Improving Language Understanding from Screenshots

Anonymous ACL submission

Abstract

An emerging family of language models (LMs), capable of processing both text and images within a single visual view, has the promise to unlock complex tasks such as chart understanding and UI navigation. We name them as *screenshot language models*. Despite their appeal, existing screenshot LMs substantially lag behind text-only models on language understanding. To close this gap, we focus on a simplified setting where screenshots are rendered from plain text. We propose a novel **P**atch-and-**T**ext **P**rediction (**PTP**) objective where we mask and recover both image patches of screenshots and text within screenshots. We also conduct careful ablation studies in masking rates, patch sizes, and designs for training stability. Our pre-trained model, while solely taking visual inputs, achieves comparable performance with BERT (within 2%) on 6 out of 8 GLUE tasks and improves up to 8% on specific datasets over prior work. Additionally, we extend PTP to train autoregressive screenshot LMs and demonstrate its effectiveness—our models can significantly reduce perplexity by utilizing the screenshot context. Together, we hope our findings can inspire future research on developing powerful screenshot LMs, and extending their reach to broader applications.

1 Introduction

The capability for language models (LMs) to process both text and images in a single visual input will enable a broad range of complex applications, such as document understanding (Mathew et al., 2021), chart reading (Masry et al., 2022), and UI navigation (Liu et al., 2018). Conventional methods like adopting an off-the-shelf OCR tool (Huang et al., 2022) is prone to error propagation (Kim et al., 2022); the multimodal approach that processes images and text separately (Alayrac et al., 2022; Liu et al., 2023b, *inter alia*) loses the spacial information between different elements. Regardless, even state-of-the-art LMs such as GPT-4V (OpenAI, 2023) and Gemini (Gemini Team, 2023) struggle at recognizing and understanding text within images (Qi et al., 2023), making them unsuitable for the *everything-in-image* paradigm.

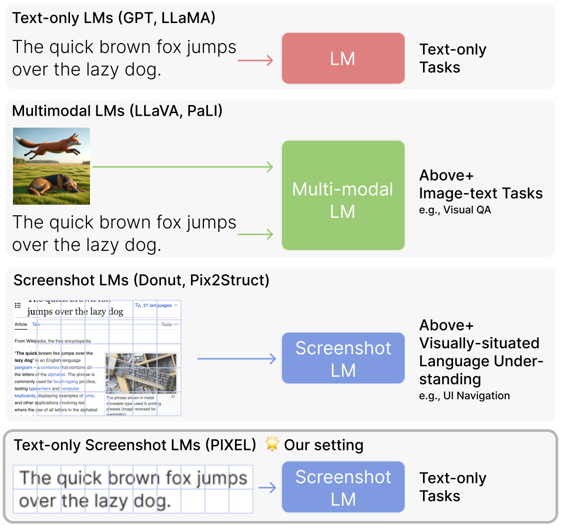

Figure 1: Illustrations on different LM paradigms and their applications. We focus on improving the text understanding ability of screenshot LMs, and adopt a text-only screenshot setting for a clear comparison.

Screenshot LMs. A new family of models (Lee et al., 2023; Rust et al., 2023) has emerged that processes text—along with images, tables, etc.—all through one visual input. We refer to them as *screenshot* LMs. They can be trained and deployed over a vast array of "screenshots"—any images that have a heavy presence of text, such as webpage screenshots, UI images, and document scans. They are designed to handle visually-situated text in an end-to-end manner, and hold the potential to reach broader applications, as illustrated in Figure 1.

Challenges in understanding text from screenshots. Recent development in screenshot LMs has shown promising results in specific scenarios, such as PIXEL (Rust et al., 2023) in multilingual trans-

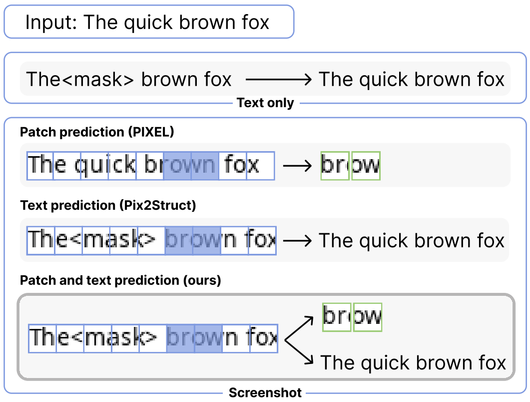

Figure 2: A comparison between existing objectives and our PTP objective for training screenshot LMs. The blue grids illustrate how input images are split into patches.

fer, PhD (Borenstein et al., 2023) in historical document understanding, and Pix2Struct (Lee et al., 2023) in chart and UI understanding. However, the modality mismatch makes it challenging for screenshot LMs to effectively process the text in the inputs, and they exhibit a noticeable deficiency in language understanding tasks when compared to text-only LMs: the prior state-of-the-art, PIXEL, still has a 7% performance gap on GLUE (Wang et al., 2019) when compared to BERT (Devlin et al., 2019). This disparity significantly restricts the utility of screenshot LMs for widespread applications. We argue that to integrate screenshot LMs in practical, real-world scenarios effectively, it is crucial to *first* close this gap on text-only tasks.

Our setting. To enhance the language understanding capability of screenshot LMs, we focus on a *text-only* screenshot setting, where inputs consist exclusively of plain text rendered as images. This particular setting facilitates direct comparison with text-only LMs, enabling us to isolate the impact of the quality of the screenshot data and concentrate on architecture and training objective changes to improve language understanding performance.

Our contributions. (1) We introduce the **P**atch-and-**T**ext **P**rediction (PTP) objective. As shown in Figure 2, previous works either only predict image patches or only predict text. Instead, we mask and predict both the screenshot patches and text within the screenshot. The choice is backed up by the intuition that a patch prediction objective helps learn local visual features of the text, while a text prediction objective is more effective in learning to understand the language.

(2) We find that screenshot LMs often exhibit training instability and are sensitive to hyperparameter choices. We conduct careful ablations on masking rates and patch sizes, as well as exploring designs to stabilize the training of screenshot LMs.

Our pre-trained screenshot LM demonstrates strong language understanding capabilities: it achieves comparable performance (within 2%) to $\text{BERT}_{\text{base}}$ (Devlin et al., 2019) on 6 out of 8 datasets from GLUE (Wang et al., 2019), improving over previous best by up to 8 points on specific tasks.

(3) We also extend our objective to autoregressive LMs, by feeding image patches and text tokens to a single decoder to predict both modalities in an autoregressive manner. We show that the autoregressive screenshot LMs can effectively utilize the screenshot context either by training from scratch or fine-tuning from existing text-only LMs.

We hope our findings will inspire future research exploring screenshot LMs and their applications. We also discuss the limitations of current screenshot LMs, including training instability and inefficiency, as well as possible future directions.

2 Problem Setup

In this section, we define our *text-only screenshot LM* setup. For pre-training, we assume a text corpus $\mathcal{C}$ and we render each text sequence $s \in \mathcal{C}$ as an image I_s. The model is allowed to utilize both s and I_s for its pre-training. For evaluation on the downstream dataset $\mathcal{D}=\{(x_i, y_i)\}, 1 \le i \le |\mathcal{D}|$, we render each input sequence x as I_x and define $\mathcal{D}_I = \{(I_x, y)\}, (x, y) \in \mathcal{D}$. Then we only fine-tune and test the model on $\mathcal{D}_I$. In other words, the model can leverage both the ground truth text and the rendered images for pre-training, but can only take the rendered images as input for downstream tasks. This is similar to a realistic end-to-end screenshot LM scenario, where the inputs are predominately screenshots at inference time.

3 PTP: Patch and Text Prediction

We introduce our screenshot LMs and our training objective PTP. All the components of our model are Transformers (Vaswani et al., 2017) or Vision Transformers (ViT; Dosovitskiy et al., 2021) and the architecture details can be found in Appendix B. In the following, we first introduce how the input images are processed, and then we describe the rendering strategy and the training objectives.

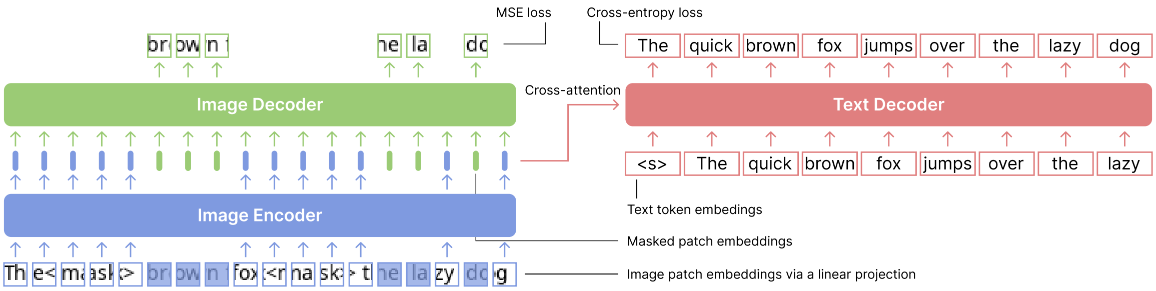

Figure 3: An illustration of our **P**atch-and-**T**ext-**P**rediction (PTP) objective. PTP applies both image patch masking (the dark blue masks) and text masking (`<mask>` tokens) to the input. Subsequently, an image decoder is used to reconstruct the masked patches, and a text decoder is used to recover the corrupted text. The illustration does not reflect all implementation details (e.g., the CLS token). Appendix C provides more details.

3.1 Input Processing

The input images (screenshots, in our case) are split into patches sized $p_h \times p_w$, where h and w denote height and width, respectively. By default, we use $p_h = p_w = 16$, a setting commonly adopted in ViT architectures (Dosovitskiy et al., 2021). Each patch in a sequence of n patches can be seen as a vector input $x_i \in \mathbb{R}^{p_h \times p_w \times c}$, where $i \in \{1, ..., n\}$ and c is the number of channels (by default $c = 3$). These patches are turned into patch embeddings via a linear layer, which are then fed as input features to the transformer. ViTs use a fixed sine/cosine 2D positional embedding (Vaswani et al., 2017).

For the autoregressive text decoder, we use the same tokenization as RoBERTa (Liu et al., 2019) and OPT (Zhang et al., 2022).

3.2 Rendering Screenshots

We follow PIXEL (Rust et al., 2023) for its rendering strategy: we render the text into an image of size $p_h \times np_w$, where n is the number of patches. We replace all the new-line characters with a special symbol and render the text into one line. We use the Google `Noto Sans` fonts collection[1].

We implement our own rendering engine, described in Appendix A. By default, we use a font size of 10px, similar to the one used in PIXEL. On average, one 16×16 patch can fit 0.57 OPT text token—this means that to encode the same amount of text, a screenshot LM following the above rendering strategy will need almost twice the number of tokens compared to a text-only LM.

3.3 Training Objectives

We adopt two training objectives: a masked patch prediction objective and a masked text prediction objective. While either objective has been explored

[1] `https://fonts.google.com/noto`

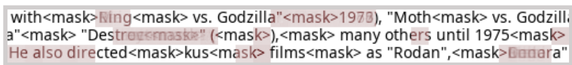

Figure 4: An example of the rendered text, the patch masks (red background; 25% span for images), and the prediction. The image is one line of text but is cut and concatenated for better visualization here.

in prior work, we are the first to combine them and we show that it is critical to leverage both objectives for a competitive performance of screenshot LMs. The intuition behind combining the two objectives is that patch prediction helps learn the local visual features while text prediction is more effective for language understanding. Figure 3 provides an overview of the training.

Patch masking and prediction. We randomly exclude input patches from the image encoder and use a bidirectional image decoder to recover them. An MSE loss is calculated between the predicted and the target pixel values of the masked patches. This paradigm was first proposed in MAE (He et al., 2022) for images and later adopted in PIXEL (Rust et al., 2023) for text-rendered screenshots.

In addition, we follow Rust et al. (2023) and leverage their span masking strategy. As shown in Figure 4, a single patch mask often leaves out shallow cues where the model can complete the word without learning the semantics. In our preliminary experiments, we verified those arguments and hence followed its setting.

We adopt a masking rate of 10%, which is drastically different from the 75% used in MAE and the 25% used in PIXEL. In Section 4.4, we demonstrate that when combined with the text masking objective, it is important to deploy the image masking objective but best to keep the masking rate low.

Text masking and prediction. The image prediction objective is effective in training the model to understand the visual representation of the text. However, they have several drawbacks: (1) the training is often unstable due to the extreme contrast of black and white pixels, especially when the masking rate is high; (2) patch masking, even with the span masking strategy, still often leaks a significant amount of shallow cues; (3) pixel prediction does not model uncertainty well—while a text objective can model a probability distribution over a vocabulary, the image decoder often predicts blurry gray pixels, or a superposition of several possible words, as shown in Figure 4.

We believe that a text prediction objective is more effective in learning the language, and combining it with the image prediction can lead to both strong visual and text understanding capabilities. We first randomly mask out tokens in the input text and then render the corrupted text as the screenshot. We then add an autoregressive text decoder to recover the corrupted text, similar to BART (Lewis et al., 2020). By default, we use a masking rate of 25% with uniform masking, replacing masked tokens with a special token `<mask>`, and merging adjacent `<mask>` tokens.

3.4 Designs to Stabilize Training

In our preliminary experiments, we observe that training screenshot LMs is often unstable in the form of optimization stalling or loss spikes. This is likely caused by the highly polarized and imbalanced distribution of pixel values in text-rendered images (either black or white, and mostly white). We identify several design choices that are critical, removing of which can lead to performance degradation, training stagnation, or loss spikes.

Pixel-value preprocessing. We follow Rust et al. (2023) and adopt the following preprocessing: (1) The input pixel values are normalized to $[0, 1]$;[2] (2) The ground truth pixel values used for calculating the MSE loss are standardized by the means and standard deviations of the *patch*, *i.e.,* $\text{target}' = (\text{target} - \text{mean})/\text{std}$. While the input normalization is critical for training stability, we find the target standardization significantly improves the final performance.

Attention masks and end-of-sequence patch. Following Rust et al. (2023), we mask out attentions to the white empty patches after the end of the text sequence; we also add an end-of-sequence black patch at the end of the text, Removing either will lead to higher chances of training collapse.

Text prefix. We observe that the image prediction loss of screenshot LMs often goes through a "plateau" phase at the beginning of training, as shown in Figure 6. In this phrase, the model only learns to predict grey blur patches and with a certain chance, the training loss never decreases and the training stagnates. We find that rendering a text prefix at the beginning of the input sequence can reduce the chance of stagnation and start the loss decrease earlier. The intuition is that the text prefix provides the model with an easy and stable target to learn. We also observe that a longer text prefix has more significant effect. In our experiments, all models have a text prefix of "`Beginning of the sequence:`". Appendix G shows examples of the rendered screenshots with the text prefix.

Embedding layernorm. We observe that a model with higher masking rates has a much higher chance to suffer training collapse. In such cases, adding a layer normalization immediately after the input embeddings can mitigate this problem while not inducing much change in the loss. Though our main model does not have the embedding layernorm, some of our ablations employ the design to accommodate training stability. We offer more details in Appendix F.

3.5 Fine-tuning for Downstream Tasks

We evaluate our screenshot models in a similar way as BERT: we fine-tune and evaluate them on natural language understanding datasets. Since our model has an encoder-decoder architecture, there are two ways of fine-tuning:

Encoder-only. We simply keep the image encoder and discard both the image and the text decoder. Following PIXEL, we take the average representation of the last layer as the sentence representation, and feed it to a linear layer for classification or regression tasks. By default, we use the encoder-only fine-tuning for evaluation.

Sequence-to-sequence (s2s). In this setting, we leverage the combination of the image-encoder and the text-decoder for downstream tasks via fine-tuning. The model is trained to directly generate text for these tasks (*i.e.,* "good" or "bad" for a sentiment classification task). For all the details, please refer to Appendix D.

[2]The other common strategy is to standardize the input pixel values, which is adopted by Lee et al. (2023). However, we find such preprocessing hinders the training stability and the performance in the text-only screenshot setting.

Model	$\|\theta\|$	PP	TP	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
BERT[†]	110M	-	-	84.0/84.2	87.6	91.0	92.6	60.3	88.8	90.2	69.5	83.0
DONUT[*]	143M	✗	✓	64.0/-	77.8	69.7	82.1	13.9	14.4	81.7	54.0	57.2
CLIPPO[♣]	93M	✗	✗	77.7/77.2	85.3	83.1	90.9	28.2	83.4	84.5	59.2	74.0
PIXEL[†]	86M	✓	✗	78.1/78.9	84.5	87.8	89.6	38.4	81.1	88.2	60.5	76.0
PIXAR[◇]	85M	✓	✗	78.4/78.6	85.6	85.7	89.0	39.9	81.7	83.3	58.5	75.3
★ PTP	86M	✓	✓	**80.9/81.1**	**87.4**	**89.6**	**92.0**	**45.7**	**87.2**	**89.7**	**68.7**	**80.2**
★ PTP_{s2s}	268M	✓	✓	82.2/82.6	87.7	90.4	92.5	48.8	83.8	90.6	67.7	80.5

Table 1: Validation results for PTP and baseline models fine-tuned on GLUE. We report F1 scores for QQP and MRPC, Matthew's correlation for CoLA, Spearman's correlation for STS-B, and accuracy for others. We report baseline results from Rust et al. (2023)[†], Borenstein et al. (2023)[*], Tschannen et al. (2023)[♣], and Tai et al. (2024)[◇]. The averaged results are calculated without MNLI-MM. We report the number of parameters $|\theta|$ used in fine-tuning. PP: the model uses patch prediction; TP: the model uses text prediction. "PTP" denotes the encoder-only fine-tuning and "PTP_{s2s}" denotes the sequence-to-sequence fine-tuning.[3]

4 Experiments

4.1 Setup

Pre-training. We pre-train our models on the English Wikipedia and BookCorpus (Zhu et al., 2015) corpora for 16 epochs with a batch size of 256 (roughly 1M steps). For each input instance, we render a 256-token text sequence to an image with size 16×8464 (529 patches, consistent with Rust et al., 2023). More details on pre-training are provided in Appendix C.

Fine-tuning. We fine-tune our models on datasets from the GLUE benchmark (Wang et al., 2019). We run grid search and report the average validation performance of 3 random seeds. Appendix D provides more details on fine-tuning.

Baselines. We compare our models with several text-only and screenshot LM baselines.

BERT (Devlin et al., 2019). We compare to BERT because it is a bidirectional text-only LM at a similar scale (110M) and it uses the same pre-training corpora as ours. Note that BERT adopts more epochs over the training data than ours (40 epochs vs. our 16 epochs).

Donut (Kim et al., 2022). Donut is an encoder-decoder model that takes document images as input and outputs text. It is pre-trained with a pseudo-OCR task on IIT-CDIP (Lewis et al., 2006) and synthetic images generated from English, Chinese, Korean, and Japanese Wikipedia.

CLIPPO (Tschannen et al., 2023). This is a variant of CLIP (Radford et al., 2021) that utilizes one single vision encoder to process both images and text that is rendered as images. We report the GLUE performance of CLIPPO trained on WebLI (Chen et al., 2023) and 25% C4 (Raffel et al., 2020), which is a *significantly* larger pre-training corpus than ours.

PIXEL (Rust et al., 2023). PIXEL is a screenshot LM that is trained with only a patch masking and prediction objective. The other configurations (e.g., pre-training corpora and hyperparameters) are mostly the same as ours. A comparison between PIXEL and our model can help us understand the effect of our new training objective.

PIXAR (Tai et al., 2024). PIXAR is a concurrent work that explores autoregressive extensions of PIXEL. To be able to generate text via generating images, PIXAR adopts adversarial training and uses OCR softwares to extract the text. PIXAR uses the same training data as PIXEL.

4.2 Main Results: PTP Outperforms Other Screenshot LMs Significantly

Table 1 shows the main results on the validation sets of the GLUE benchmark. We also report the full results with standard deviation in Table 13 and the test results of our models and reproduced baselines in Table 14. Firstly, compared to previous state-of-the-art screenshot LMs, our PTP achieves significantly better performance on almost all GLUE tasks, indicating that our objective leads to better language understanding capabilities. Our model improves upon the previous state-of-the-art by up to 8% on specific tasks and more than 4% on average. Comparing PTP to BERT, the performance gap is substantially reduced—on 6 out of the 8

[3]For Pix2Struct, there are no publicly reported GLUE results. In our preliminary experiments, the performance on GLUE tasks significantly lags behind other baselines, so we leave it out in our comparison.

PM	PP	TM	TP	MNLI	SST-2	MRPC	RTE
✓	✓	✗	✗	78.6	89.2	88.5	66.5
✓	✗	✗	✓	79.5	90.0	88.7	66.7
✗	✗	✓	✓	**82.4**	90.9	86.5	62.0
✓	✗	✓	✓	81.1	91.0	88.1	63.8
✓	✓	✓	✓	80.9	**92.0**	**89.7**	**68.7**

Table 2: Ablations on different training objectives. PM: patch masking; PP: patch prediction; TM: text masking; TP: text prediction. The patch masking rate is 10% with span masking and the text masking rate is 25%.

tasks, the difference is within 2% (for the previous best model, this number is 1 out of the 8 tasks).

4.3 Ablation on Training Objectives

Several previous works explored patch masking and text masking separately (Rust et al., 2023; Lee et al., 2023), while we combine both for the first time. Here we conduct ablations to study the efficacy of (1) patch masking, (2) patch prediction, (3) text masking, and (4) text prediction. Table 2 demonstrates a clear comparison: while using the text objective alone is better than predicting image patches, combining both leads to significantly improved results, showing that the model trained with both objectives can better understand the text within the screenshot images.

4.4 Ablation on Masking Rates

Table 3 shows the comparison with different masking rates. The first unique phenomenon we observe is that a high patch masking rate (e.g., 40%) leads to frequent loss spikes and training collapses. We then see that with a smaller patch masking rate (10%) and a 25% text masking rate, the model can achieve the best fine-tuning performance. We hypothesize that the patch masking mostly helps learn the visual representations of the text (hence no need for a very high masking rate), while the text masking and prediction objective plays a pivotal role in facilitating learning the language. We also show in Table 16 that using *span* patch masking improves over uniform patch masking.

4.5 Ablation on Patch Sizes

Table 4 shows that the best patch size varies depending on the masking rates: when using a patch masking rate of 25% (span), a larger patch size of 16×32 is better; when using a patch masking rate of 10% (span), the smaller 16×16 is better; using larger patches (e.g., 16×64) leads to significant but non-catastrophic performance drops, yet they also come with efficiency gains by reducing the number of tokens needed to encode the same information. We hypothesize that patch sizes have a twofold effect: while a larger patch size reveals less superficial clues, it also reduces the sequence length and leaves the model less compute to process the input. Note that one factor we ignore is that changing the patch size modifies the span masking behavior—larger patch sizes are effectively larger spans at a smaller patch size—and we leave it out for future research.

Mask Rate		MNLI	SST-2	MRPC	RTE
Patch	Text				
5%	25%	79.2	90.0	85.7	62.3
10%	10%	80.0	90.2	88.7	64.5
10%	25%	**80.9**	**92.0**	**89.7**	**68.7**
25%	25%	80.8	90.7	88.6	65.3
10%	40%	79.6	90.2	85.1	62.2
25%	40%	80.6	90.9	88.9	67.9

Table 3: Results on different masking rates. All patch masking has span masking.

PM	Patch Size	MNLI	SST-2	MRPC	RTE
25%	16×16	80.8	90.7	88.6	65.3
	16×32	80.7	91.6	**89.7**	67.4
10%	16×16	**80.9**	**92.0**	**89.7**	**68.7**
	16×32	80.4	91.2	89.4	66.5
	16×64	76.7	88.5	88.1	64.4

Table 4: Ablation study on patch sizes. We show that the optimal patch size depends on the masking rates. "PM" refers to the span patch masking rate of the input image. All models here use a 25% text masking rate.

5 Extension to Autoregressive Screenshot LMs

Autoregressive LMs have become the predominate form of large LMs due to their powerful generation capabilities and emerging properties such as in-context learning (Brown et al., 2020; OpenAI, 2023; Touvron et al., 2023). Nevertheless, existing screenshot LMs[4] are built with an encoder-decoder architecture. In this section, we explore the feasibility of extending PTP to decoder-only, autoregressive screenshot LMs.

5.1 Methods

Inspired by Bavishi et al. (2023), we employ a single decoder architecture, where both visual and

[4] Concurrent work PIXAR (Tai et al., 2024) also explores autoregressive screenshot LMs but they generate images and rely on OCR softwares to turn them into text.

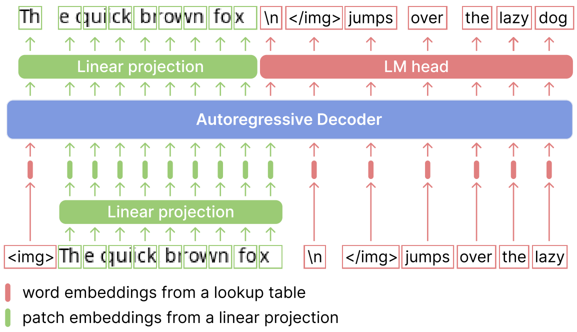

Figure 5: Our autoregressive screenshot LM.

textual inputs are first mapped into token embeddings and then processed by the same Transformer backbone. The input of the model encompasses two segments: the first segment is a sequence of patches that collectively constitutes a screenshot, while the second segment is a sequence of text tokens that follows the screenshot content. Unlike Bavishi et al. (2023) which only trains the model to predict the text segment, we apply an autoregressive objective on *both* the screenshot and the text segments. Figure 5 illustrates our autoregressive model and details are as follows.

Input format. Given a text sequence with m text tokens, we first split it into the screenshot segment (m_s tokens) and the text segment (m_t tokens), where $m_s + m_t = m$. The screenshot image is of size $p_h \times np_w$, where n is the number of patches. We insert three special tokens into the screenshot segment: a beginning-of-image token <img>, an image new line token \n[5], and an end-of-image token </img>. The special tokens aim to inform the model the boundary between the screenshot and text segments. In our experiments, we set $m_s = m_t = 256$ and $n = 512$.

By default, we use the same rendering strategies as our encoder-decoder model: we use the Google `Noto Sans` fonts, a font size of 10 and a line space of 6. On average, one token produced by the LLaMA (Touvron et al., 2023) tokenizer takes 1.28 patches when the patch size is 16×16.

Architecture. We follow the same architecture as LLaMA (Touvron et al., 2023), one of the most popular open-source autoregressive LMs. We experiment with two model sizes: (1) a smaller model with 380M parameters which we train from scratch, and (2) a larger model with 1.3B parameters which we continue training from a pre-trained Sheared-LLaMA checkpoint (Xia et al., 2023). For detailed configurations, please refer to Appendix E.

For the image patch input, we use a linear projection to map the pixel values to patch embeddings; For the text token input, we use their corresponding word embeddings. The patch and text embeddings are jointly fed into the Transformer blocks.

Training objectives. We adopt both a next patch prediction and a next token prediction objective, as illustrated in Figure 5. For the next patch prediction, we take the last layer representation, use a linear projection to map it to pixel values, and calculate the MSE loss. For the next text token prediction, we use an LM head and calculate the cross-entropy loss.

[5]This is to follow Bavishi et al. (2023) which allows the model to know that there is a new line of patches without using two-dimension positional embedding.

Model	Context	PPL
Text only	None	10.28
Ours	256 text tokens (in screenshots)	9.66
w/o patch pred	256 text tokens (in screenshots)	10.77
Text only	256 text tokens (in text)	8.60

Table 5: Comparisons between 380M LMs trained from scratch. The input consists of two parts: a segment of additional "context", and subsequent 256 text tokens which the perplexity (PPL) is evaluated on. "w/o patch pred": without the patch prediction objective.

Model	Context	PPL
Text only	None	10.20
Ours	256 text tokens (in screenshots)	9.09
Text only	256 text tokens (in text)	7.68

Table 6: Comparison between 1.3B LMs fine-tuned from Sheared-LLaMA (Xia et al., 2023).

5.2 Experiment Results

Evaluation setting. Our main goal is to verify whether the autoregressive screenshot LMs can understand the text from the screenshot context. The screenshot LM is given 256 text tokens in screenshot context and 256 text tokens in the text format, and its text-only counterpart is given just 256 text tokens. Then we measure perplexity on the last 256 text tokens. If the screenshot LM can effectively utilize the screenshot context, it will achieve lower perplexity compared to the text-only baseline.

Training from scratch. We train 380M parameter LLaMA-based models from scratch on the English Wikipedia and BookCorpus (Zhu et al., 2015) corpora for 16 epochs. Details are in Appendix E.

Table 5 shows that our autoregressive screenshot LM is able to effectively utilize the screenshot context and reduce the validation perplexity ($10.28 \rightarrow 9.66$). We also conduct an ablation without the patch prediction objective, which performs significantly worse. When compared to the text-only baseline using the same additional context but in text modality, there is still a significant gap. We hypothesize that the gap comes from two aspects: (1) training on screenshots is less effective than training on plain text; (2) it is more challenging to process the content in screenshots than in text.

Fine-tuning pre-trained LMs. We also fine-tune a pre-existing text-only LM, Sheared-LLaMA-1.3B (Xia et al., 2023), with our autoregressive screenshot objective on RedPajama (TogetherAI, 2023) for 5 billion tokens. More details are provided in Appendix E.

Table 6 shows that our objective can be effectively deployed for fine-tuning an existing LM, where our model improves the perplexity by using the additional screenshot context. Though the screenshot model still has a substantial gap to the text-only baseline when the text-only model uses the same context in text, it is a first step towards effective autoregressive screenshot modeling.

6 Related Work

Multimodal LMs. A majority of work along the line focuses on effective adaptation of visual representations—acquired via a separate visual encoder (Radford et al., 2021; Dosovitskiy et al., 2021)—into the text LMs. One approach is to incorporate the visual representations via cross-attention (Alayrac et al., 2022; Li et al., 2023b; Bai et al., 2023). More works directly use visual embeddings as input "tokens" of LMs (Lu et al., 2019; Liu et al., 2023b,a; Zhang et al., 2023; Gao et al., 2023; Wang et al., 2023; Chen et al., 2023; Driess et al., 2023), sometimes with additional processing (Li et al., 2023c; Dai et al., 2023; Zhu et al., 2023a). Bavishi et al. (2023); Li et al. (2023a) instead directly process the patches with a linear layer and use the embeddings as input, omitting the additional visual encoder.

Screenshot LMs. Two motivations inspire the development of screenshot LMs in previous literature. The first one is to develop tokenizer-free models for the purpose of better cross-lingual transferability. Early work dates back to Meng et al. (2019) which explore glyph embedding for Chinese characters; Salesky et al. (2021) adopt visual text representations on machine translation tasks to improve robustness. The representative work along this line is PIXEL, where Rust et al. (2023) train a ViT-MAE model over text-rendered images with masked patch prediction. Compared to its text-only counterpart, PIXEL achieves better performance on non-Latin languages and is more robust toward orthographic noises, but it lags behind on English tasks. Subsequent literature explores rendering strategies (Lotz et al., 2023), extension to historical documents (Borenstein et al., 2023), and generating text via generating images (Tai et al., 2024; Li et al., 2023d).

The second motivation is to build end-to-end systems for understanding visually-situated text, for example, scanned documents, webpages, or UIs. Early works are mostly pipeline systems where they use OCR tools (Huang et al., 2022; Powalski et al., 2021; Appalaraju et al., 2021) or feed view hierarchy information (Li et al., 2020; Bai et al., 2021). Recent works explore end-to-end models that take in solely visual inputs and generate text (Li and Li, 2023; Davis et al., 2022; Aggarwal et al., 2023; Kim et al., 2022; Zhu et al., 2023b). Pix2Struct (Lee et al., 2023), as one of the latest development, pre-trains encoder-decoder models on large-scale webpage screenshots by masking out certain HTML elements and predicting the masked HTML code. It achieves state-of-the-art performance on several visually-situated language understanding tasks and inspires followup works on table understanding (Alonso et al., 2023), UIs (Shaw et al., 2023), and more.

7 Conclusion

We introduce PTP, a new objective for training screenshot LMs by predicting both masked image patches and masked text. Our model achieves the state-of-the-art results on GLUE and for the first time pushes the performance of screenshot LMs close to their text-only counterparts. We also demonstrate the effectiveness of our objective on autoregressive screenshot LMs.

Numerous challenges persist in the development of screenshot LMs: for example, the patch prediction objective often makes the training unstable; the model is less efficient than the text-only LMs due to the longer input; We hope that our work will inspire more effort in the domain, and we look forward to stronger screenshot LMs in the future.

Limitations

Our work focuses on text-only screenshot LMs, which is a simplified setting of the general screenshot LMs. Though we argue that understanding text is the most challenging and fundamental aspect of screenshot LMs, we acknowledge that our findings may not generalize to the real screenshot scenarios. We will explore extending our models to real-world screenshots as a future direction.

Our ablation study, though extensive, is not exhaustive. For example, due to limited computational resources, we are unable to explore all possible combinations of masking rates and masking strategies. Considering that changing the patch size will also affect the masking (larger patches will essentially lead to more span masking), a more comprehensive study is needed to fully understand the effect of masking and the optimal setting. We also did not thoroughly explore the pre-training hyperparameters and simply followed existing works. Regardless, we believe that the above limitations do not affect our main findings and contributions.

Ethical Considerations

We do not foresee any direct ethical concerns arising from our work. Our research involves training and evaluating language models, which carry the same ethical considerations as other LM research, including but not limited to bias from pre-training data, bias towards the English language, and environmental impact from the large amount of computation required for the experiments.

References

Kriti Aggarwal, Aditi Khandelwal, Kumar Tanmay, Owais Khan Mohammed, Qiang Liu, Monojit Choudhury, Hardik Chauhan, Subhojit Som, Vishrav Chaudhary, and Saurabh Tiwary. 2023. DUBLIN: Visual document understanding by language-image network. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 693–706.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. 2022. Flamingo: a visual language model for few-shot learning. In *Advances in Neural Information Processing Systems*.

Iñigo Alonso, Eneko Agirre, and Mirella Lapata. 2023. Pixt3: Pixel-based table to text generation. *arXiv preprint arXiv:2311.09808*.

Srikar Appalaraju, Bhavan Jasani, Bhargava Urala Kota, Yusheng Xie, and R Manmatha. 2021. Docformer: End-to-end transformer for document understanding. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 993–1003.

Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, and Blaise Agüera y Arcas. 2021. Uibert: Learning generic multimodal representations for ui understanding. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond.

Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge.

Rohan Bavishi, Erich Elsen, Curtis Hawthorne, Maxwell Nye, Augustus Odena, Arushi Somani, and Sağnak Taşırlar. 2023. Fuyu-8b: A multimodal architecture for ai agents.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth PASCAL recognizing textual entailment challenge. In *TAC*.

Nadav Borenstein, Phillip Rust, Desmond Elliott, and Isabelle Augenstein. 2023. PHD: Pixel-based language modeling of historical documents. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 87–107.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *the 11th International Workshop on Semantic Evaluation (SemEval-2017)*.

Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, Piotr Padlewski, Daniel Salz, Sebastian Goodman, Adam Grycner, Basil Mustafa, Lucas Beyer, Alexander Kolesnikov, Joan Puigcerver, Nan Ding, Keran Rong, Hassan Akbari, Gaurav Mishra, Linting Xue, Ashish V Thapliyal, James Bradbury, Weicheng Kuo, Mojtaba Seyedhosseini, Chao Jia, Burcu Karagol Ayan, Carlos Riquelme Ruiz, Andreas Peter Steiner, Anelia Angelova, Xiaohua Zhai, Neil Houlsby, and Radu Soricut. 2023. PaLI: A jointly-scaled multilingual language-image model. In *International Conference on Learning Representations (ICLR)*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. 2023. Instructblip: Towards general-purpose vision-language models with instruction tuning.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:16344–16359.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *International Conference on Machine Learning (ICML)*, pages 933–941. PMLR.

Brian Davis, Bryan Morse, Brian Price, Chris Tensmeyer, Curtis Wigington, and Vlad Morariu. 2022. End-to-end document recognition and understanding with dessurt. In *European Conference on Computer Vision*, pages 280–296. Springer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional Transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

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

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. 2023. Palm-e: an embodied multimodal language model. In *International Conference on Machine Learning (ICML)*.

Peng Gao, Jiaming Han, Renrui Zhang, Ziyi Lin, Shijie Geng, Aojun Zhou, Wei Zhang, Pan Lu, Conghui He, Xiangyu Yue, Hongsheng Li, and Yu Qiao. 2023. Llama-adapter v2: Parameter-efficient visual instruction model. *arXiv preprint arXiv:2304.15010*.

Gemini Team. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16000–16009.

Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. 2022. Layoutlmv3: Pre-training for document ai with unified text and image masking. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 4083–4091.

Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. 2016. pybind11 — seamless operability between c++11 and python. Https://github.com/pybind/pybind11.

Geewook Kim, Teakgyu Hong, Moonbin Yim, JeongYeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. 2022. Ocr-free document understanding transformer. In *European Conference on Computer Vision*, pages 498–517. Springer.

Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. 2023. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning (ICML)*, pages 18893–18912.

David Lewis, Gady Agam, Shlomo Argamon, Ophir Frieder, David Grossman, and Jefferson Heard. 2006. Building a test collection for complex document information processing. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 665–666.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Association for Computational Linguistics (ACL)*, pages 7871–7880.

Bo Li, Peiyuan Zhang, Jingkang Yang, Yuanhan Zhang, Fanyi Pu, and Ziwei Liu. 2023a. Otterhd: A high-resolution multi-modality model.

Bo Li, Yuanhan Zhang, Liangyu Chen, Jinghao Wang, Jingkang Yang, and Ziwei Liu. 2023b. Otter: A multi-modal model with in-context instruction tuning. *arXiv preprint arXiv:2305.03726*.

Gang Li and Yang Li. 2023. Spotlight: Mobile UI understanding using vision-language models with a focus. In *International Conference on Learning Representations (ICLR)*.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023c. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models.

Junyi Li, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023d. Renderdiffusion: Text generation as image generation. *arXiv preprint arXiv:2304.12519*.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile UI action sequences. In *Association for Computational Linguistics (ACL)*, pages 8198–8210.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations (ICLR)*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023a. Improved baselines with visual instruction tuning.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023b. Visual instruction tuning. In *NeurIPS*.

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. 2017. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*.

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

Jonas Lotz, Elizabeth Salesky, Phillip Rust, and Desmond Elliott. 2023. Text rendering strategies for pixel language models. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 10155–10172.

Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in Neural Information Processing Systems (NeurIPS)*, 32.

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2263–2279.

Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. 2021. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2200–2209.

Yuxian Meng, Wei Wu, Fei Wang, Xiaoya Li, Ping Nie, Fan Yin, Muyu Li, Qinghong Han, Xiaofei Sun, and Jiwei Li. 2019. Glyce: Glyph-vectors for chinese character representations. *Advances in Neural Information Processing Systems (NeurIPS)*, 32.

OpenAI. 2023. GPT-4 Technical Report.

Rafał Powalski, Łukasz Borchmann, Dawid Jurkiewicz, Tomasz Dwojak, Michał Pietruszka, and Gabriela Pałka. 2021. Going full-tilt boogie on document understanding with text-image-layout transformer. In *Document Analysis and Recognition–ICDAR 2021: 16th International Conference, Lausanne, Switzerland, September 5–10, 2021, Proceedings, Part II 16*, pages 732–747. Springer.

Zhangyang Qi, Ye Fang, Mengchen Zhang, Zeyi Sun, Tong Wu, Ziwei Liu, Dahua Lin, Jiaqi Wang, and Hengshuang Zhao. 2023. Gemini vs gpt-4v: A preliminary comparison and combination of vision-language models through qualitative cases. *arXiv preprint arXiv:2312.15011*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, pages 8748–8763.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text Transformer. *The Journal of Machine Learning Research (JMLR)*, 21(140).

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Phillip Rust, Jonas F. Lotz, Emanuele Bugliarello, Elizabeth Salesky, Miryam de Lhoneux, and Desmond Elliott. 2023. Language modelling with pixels. In *International Conference on Learning Representations (ICLR)*.

Elizabeth Salesky, David Etter, and Matt Post. 2021. Robust open-vocabulary translation from visual text representations. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 7235–7252.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. 2023. From pixels to UI actions: Learning to follow instructions via graphical user interfaces. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Yintao Tai, Xiyang Liao, Suglia Alessandro, and Antonio Vergari. 2024. Pixar: Auto-regressive language modeling in pixel space. *arXiv preprint arXiv:2401.03321*.

TogetherAI. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*.

Michael Tschannen, Basil Mustafa, and Neil Houlsby. 2023. Clippo: Image-and-language understanding from pixels only. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11006–11017.

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 (NIPS)*, 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Cogvlm: Visual expert for pretrained language models.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association of Computational Linguistics (TACL)*, 7.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-art natural language processing. In *Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*.

Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. 2023. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023a. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.

Wang Zhu, Alekh Agarwal, Mandar Joshi, Robin Jia, Jesse Thomason, and Kristina Toutanova. 2023b. Efficient end-to-end visual document understanding with rationale distillation. *arXiv preprint arXiv:2311.09612*.

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 Rendering Strategy

There are two rendering strategies: (1) pre-rendering the text and storing the screenshots, or (2) rendering the text on-the-fly during training. While the first one is more efficient (for training), it requires a large amount of storage space and has limited flexibility (e.g., we need to regenerate the whole dataset if we need to change the font). We choose to render the text in an online fashion, and thus we need a fast renderer to avoid data processing becoming a bottleneck.

At the time of writing, various renderers are available that provide varying combinations of features. However, they are either too slow or not compatible with PyTorch's multi-process data loading. We then developed our own in-house renderer. Our renderer is implemented in C++ and meshed to Python via PyBind11 (Jakob et al., 2016). We use the FreeType library to get the glyphs for the characters. It is helpful that FreeType already provides the horizontal and vertical offsets to be applied to each character, so we can simply render each character in turn. We allow the caller to control the font, font size, height, width, line spacing, word spacing, and margins. We observed roughly a $6.4\times$ speedup compared to PyGame[6], a renderer used by Rust et al. (2023).

B Model Architectures

Our model architecture mainly consists of three components: (1) an image encoder; (2) an image decoder; and (3) a text decoder. All of these components are based on transformers. Following Rust et al. (2023), we add a `CLS` token at the beginning for the encoder input. During pre-training, the image encoder takes input from unmasked image patches. The encoder output is used by the image decoder (along with masked tokens, represented by a mask embedding). The image decoder predicts only on the masked patches. The text decoder uses cross attention to attend to outputs of the image encoder (hence only the unmasked image patches).

We strictly follow Rust et al. (2023); He et al. (2022) for the image encoder and the image decoder settings. They are standard ViTs with pre-layer normalizations. For the text decoder, we follow Lee et al. (2023), which uses a GLU (Gated Linear Unit; Dauphin et al., 2017) version of MLP (Shazeer, 2020). We also change the positional embedding of the text decoder to a learnable absolute positional embedding for simplicity. For "our text LM", we use the same configurations as our encoder-decoder model except that we also use GLU for the text encoder. Specific hyperparameter settings for the architecture are provided in Table 7. Note that for our 16×16 patch models, to form a direct comparison with PIXEL, we follow the setting of Rust et al. (2023) and use an input image size of 16×8464; for other patch size, we use an image size of 16×8192.

In our preliminary experiments, we find that adding a layernorm after the input linear projection will lead to more stable training and fewer loss spikes, while not changing the final performance much. We adopt the input layernorm for some of the ablation models demonstrated in Table 15.

Component	Image Encoder	Image Decoder	Text Decoder
Image patch size	16×16	-	-
Hidden size	768	512	768
Intermediate size	3072	2048	3072
#Attention heads	12	16	12
#Layers	12	8	12

Table 7: Architecture configurations of PTP.

C Pre-training Details

We use Huggingface's Transformers package (Wolf et al., 2020) to perform all our pre-training and fine-tuning experiments. We provide the data and optimization hyperparameters during the pre-training of

[6] https://github.com/pygame/pygame

our PTP model in Table 8. We use FlashAttention (Dao et al., 2022) to speedup training. We also have several special design choices: (1) We always render a black patch at the end of the text, indicating the end of the sequence (following PIXEL); (2) we do not attend to the white patches after the end-of-sequence black patch (following PIXEL); (3) we normalize the input pixel values and standardize the target pixel values in each patch before calculating the MSE loss (following PIXEL); (4) we always render a prefix text `Beginning of the sequence:` at the beginning, which serves as an "anchor point" and helps warmup the training (new compared to PIXEL). We find that the above designs are crucial for making the training stable and avoiding stalling or loss spikes.

Parameter	Value
Data	
Image size (height, width)	$(16, 8464)$
Image mode	RGB
Font	Google Noto Sans
Font size	10
Line space	6
Newline symbol	////
Patch masking rate	10%
Patch span masking	`true`
Patch span masking max length	6
Patch span masking cumulative weights	$\{0.2, 0.4, 0.6, 0.8, 0.9, 1\}$
Text masking rate	25%
Text masking token	`<mask>`
Merge consecutive text masks	`true`
Optimization	
Learning rate	$1.5e-4$
Minimum learning rate	$1.0e-5$
Warmup	50K steps
Learning rate scheduler	Cosine decay (Loshchilov and Hutter, 2017)
Batch size	256
Optimizer	AdamW (Loshchilov and Hutter, 2019)
Mixed precision training	`fp16`
Number of epochs	16 (roughly 1M steps)

Table 8: Hyperparameters in PTP pretraining.

D Fine-tuning Hyperparameters

We fine-tune our models on datasets from the GLUE benchmark (Wang et al., 2019), including SST-2 (Socher et al., 2013), CoLA (Warstadt et al., 2019), MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016), RTE (Dagan et al., 2005; Bar Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), MRPC (Dolan and Brockett, 2005), QQP[7] and STS-B (Cer et al., 2017). We did not include WNLI due to its abnormal data distribution issue noted on the GLUE website[8]. We run grid search for fine-tuning and report the average of the best validation results over three seeds. Table 9 shows the hyperparameters used for fine-tuning. For rendering, we use the same rendering engine, font, and font size as in pre-training. We also add the trailing black patch and the prefix text to be consistent with pre-training. We mask out the attention to the white patches after the end-of-sequence black patch. We use an image size of $(16, 8192)$ for MNLI, QQP, QNLI, and RTE, and $(16, 4096)$ for the rest. We evaluate the model every 100 steps for MRPC, STS-B, and CoLA, every 250 steps for RTE, and every 500 steps for the remaining tasks. Unlike text models like BERT, we find that screenshot LMs are sensitive to the number of optimization steps instead of epochs of data, thus we control the total number of training steps, similar to Rust et al. (2023).

[7] https://www.quora.com/q/quoradata/

[8] https://gluebenchmark.com/faq

For sentence pair tasks, we render a `////` between the two sentences. We replace all newlines in the text with `////`. For the sequence-to-sequence setting, we show the corresponding label text for each task in Table 10. Specifically, for STS-B (a regression task), we follow the setting from T5 (Raffel et al., 2020), where we round up all the values to the nearest increment of 0.2; during evaluation, we convert the output text to floats and compute the metric using the original label values.

Parameter	**Value**
Optimizer	AdamW
Warmup steps	100
Learning rate scheduler	Linear decay
Mixed precision training	`fp16`
Random seeds	$\{42, 43, 44\}$
Learning rate	$\{1e-5, 3e-5, 5e-5\}$
Batch size	$\{32, 64, 256\}$
Training steps	$\{8000, 15000, 30000\}$

Table 9: Hyperparameters in PTP fine-tuning.

Task	**Label Text**
MNLI	`yes,maybe,no`
QNLI, QQP, MRPC, RTE, CoLA	`yes,no`
SST-2	`good,bad`
STS-B	`0.0,0.2,0.4,...`

Table 10: Label text for the sequence-to-sequence setting of GLUE fine-tuning.

E Autoregressive Screenshot LMs

For both the train-from-scratch and the fine-tuning-from-Sheared-LLaMA settings, we set each training instance to consist of 512 image patches (rendered from a text sequence with 256 text tokens) and its subsequent 256 text tokens. We also train text baselines with the same configuration, except that its inputs are sequences of 512 text tokens.

Table 11 shows the configurations of our 380M models and our 1.3B models. Both the screenshot versions and the text-only versions follow these configurations. Note that the 1.3B configuration follows Xia et al. (2023), as we fine-tune the models from Sheared-LLaMA-1.3B.

Component	**380M**	**1.3B**
Image patch size	16×16	16×16
Hidden size	1024	2048
Intermediate size	2816	5504
#Attention heads	16	16
#Layers	24	24

Table 11: Architecture configurations of our autoregressive models.

We follow most of the settings from the encoder-decoder experiments: we use FlashAttention to speedup training; we render the text with Google Noto Sans font, size 10, line space 6; we replace all the newline symbols with `////`; we set an attention mask to avoid attending to the white patches at the end; we normalize the pixel inputs and standardize the target pixels in each patch before calculating the MSE loss. There are also several differences: we do not use the prefix text or the end-of-sequence black patch, as we find autoregressive training is more stable and does not require these design choices.

Table 12 shows the hyperparameters used for training our 380M and 1.3B models. Note that the data for the two settings are different: we use Wikipedia and BookCorpus for the 380M train-from-scratch setting, and RedPajama (TogetherAI, 2023) for the fine-tuning from Sheared-LLaMA setting. In both cases, each training instance contains 512 text tokens; for a screenshot autoregressive model, the first 256 tokens are rendered as a 16×8192 image, which leads to 512 patch tokens with a patch size of 16×16.

Parameter	**Value (380M / 1.3B)**
Learning rate	$1.5e-4$
Minimum learning rate	0
Warmup	50K / 2K
Learning rate scheduler	Cosine decay
Batch size	256
Optimizer	AdamW
Mixed precision training	fp16
Number of steps	16 epochs (roughly 500K steps) / 50K steps

Table 12: Hyperparameters in autoregressive screenshot LM training.

F More Results

Training loss curve. Figure 6 shows the image prediction loss curve of our main PTP model. Distinct from text-only LMs whose loss usually drops quickly at the beginning of the training, screenshot LMs first go through a "plateau" phase (roughly 20K steps) and then the loss starts to decrease.

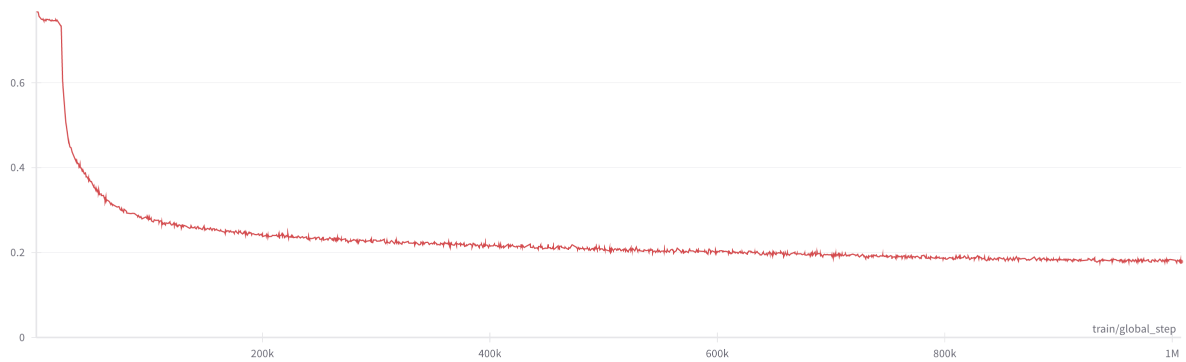

Figure 6: The patch prediction loss curve of our PTP model.

Main experiments. Table 13 shows our main GLUE results with standard deviation, averaged over three seeds. Table 14 shows the GLUE test results of our models. We select the best performing model on the validation set (with seed 42) and submit the test prediction results to the GLUE leaderboard.

We also add a new text-only LM baseline named **Our text LM**, for a more fair comparison in terms of both the amount of training data and the objective. This is a text-only encoder-decoder model with the same text objective as our screenshot LMs—we randomly mask 25% of the text tokens, replace them with <mask>, and use a decoder to recover the original sequence.

Embedding layernorm ablation. Table 15 shows the effect of using embedding layernorm on models with different masking rates. Note that the training stability results may vary depending on the hardware, software, and random seeds used. As the table has demonstrated, higher masking rates or larger patch sizes lead to an increased chance of training instability. Using embedding layernorm can prevent loss spikes in most cases.

Model	$\lvert\theta\rvert$	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
Our text LM	115M	$85.4_{0.2}$ / $85.4_{0.0}$	$88.3_{0.1}$	$92.2_{0.0}$	$92.6_{0.3}$	$56.3_{0.4}$	$90.1_{0.1}$	$90.8_{0.4}$	$62.5_{2.1}$
Our text LM_{s2s}	297M	$86.0_{0.1}$ / $86.0_{0.3}$	$88.5_{0.0}$	$92.4_{0.1}$	$92.7_{0.2}$	$58.8_{1.5}$	$89.3_{0.1}$	$91.6_{0.3}$	$74.2_{0.9}$
PTP	86M	$80.9_{0.1}$ / $81.1_{0.1}$	$87.4_{0.2}$	$89.6_{0.2}$	$92.0_{0.3}$	$45.7_{1.4}$	$87.2_{0.2}$	$89.7_{0.3}$	$68.7_{0.2}$
PTP_{s2s}	268M	$82.2_{0.1}$ / $82.6_{0.1}$	$87.7_{0.1}$	$90.4_{0.0}$	$92.5_{0.2}$	$48.8_{0.7}$	$83.8_{0.2}$	$90.6_{0.2}$	$67.7_{0.3}$

Table 13: GLUE validation results with standard deviation.

Model	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
BERT	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4
$\text{PIXEL}_{\text{reproduced}}$	73.5/73.7	69.7	85.0	88.7	9.9	79.4	85.1	56.8
PTP	79.9/79.4	69.8	88.4	90.0	37.9	79.4	85.7	62.0
PTP_{s2s}	81.8/80.9	70.5	89.3	91.6	43.1	87.9	87.5	61.7

Table 14: GLUE test results.

Patch Size	Mask Rate		w.o. Embedding LN	w. Embedding LN
	Patch	Text		
16×16	10%	25%	✓	✓
	10%	40%	✓	✓
	25%	25%	✓	✓
	25%	40%	✗	✓
16×32	10%	25%	✗	✓
16×64	10%	25%	✗	✓

Table 15: Effects of using embedding layernorm. ✓ indicates that the training can be completed smoothly while ✗ indicates that the training collapsed due to loss spikes. Note that this result may vary depending on the hardware, software versions, and random seeds.

Span masking. Table 16 shows the comparison between using span masking vs. not using span masking on image patches. We observe that at this masking rate, using span masking for image patches leads to a significantly better result.

Span Masking	MNLI	SST-2	MRPC	RTE
✓	**80.9**	**92.0**	**89.7**	**68.7**
✗	80.5	90.9	89.3	64.9

Table 16: Ablation on span masking. Both models use a 10% patch masking rate and a 25% text masking rate.

G Examples

Figure 7, Figure 8, and Figure 9 show how the prediction of our main model evolves during pre-training. The model becomes more and more capable of predicting longer masked spans.

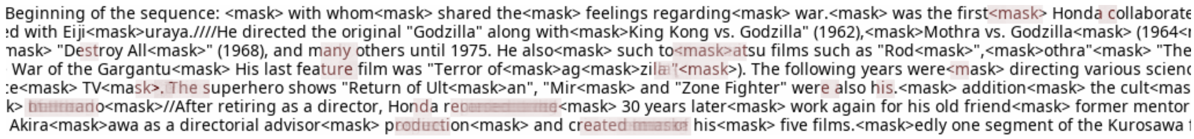

Figure 7: The prediction of our main model at 100K training step.

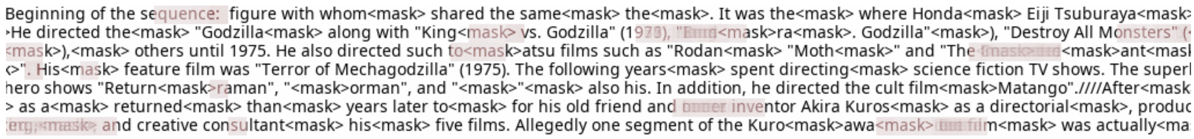

Figure 8: The prediction of our main model at 500K training step.

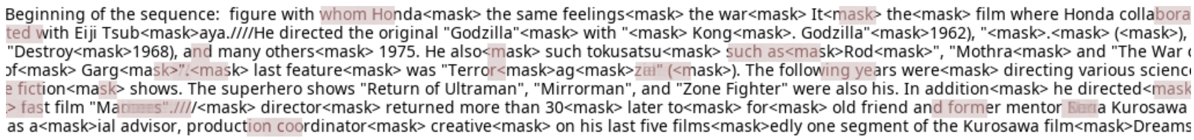

Figure 9: The prediction of our main model at 1M training step.

Figure 10 shows the prediction of the 25%span/25% (patch/text) masking model with 16×16 patch size. With more masking, accurately predicting the masked patches becomes increasingly difficult. Figure 11 shows the prediction of the 10%span/25% (patch/text) masking model with 16×32 patch size. The larger patch size essentially leads to more "span" masking, making the pre-training task more challenging.

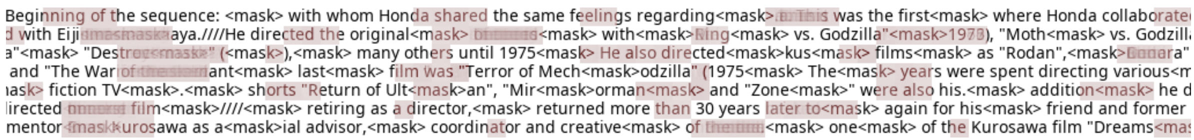

Figure 10: The prediction of the 25%span/25% (patch/text) masking model with 16×16 patch size.

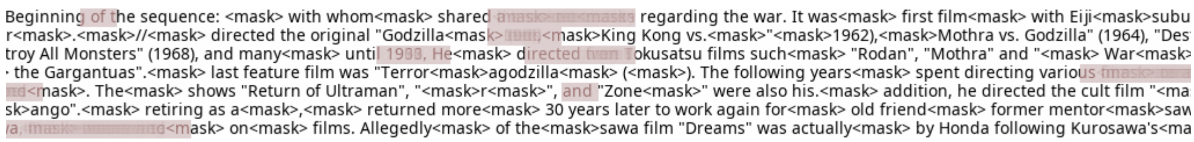

Figure 11: The prediction of the 10%span/25% (patch/text) masking model with 16×32 patch size.