tatiana Shavrina, Thomas Scialom, Tian Yun, Tomasz Limisiewicz, Verena Rieser, Vitaly Protasov, Vladislav Mikhailov, Yada Pruksachatkun, Yonatan Belinkov, Zachary Bamberger, Zdeněk Kasner, Alice Rueda, Amanda Pestana, Amir Feizpour, Ammar Khan, Amy Faranak, Ana Santos, Anthony Hevia, Antigona Unldreaj, Arash Aghagol, Arezoo Abdollahi, Aycha Tammour, Azadeh HajiHosseini, Bahareh Behroozi, Benjamin Ajibade, Bharat Saxena, Carlos Muñoz Ferrandis, Daniel McDuff, Danish Contractor, David Lansky, Davis David, Douwe Kiela, Duong A. Nguyen, Edward Tan, Emi Baylor, Ezinwanne Ozoani, Fatima Mirza, Frankline Ononiwu, Habib Rezanejad, Hessie Jones, Indrani Bhattacharya, Irene Solaiman, Irina Sedenko, Isar Nejadgholi, Jesse Passmore, Josh Seltzer, Julio Bonis Sanz, Livia Dutra, Mairon Samagaio, Maraim Elbadri, Margot Mieskes, Marissa Gerchick, Martha Akinlolu, Michael McKenna, Mike Qiu, Muhammed Ghauri, Mykola Burynok, Nafis Abrar, Nazneen Rajani, Nour Elkott, Nour Fahmy, Olanrewaju Samuel, Ran An, Rasmus Kromann, Ryan Hao, Samira Alizadeh, Sarmad Shubber, Silas Wang, Sourav Roy, Sylvain Viguier, Thanh Le, Tobi Oyebade, Trieu Le, Yoyo Yang, Zach Nguyen, Abhinav Ramesh Kashyap, Alfredo Palasciano, Alison Callahan, Anima Shukla, Antonio Miranda-Escalada, Ayush Singh, Benjamin Beilharz, Bo Wang, Caio Brito, Chenxi Zhou, Chirag Jain, Chuxin Xu, Clémentine Fourrier, Daniel León Periñán, Daniel Molano, Dian Yu, Enrique Manjavacas, Fabio Barth, Florian Fuhrimann, Gabriel Altay, Giyaseddin Bayrak, Gully Burns, Helena U. Vrabec, Imane Bello, Ishani Dash, Jihyun Kang, John Giorgi, Jonas Golde, Jose David Posada, Karthik Rangasai Sivaraman, Lokesh Bulchandani, Lu Liu, Luisa Shinzato, Madeleine Hahn de Bykhovetz, Maiko Takeuchi, Marc Pàmies, Maria A Castillo, Marianna Nezhurina, Mario Sänger, Matthias Samwald, Michael Cullan, Michael Weinberg, Michiel De Wolf, Mina Mihaljcic, Minna Liu, Moritz Freidank, Myungsun Kang, Natasha Seelam, Nathan Dahlberg, Nicholas Michio Broad, Nikolaus Muellner, Pascale Fung, Patrick Haller, Ramya Chandrasekhar, Renata Eisenberg, Robert Martin, Rodrigo Canalli, Rosaline Su, Ruisi Su, Samuel Cahyawijaya, Samuele Garda, Shlok S Deshmukh, Shubhanshu Mishra, Sid Kiblawi, Simon Ott, Sinee Sang-aroonsiri, Srishti Kumar, Stefan Schweter, Sushil Bharati, Tanmay Laud, Théo Gigant, Tomoya Kainuma, Wojciech Kusa, Yanis Labrak, Yash Shailesh Bajaj, Yash Venkatraman, Yifan Xu, Yingxin Xu, Yu Xu, Zhe Tan, Zhongli Xie, Zifan Ye, Mathilde Bras, Younes Belkada, and Thomas Wolf. 2023. Bloom: A 176b-parameter open-access multilingual language model.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mt5: A massively multilingual pre-trained text-to-text transformer.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. 2023. Megabyte: Predicting million-byte sequences with multiscale transformers. *arXiv preprint arXiv:2305.07185*.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization.

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 *The IEEE International Conference on Computer Vision (ICCV)*.

A Limitations & Risks

In this paper, we train and evaluate PIXAR on English language data only to compare with other well-known autoregressive language models like GPT-2. Salesky et al. (2023) showed promising results for encoder-only models applied to multilingual data. Due to the simplicity of PIXAR design, we foresee that it can be pretrained using multilingual data following previous work. In this paper, we explore the training regime for autoregressive pixel-based language models and find that a simple maximum likelihood regime is suboptimal for these models. Although the short stage 2 adversarial pretraining improved the generation readability and accuracy, PIXAR still has limitations in generating long sentences due to the readability issue. The notorious instability of adversarial training also makes the result of stage 2 pretraining hard to optimise even with the help of automatic GAN ratio balancing. Therefore, we believe that future work should explore more stable pertaining methods as an important research direction. Because experiments with PIXAR cover only the English language, we acknowledge that there might be challenges to be solved when tackling different writing systems.

B More related works

B.1 Generative Language Models

Language models learn to predict probability distributions over a sequence of input elements which can have different granularity including words, tokens, characters, or pixels, which is the approach we follow in this work. This task is generally referred to as next token prediction and it represents a crucial problem in natural language processing (Shannon, 1948). The advent of transformers brought about a huge improvement in this field, especially in parallel training and capturing long-range dependencies (Vaswani et al., 2017). Surprisingly, the family of generative pre-trained transformers (Radford et al., 2019; Brown et al., 2020; Workshop et al., 2023; Touvron et al., 2023a,b; Penedo et al., 2023) demonstrated their strong ability to understand and generate natural language.

B.2 Transformer-based Image Generative Models

The task of image generation can be formulated as pixel-based autoregressive modeling. In the literature, this task was attempted using different architectures such as CNNs (Salimans et al., 2017), RNNs (van den Oord et al., 2016) and Transformers (Chen et al., 2020b; Menick and Kalchbrenner, 2018). However, the method of generating only 1 pixel at each step is limited to low-resolution images but not high-resolution images due to long sequence lengths and poor fidelity. Also, the sequential generative modeling and teacher-forcing training of generative transformers are not suitable for flattened image vectors since they learn to predict the probability distribution over a fixed finite vocabulary instead of image patches.

To mitigate this issue, a two-stage approach is proposed to learn a Vector-Quantized Variational Autoencoder(VQ-VAE) (van den Oord et al., 2018) to map continuous pixel values into a sequence of discrete tokens and then use a transformer decoder to model the distribution of the latent image tokens. This two-stage image tokenization approach not only contributes to generative modeling for high-fidelity images (Chang et al., 2023, 2022; Ramesh et al., 2022; Tschannen et al., 2023a) but also shed the way for interleaved multimodal generation by simply concatenating the vocabularies for both text and images (Aghajanyan et al., 2022, 2023). Learning from the discrete latent space of VQ-VAE becomes a popular way for models to improve image generation quality and enable language-inspired self-supervised learning (Bao et al., 2022; Li et al., 2023b).

B.3 Byte-level tokenizers

The idea of using character-level visual features is not limited to pixel-based models. For instance, MacBERT (Cui et al., 2020) and Jozefowicz et al. (2016) also used character-level encodings. However, they still relied on traditional ID-based embeddings and thus suffered from the vocabulary bottleneck problem.

Alternatively, byte-level tokenization is also capable of encoding any code sequence. Xue et al. (2021) trained mT5 on UTF-8 byte sequence to adapt to different languages. Perceiver (Jaegle et al., 2021) proposed an iterative transformer structure which attended information directly through raw byte arrays. The increased sequence length poses challenges for Transformer-based architectures. For this reason, MEGABYTE (Yu et al., 2023) proposes a hierarchical architecture where a low-level patch decoder is conditioned on a global-level learned patch representations. Another line of research uses recent state space models to extrapolate over extremely long sequence lengths. This is

the case for Mamba byte (Wang et al., 2024) which pretrains a Mamba model (Gu and Dao, 2023) using byte representations.

C Data Preprocessing

Dataset	#samples	avg. #characters
English Wikipedia	6,458,670	3,028
Bookcorpus	17,868	370,756
processed	26,759,562	987

Table 3: Pretraining dataset statistics.

Because the length of Wikipedia articles and books is usually longer than PIXAR's context window. We segment each article or book into several samples using the algorithm 1. In particular, we use the "PunktSentenceTokenizer" from the Natural Language Toolkit (NLTK) (Bird et al., 2009) to segment each article or book into sentences. Then, we concatenate short sentences to training samples within a character limit. Finally, we filter out samples with less than 100 characters. Table 3 demonstrates the length statistics of processed data.

During the pretraining, due to computational constraints, we fix the window size of PIXAR as 360 patches. Thus, the model trained with a longer patch length can process more texts in each forward pass. We manually selected 1180 as the character limit for patch length 2.

D Computational budget

The PIXAR's pretraining stage was completed in ∼4 days using 16 NVIDIA V100 GPUs on an HPC computing cluster by extending the codebase originally released by Rust et al. (2023). Finetuning experiments are shorter and use the same computing infrastructure.

E Hyperparameter Ablations

In our preliminary experiments, we explored several design choices for a PIXAR with 113M parameters, including different patch lengths, CNN-based projection layers, and the different formats of data. Because of our limited computational budget, we experimented with a selected number of variants that could allow us to make informed decisions about our model design. For each setting, we conducted 0.1M stage 1 pretraining steps and evaluated their performance on the GLUE benchmark (Wang et al., 2018).

Patch Length Selection

In this ablation, the height of image patches is fixed at 8 pixels and we render all texts as a long image with 8 pixels high. Thus, the length of each patch defines how many characters or words a patch can hold and also determines the complexity of predicting the next patch.

We pretrained models on images with patch length [2, 5] for 0.1M stage 1 steps. The results in Table 5 show that in all combinations, patch length 2 achieved the highest average score. Thus, we used patch length 2 as our reference value for all the experiments.

Linear Projection vs CNN

We experimented with two different strategies to convert image patches to vectors: 1) linear projection of the patches; and 2) a CNN auto-encoder as the input and output layers (Rombach et al., 2022). We first trained a lightweight CNN auto-encoder with a bottleneck layer of dimension 8 on the pretraining data. Then we take the encoder part as the model input layer, and the decoder as the output layer. During the pretraining, the PIXAR backbone is trained in the latent space of the auto-encoder (Rombach et al., 2022). We freeze parameters from the auto-encoder and train PIXAR to minimize the MSE of the next latent vector rather than the next image patch. However, as results shown in Table 5, the extra auto-encoder did not bring significant improvement compared with the simple linear projection layer. Therefore, we opted for the simpler linear projection instead.

RGB vs binary images

As detailed in Section 3, we explored training with both continuous RGB images and binary images. When we use linear projection layers, we flatten the values of each channel of the RGB image into a vector as the input and calculate the MSE between the predicted vector and the ground truth vector as the loss. When using the CNN auto-encoder as input and output layers, RGB image patches are converted to latent vectors with CNN layers with corresponding layers.

The results in Table 5 indicate that models trained with continuous RGB images have ∼1.4 points lower than their binary counterparts. We therefore used binary rendering in further pretraining experiments.

Algorithm 1 Text Segmentation

Input: text s, maximum length l_{max}, minimum length l_{min}
Initialize $sentList = SentTokenize(s)$, $sampleList = EmptyList()$, $sample = EmptyString()$
for $i = 1$ **to** $||sentList||$ **do**
  if $len(sample) + len(sentList_i) > l_{max}$ **then**
    if $len(sample) \geq l_{min}$ **then**
      $Append(sampleList, sample)$
    end if
    $sample = sentList_i$
  else
    $sample = Concat(sample, sentList_i)$
  end if
end for
if $len(sample) \geq l_{min}$ **then**
  $Append(sampleList, sample)$
end if
Output: $sampleList$

F Licensing

As mentioned before, the starting point for our work was the PIXEL codebase released on GitHub under Apache 2.0 license. We also used the same pretraining datasets that was released on Huggingface. Upon acceptance, we will release our codebase on GitHub following the same approach and license.

G Pretraining Details & Model Configurations

Table 4 demonstrates the stage 1 pretraining hyperparameters, render configuration, and model architecture details of PIXAR.

H GLUE Finetuning Details

Table 6 displays the hyperparameter details of GLUE finetuning.

I Generation Samples

Figure 8 shows more generation samples from LAMBADA and bAbI datasets. Images are visualized using a 0.5 threshold and reshaped to a more square size for a better representation.

J Attention Heatmap Samples

Figure 9, Figure 10 and Figure 11 show more attention heatmaps from LAMBADA datasets. The first generated image patches are marked by red rectangles. Samples are drawn using the attention weights from the last transformer layer of the first generated image patch to the given prompt.

K Adversarial Training

PIXAR is trained using a second stage of pretraining using an adversarial training regime that resembles the one used in GANs (Goodfellow et al., 2014). Figure 12 exemplifies how PIXAR is used for the stage 2 pretraining. Each adversarial step can be divided into 3 stages:

1) Calculate the Reconstruction Loss

we fist sample a batch of sequences $\boldsymbol{x}$ from the pretraining dataset and generate the fake sequences $\tilde{\boldsymbol{x}}$ using the generator. Then we calculate the reconstruction loss $\mathcal{L}_{\text{rec}}$ as well as generator's gradients w.r.t. $\mathcal{L}_{\text{rec}}$. Because we need to reuse the computational graph later for the PCAA loss, we thus set "retain_graph = True" while doing backward propagation using Pytorch. We record the current gradients of the last generator layer as $\boldsymbol{g}_1$ and calculate the scale of it $\nabla_{G_L}[\mathcal{L}_{\text{rec}}] = |\boldsymbol{g}_1|$.

2) Calculate the balanced PCAA and update the generator

First, we calculate key and value vectors for every real patch using the discriminator and cache them for the PCAA calculation. To accelerate the calculation, we randomly sample 30 fake patches for each generated sequence and calculate the PCAA loss with the cached key and value vectors using the discriminator. Then we multiply the PCAA loss with λ_{m} and λ_{auto} which is calculated from the previous step, and backpropa-

Pretrain Hyperparameters		Render Configuration		Model Structure	
peak lr	3e-4	patch length	2	#layers	12
lr scheduler	CosineAnnealing	#patches	360	#attention heads	12
min. lr	3e-5	max #char.	1180	hidden size	768
optimizer	AdamW	min. #char.	100	activation	SwiGLU
β_1	0.9	patch height	8	intermediate size	2048 / 3072
β_2	0.95	render DPI	80	#parameters	85M / 113M
weight decay	0.1	font size	8		
steps	1M	font	PixeloidSans		
warm up	2000	binary	true		
batch size	384	Temperature (T)	1		
precision	fp16 & fp32	Threshold (θ)	0.5		
random seed	42				

Table 4: PIXAR pretrain configuration.

Embedding	CNNAutoencoder				Linear projection			
Image type	Binary	Binary	RGB	RGB	RGB	Binary	Binary	RGB
Patch length L	5	2	2	5	2	2	5	5
MNLI-m/mm	72.5/74.1	75.8/76.2	75.8/76.4	72.4/73.4	74.9/76.3	**76.4/77.6**	72.5/74.0	72.0/73.1
QQP	81.2	**87.8**	83.4	81.8	83.5	84.3	82.1	82.3
QNLI	82.4	**84.3**	84.1	81.7	83.5	84.2	82.8	81.6
SST-2	83.3	86.2	86.2	84.2	86.6	**88.0**	82.6	83.3
COLA	15.2	**30.5**	26.6	13.2	30.4	27.7	16.7	10.8
STSB	69.2	74.2	74.7	68.9	73.5	**81.2**	71.5	75.2
MRPC	81.2	83.4	82.9	82.4	81.8	82.5	82.7	**84.2**
RTE	57.0	59.6	56.3	61.4	54.9	58.5	**61.7**	58.1
WNLI	**57.7**	56.3	**57.7**	56.3	56.3	56.3	56.3	56.3
AVG	67.4	71.4	70.4	67.6	70.2	**71.6**	68.3	67.7

Table 5: **PIXAR trained with binary data, linear projection, and patch length 2 achieved the best GLUE performance** in our ablation tests where we compare variants pretrained and finetuned (only stage 1) with 0.1M steps and 113M parameters with identical hyperparameter selected heuristically as described in Section 4.2 We therefore select this best setting and use it for the full 1M step pretraining.

gate all the way to the generator. Because Pytorch accumulates gradients of multiple backwards steps, to attain the gradients of the last generator layer w.r.t. the PCAA loss, we subtract the previously recorded gradients $\boldsymbol{g}_1$ and calculate the scale $\nabla_{G_L}[\mathcal{L}_{\text{PCAA}}] = |\boldsymbol{g}_2 - \boldsymbol{g}_1|/\lambda_{\text{m}}/\lambda_{\text{auto}}$. Because the PCAA is balanced, we re-scale it to get the true gradients scale. The scale is recorded for the next batch. During the first step, because we initialize λ_{auto} as 1. Now gradients w.r.t. PCAA and reconstruction loss are ready and we can update the parameters of the generator using the specified optimizer.

3) Update the discriminator

Finally, we calculate the classification loss using the cross entropy loss of the discriminator and update its parameters. The procedure is similar to the PCAA calculation, we calculate the loss on real patches and cache key and value vectors, then calculate the loss on 30 randomly sampled fake patches for each sequence.

L Performance under Visual Attack

Table 7 records the accuracy details of GPT-2 and LAMBADA under different visual attack ratios. While evaluating the accuracy, we randomly select a ratio of letters from the prompt and replace them with random look-alike symbols. If the first word the model predicts matches the target word regardless of its casing, we count it as a correct prediction.

PIXAR (stage 1)	MNLI	QQP	QNLI	SST-2	COLA	STSB	MRPC	RTE	WNLI
lr	3e-5	3e-5	3e-5	3e-5	3e-5	3e-5	6e-5	3e-5	3e-5
Optimizer	AdamW								
β_1	0.9								
β_2	0.95								
weight decay	0.1	0.1	0.1	0.01	0.01	0.01	0.01	0.01	0.01
warmup	Linear Warmup								
warmup steps	1000	1000	500	200	50	100	20	50	2
max steps	8000	8000	4000	2000	500	2000	500	500	20
batch size	256	256	256	256	256	32	64	32	128
evaluation freq.	500	500	200	200	100	100	50	50	$\sim$ 1 epoch
random seed	42								
PIXAR (stage 2)	MNLI	QQP	QNLI	SST-2	COLA	STSB	MRPC	RTE	WNLI
lr	3e-5	3e-5	3e-5	3e-5	3e-5	3e-5	3e-5	6e-5	3e-5
Optimizer	AdamW								
β_1	0.9								
β_2	0.95								
weight decay	0.1	0.1	0.1	0.01	0.01	0.01	0.01	0.01	0.01
warmup	Linear Warmup								
warmup steps	1000	1000	500	200	50	100	20	50	2
max steps	8000	8000	4000	2000	500	2000	500	500	20
batch size	256	256	256	256	256	128	128	128	128
evaluation freq.	500	500	200	200	100	100	50	50	$\sim$ 1 epoch
random seed	42								

Table 6: Hyperparameters we used to finetune PIXAR on the GLUE benchmark.

attack	LAMBADA		bAbI	
ratio	GPT-2	PIXAR	GPT-2	PIXAR
0.0	17.1	13.8	26.8	19.6
0.01	15.0	9.1	21.6	11.4
0.05	7.1	6.0	12.1	7.4
0.1	2.4	4.4	6.2	4.8
0.2	0.3	1.8	1.3	1.6
0.3	0.0	1.0	0.2	1.0
0.4	0.0	0.5	0.1	0.3
0.5	0.0	0.1	0.0	0.1

Table 7: Accuracy on the LAMBADA and bAbI benchmarks that we used to assess the robustness of PIXAR to visual attacks generated following the procedure proposed by Eger and Benz (2020).

Figure 8: PIXAR generation samples from LAMBADA and bAbI dataset. Images are folded into rectangles for better readability.

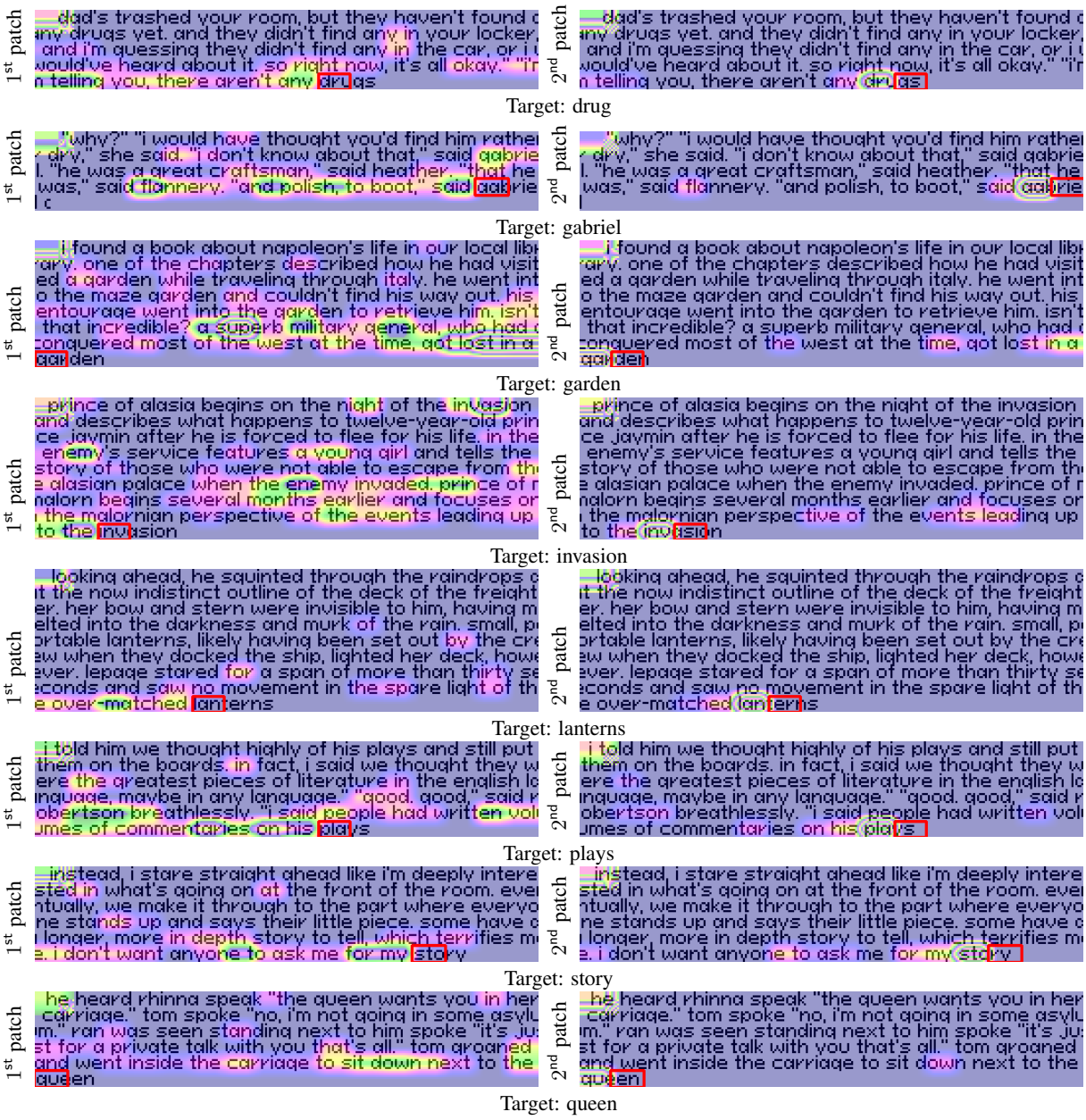

Figure 9: Heatmaps of the attention weights of consecutively generated image patches. The attention weights are collected from the last transformer layer of PIXAR. The corresponding image patches of the sub-figure are mark by the red rectangles.

Figure 10: Heatmaps of the attention weights of PIXAR in correct and incorrect generations. The attention weights are collected from the last transformer layer of the first generated patches, as marked by the red rectangles.

Figure 11: Heatmaps of the attention weights from the selected transformer layers of first generated image patches, as marked by the red rectangles.

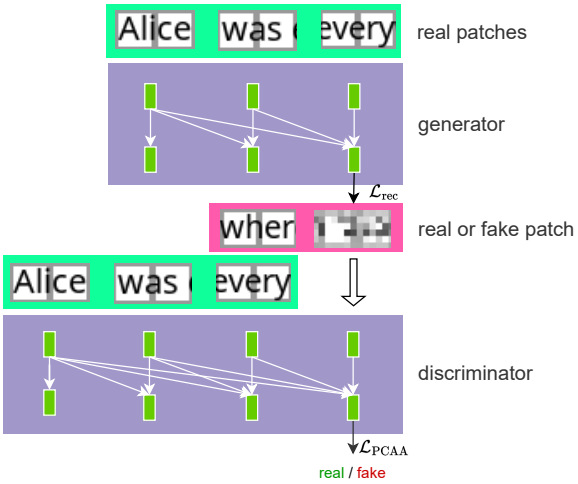

Figure 12: **We copied the PIXAR from stage 1 pretraining as both the generator and the discriminator** for the adversarial training in stage 2. The generator is optimized by both reconstruction loss $\mathcal{L}_{\text{rec}}$ and adversarial loss $\mathcal{L}_{\text{PCAA}}$ to improve the generation readability.