VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

PHAM HUYNH TIN – 522H0150

NGUYEN TRUNG THANG – 522H0145

FINAL REPORT
DEEP LEARNING

HO CHI MINH CITY, 2025

VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

PHAM HUYNH TIN – 522H0150

NGUYEN TRUNG THANG – 522H0145

FINAL REPORT

DEEP LEARNING

Instructor

Assoc. PhD. Le Anh Cuong

HO CHI MINH CITY, 2025

ACKNOWLEDGEMENT

Dear Mr. Le Anh Cuong,

We would like to express our sincere and profound thanks to the teacher for the dedication and valuable sharing during the learning process. The enthusiasm, erudite knowledge as well as the encouragement of the teacher helped us not only master the knowledge but also inspired to explore more deeply and be more passionate about the field of study. The patience and dedication of the teacher have opened us with many knowledge doors, helping us to confidently move on the path of conquering new challenges.

We are extremely grateful for the precious moments that he has given us, advice and priceless motivation. Once again, sincerely thank you for all.

Sincerely welcome!

Ho Chi Minh City, March 24th, 2025

Authors

(Sign and write full name)

THE WORK IS COMPLETED

AT TON DUC THANG UNIVERSITY

I hereby declare that this is my own research project and is under the scientific guidance of Assoc. PhD. Le Anh Cuong. The research content and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluation were collected by the author from different sources and clearly stated in the reference section.

Project also uses a number of comments, assessments as well as data from other authors and other organizations, all with citations and source notes.

If any fraud is detected, I will take full responsibility for the content of my Project. Ton Duc Thang University is not involved in copyright violations caused by me during the implementation process (if any).

Ho Chi Minh City, March 24th, 2025

Authors

(Sign and write full name)

ABSTRACT

This report presents the implements of an Optical Character Recognition (OCR) system for extracting Vietnamese text from images. We implement a custom Sequence-to-Sequence model utilizing a CNN as the Encoder to extract imeage feature representation, and a Transformer Decoder to generate the corresponding character sequence. This model is trained directly on image regions cropped based on Ground Truth polygons and the corresponding Ground Truth text from the MC-OCR dataset. We integrate the trained Recognition model with an effective and pre-trained Text Detection modl, PaddleOCR (only use its Detection component). This pipeline runs PaddleOCR detection on the full image to identify text regions, then crops these regions and feeds them into our custom Recognition model for text content extraction. Quantitative evaluation results for the standalone Recognition model are presented on a separate test set using Ground Truth data (Test Loss, CER, accuracy). We also demostrate the capability of the pipeline through visualization of results on sample images from the test set and a user-provided image. This work showcases the potential of combining specialized models for each OCR stage to build a comprehensive text extraction system.

TABLE OF CONTENT

ABBREVIATIONS

OCR	Optical Character Recognition
CNNs	Convolutional Neural Networks
GT	Ground Truth
CSV	Comma Separated Value
CER	Character Error Rate
GPU	Graphics Processing Unit
LLMs	Large Language Model
MQA	Multi Query Attention
GQA	Grouped Query Attention

CHAPTER 1. ATTENTION

1.1 Self-Attention

⁕ **Mechanism of operation and main ideas:**

Self-Attention is a core mechanism in the Transformer architecture, which allows the model to learn relationships between tokens in a string without depending on the order or distance between them. Unlike sequential models like RNN, Self-Attention processes the entire chain at the same time, helping the model focus on related tokens regardless of their location. Specifically, for each token in the chain, the model creates three vectors:

- Query (Q): Representing the token is being focused on analysis.
- Key (K): Representing other tokens in the chain, used to compare with Query.
- Value (V): Contains the real information of tokens, taken with a weight based on the level of relevance.

Attention matrix is calculated by taking dot-phoduct between Q and K of the tokens, standardized by dividing by $\sqrt{d_k}$ (d_k is the dimension of the Vector Key) to stabilize the gradient, then apply the Softmax function to create the Attention.

The weight of this Attention is multiplied by V to create the output, indicating the linear combination of the tokens based on the level of relevance.

Formula:

1. Attention Scores: Calculate the level of relevance between positions:

$$Attention\ Scores\ = \frac{QxK^T}{\sqrt{d_k}}$$

2. Softmax: Standardize Scores into probability distribution:

$$Attention\ Weights\ = softmax(Attention\ Scores)$$

3. Combining value values according to the number of Attention:

$$Output\ =\ Attention\ Weights\ \times\ V$$

⁕ **Computational Complexity:**

- **Memory:**
  - Attention matrix size $n \times n$, with n is the string length, leads to storage costs are $O(n^2)$.
  - Matrices Q, K, V are sized $n \times d$, with d is the hidden dimension, so the total memory cost is $O(n^2 + n \times d)$.
  - This is a big drawback when processing a very long string.
- **Time**
  - QxK^T: $O(n^2 \times d)$, due to the performance of the middle matrix Q $(n \times d)$ and $K^T(d \times n)$.
  - Calculate Softmax and multiply with V: $O(n^2 \times d)$
  - Total time: $O(n^2 \times d)$ for each layer.

⁕ **Advantage:**

- Global Dependencies: Self-Attention is able to learn relationships between tokens in any position in the chain, suitable for natural language tasks that require long context.
- Parallel: Unlike RNN, Self-Attention allows processing the entire string at the same time, making good use of hardware like GPU.
- Flexibility: Can be applied to many types of data (text, images, sounds) by adjusting the input representation.

⁕ **Disadvantage:**

- High calculation costs: complexity $O(n^2)$ make self-attention not effective with long chain. For instance when n > 1024.
- Memory consumption: Attention matrix $n \times n$ requires large memory, especially with long chains.
- Difficult to optimize hardware: The storage and calculation of the large Attention matrix cause Bottleneck I/O on GPU.

- Lack of inherent position information: Self-Attention does not automatically encrypt token position, requires additional mechanisms such as Positional encoding or Rotary Positional Embedding (Rope).

⁕ **Compare with the original Self-Attention in transformer:**

- **The Similarity:** Self-Attention in LLMS is the original mechanism introduced in the article "Attention is all you need" (Vaswani et al., 2017). There is no theoretical difference between Self-Attention in Transformer and LLMS.
- **The Difference:**
    - Practical optimization: In modern LLMS, Self-Attention is often optimized by techniques such as Flashattention, Fused Kernels, or integrated with other mechanisms (MQA, GQA) to reduce memory costs and speed up.
    - Model size: LLMS usually has a much larger number of layers and dimensions, highlights the limitations of memory and calculation of the original Self-Attention.
    - Location encryption: original transformer uses static Positional encoding (sinusoidal or learned), while modern LLMS can use Rope or dynamic coding methods to improve performance.

⁕ **Application in modern LLM models:**

- **Bert and Encoder:** Self-Attention models are used in Encoder layers to learn bidirectional, suitable for tasks such as text classification, information extraction.
- **GPT and Decoder models:** In models such as GPT-3, LLAMA, Self-Attention are used in Decorder layers with Attention with mask (Causal Attention) to ensure only the previous tokens, in accordance with the text of text.

- **T5 and Encoder-Decoder models:** Self-Attention appears in both encoder (comprehensive context learning) and decoder (learning left to right), supporting machine translation tasks, text summaries.
- **Optimization model:** In Llama, Grok, or modern models, Self-Attention is often combined with techniques such as Flashattention or replaced with MQA/GQA to improve reasoning and training performance on long chains.
- **Long chain application:** Some LLMS (such as Transformer-XL, Longformer) modify Self-Attention to process longer chains by combining Sparse Attention or memory mechanism.

1.2 Multi-Query Attention (MQA)

⁕ **Mechanism of operation and main ideas:**

Multi-Query Attention (MQA) is a variant of Multi-Head Attention (MHA) in Transformer, designed to reduce memory costs and increase reasoning speed, especially suitable for large LLMS. In MQA, all heads share a set of key (k) and value (v), instead of each head has K and V as in MHA. However, Query (Q) is still maintained for each head to ensure the diversity in learning different aspects of data.

Mechanism of operation:

- From the input vector of each token, a set Q is created for each head, but only one K and a V is created for the entire model (not divided by head).
- The scalar part between the Q of each head and K is calculated to create the Attention matrix, then standardized by $\sqrt{d_k}$ and apply SoftMax to create Attention.
- This weight is multiplied by V to create output for each head, then the outputs are connected and linear projection.

Formula of one head:

$$Attention(Q_h, K, V) = softmax\left(\frac{Q_h x K^T}{\sqrt{d_k}}\right) \times V$$

In there, Q_h is Query of head, K and V are general for all heads

Multi-Head in MQA: Each head has its own Q, but K and V are shared, significantly reducing the number of parameters and memory compared to MHA.

⁕ **Computational Complexity:**

- In terms of calculation, MQA still retains the quantity of query heads, so the calculation of dot-phoduct between Q and K is still equivalent to MHA.
- However, the storage memory and bandwidth access cache K, V decrease in a large amount, namely reduced with the number of heads H (for example, decreases 8 times if there are 8 heads).
- This helps increase the effect efficiency, reduces delay and memory costs.

Type	**Parameters K, V**	**Inference Memory**	**FLOPs**
Multi-Head	$O(h \times d)$	$O(h \times n^2)$	$O(n^2 \times d)$
Multi-Query	$O(d)$	$O(n^2 + h \times n)$	$O(n^2 \times d)$
Efficiency	Reduces by h times	Significantly reduced	Similar

⁕ **Advantage:**

- Inference acceleration: Reduce K, V cache, reduce memory access bandwidth, help speed up new tokens in decoder.
- Memory saving: Cache K, V storage memory is significantly reduced, very important for LLM with tens of billions of parameters.
- Easy to integrate: MQA can be converted from Checkpoint MHA via "uptraining" (light-tuning) to keep the quality.

- Applied in large models: For example, Palm, Falcon, Llama-V2 used MQA to optimize Inference.

⁕ **Disadvantage:**

- Reducing the quality of representation: Due to K, V is compressed, reducing the diverse and rich performance of heads, leading to a reduction in model performance compared to traditional MHA.
- Difficulties in stable training: A number of studies noted that MQA can cause instability when training without supportive techniques.
- Reduce the parallel ability in the dispersion model: Because all heads shared K, V, when dispersed according to the head (Tensor parallelism), the calculation and cache K, V are duplicated, causing wasting resources.
- Not suitable for Tasks to require high accuracy.

⁕ **Compare with the original Self-Attention in transformer:**

- **The Similarity:**
    - MQA is based on the original Self-Attention, using Scaled Dot-Product Attention.
    - Both learn all overlapping relationships between tokens.
- **The Difference:**
    - The number of K, V: the original Self-Attention (in MHA) created K, V separately for each head, while MQA shared a K, V unique, reducing memory.
    - Memory cost: MQA saves more memory, especially with large heads, suitable for large -scale LLMS.
    - Performance: MQA may be inferior to MHA in some cases due to reduced diversity of K, V.

- Application: MQA is optimized for inference in modern LLMS, while the original Self-Attention is more popular in the original transformer models (such as Bert, GPT-1).

⁕ **Application in modern LLM models:**

- **LLaMA và LLaMA-2:** MQA is used to optimize reasoning, reduce memory and increase processing speed, especially in effective research models.
- **Grok (xAI):** Some Grok variants can use MQA to improve performance on limited hardware platforms.
- **Quick reasoning model:** MQA is popular in models deployed on equipment with limited resources, thanks to the ability to reduce memory.
- **Combined with other techniques:** MQA is usually integrated with Flash-Attention or Grouped-Query Attention (GQA) to optimize additional, as in LLAMA-3 or modern models.
- **Natural language tasks:** MQA supports tasks such as texting, translating, and answering questions, with an efficiency equivalent to MHA but saving resources.

1.3 Grouped-Query Attention (GQA)

⁕ **Mechanism of operation and main ideas:**

Grouped-Query Attention (GQA) is an intermediate mechanism between Multi-Stting (MHA) and Multi-Query Attention (MQA), designed to balance the calculated performance and the quality of representation in LLMS. In GQA, the heads are divided into groups, each sharing a set of key (k) and value (v), instead of each head has K, V separately (like mha) or all head shared a k, v unique (like mqa). Query (Q) is still maintained for each head to ensure diversity.

Mechanism of operation:

- The heads are divided into g group, with each group containing $\frac{h}{g}$ head (where h is the total head, g is the number of groups).
- Each group has a collection K and V, used by all heads in the group. Q is still calculated for each head.
- The scalar product between Q of each head and K of the corresponding group is calculated to create an Attention matrix, standardized $\sqrt{d_k}$ and apply SoftMax to create Attention.
- This weight is multiplied by V of the group to create output for each head, then the outputs are connected and linear projection.

Formula (for head h in the group i):

$$Attention(Q_h, K_i, V_i) \ = softmax\left(\frac{Q_h x {K_i}^T}{\sqrt{d_k}}\right) \times V_i$$

In there, K_i, V_i are Key and Value of group i.

Multi-Head trong GQA: GQA maintains many heads like MHA, but the number of K and V is reduced by sharing in groups, creating a trade-off between diversity and performance.

⁕ **Computational Complexity:**

- In terms of calculation, GQA still has the same complexity as MHA and MQA, that is $O(n^2)$ with n is the string length.
- However, GQA significantly reduces the cost of memory and cache K, V compared to MHA, especially when G is much smaller than H.
- This helps to speed up the Inference, reduces the latency and saves memory, is important for LLM with large size.

⁕ **Advantage:**

- Quality and performance balance: GQA maintains performance close to MHA (thanks to many K, v) groups while saving memory compared to MHA (by sharing K, V in the group).

- Acceleration of reasoning: Reducing the size K, V helps access memory faster, improves performance on GPU.
- Flexible than MQA: Many groups of K and V allows learning more diverse performances than MQA, improving quality in complex tasks.
- Suitable for large LLMS: GQA is a popular choice in models that need to optimize resources while ensuring high performance.

⁕ **Disadvantage:**

- Like MHA and MQA, GQA does not solve the square complexity of the Attention with the string length.
- Number of group numbers need to be adjusted: The number of groups g needs to be carefully selected to balance the quality and performance. Too few groups (near MQA) reduce quality; Too many groups (near MHA) loses memory benefits.
- Complex deployment: Compared to MQA, GQA requires management of groups K, V, increasing the source complexity.
- Effectiveness depends on hardware: GQA benefits are more pronounced on GPU with long chain and large number of heads.

⁕ **Compare with the original Self-Attention in transformer:**

- **The Similarity:**
  - GQA is based on the original self-off mechanism, using Scaled Dot-Product Attention.
  - Both learn all overlapping relationships between tokens.
- **The Difference:**
  - The number of K, V: the original self-off (in MHA) created K, V separately for each head, while GQA shared K, V in groups, reducing memory.
  - Memory costs: GQA saves memory than MHA, especially with large heads, but still cost more than MQA.

- Performance: GQA usually reaches its efficiency near MHA, surpassing MQA in complex tasks, but it can still be a bit less MHA due to the reduction of the diversity of K, V.
- Application: GQA is optimized for deduction in modern LLMS, While the Self-Attention is more common in the original transformer models.

⁕ **Application in modern LLM models:**

- **LLaMA và LLaMA-2:** GQA is used to improve the deduction performance compared to MHA, while maintaining better performance than MQA.
- **Grok (xAI):** GQA can be integrated in some variants of Grok to optimize performance on natural language tasks.
- **Effective reasoning model:** GQA is popular in models deployed on equipment with limited resources, thanks to the ability to reduce memory while keeping high performance.
- **Combined with Flashattention:** GQA is usually integrated with FlashAttention to optimize more memory and speed, as in modern models.
- **Natural language tasks:** GQA supports tasks such as texting, translating machines, answering questions, and summarizing, with efficiency near MHA but the cost is lower.

1.4 FlashAttention

⁕ **Mechanism of operation and main ideas:**

FlashAttention is an optimized algorithm for Self-Attention (specifically Scaled Dot-Product Attention), designed to reduce memory costs and increase calculation speed on GPU hardware. Instead of calculating and storing the entire Attention matrix size n x n. FlashAttention divide the string into blocks and perform calculations by blocks, take advantage of the SRAM of GPU to reduce memory access (HBM).

Mechanism of operation:

- The input chain is divided into small blocks (block size B). Q, K, V are processed by each block, calculating the Attention only within the block.
- Instead of calculating Softmax on the entire Attention matrix, FlashAttention uses online Softmax techniques, gradually updating Softmax and output values without storing large matrix.
- FlashAttention integrates calculations (matrix QxK^T, Softmax, multiply V) into a single kernel, reducing the number of read/memory recording.
- FlashAttention retains the Self-Attention's mathematical results, but optimizes the implementation to reduce I/O costs.

Formula:

$$Attention(Q,K,V) = softmax\left(\frac{QxK^T}{\sqrt{d_k}}\right) \times V$$

But calculated in each block, with the steps:

- Divide Q, K, V into blocks.
- Calculate QxK^T and softmax for each pair of blocks.
- Accumulate output without storing full Attention matrix.

FlashAttention-2: An improved version, optimizing the work partitioning and reducing memory conflicts, increasing more speed.

⁕ **Computational Complexity:**

- Tradition: Self-Attention has a memory complexity and calculation is $O(n^2)$ with n is the string length, because it must calculate the Attention point for every pair of tokens.
- FlashAttention: Still keeping complexity $O(n^2)$ regarding calculations, but significantly reducing memory costs and execution time thanks to:

    - Do not store the entire Matrix of Intermediate Attention to HBM.
    - Only download and handle each small block, reducing the number of large memory access.
    - Make the most parallel use on GPU.
- Actual results show that FlashAttation-2 can accelerate 2-4 times compared to the optimal Implationation before, and reach 3 times the speed when benchmark on GPU A100/H100.

⁕ **Advantage:**

- Speed on training and Inference: Reducing training time and deduction, especially with long chain (8k, 32k, even 100k tokens)
- Memory saving: allows the processing of longer chains without spilling GPU memory.
- Do not lose accuracy: no approximation, keep the output quality compared to the standard Self-Attention.
- Easy to integrate: have been many frameworks and large models (GPT-3, GPT-4, Claude, MPT, Falcon, Llama ...).

⁕ **Disadvantage:**

- Modern GPU hardware requirements: To make the most, it is necessary to have a large memory and good support for the parallel calculation (Tensor Core).
- Complex deployment: Need to install specially at Kernel GPU, difficult to integrate manually into the old pipeline.
- Effect depends on block size: need to choose the size of the block B is suitable for hardware and string length to optimize.

⁕ **Compare with the original Self-Attention in transformer:**

- **The Similarity:**
    - FlashAttention produces the same mathematical results as the original dot-phoduct attack scaled.

    - Both learn all overlapping relationships between tokens.
- **The Difference:**
    - Memory cost: FlashAttention decreases from $O(n^2)$ near $O(n)$ by not saving the full Attention matrix.
    - Performance: FlashAttention significantly faster on GPU thanks to I/O and Fused Kernel optimization.
    - Deployment: FlashAttention requires custom Kernel, while the original Self-Attention is easy to deploy in frameworks like Pytorch.
    - Application: FlashAttention is designed for modern LLMS with long chain, while the original Self-Attention in original transformer models.

⁕ **Application in modern LLM models:**

- **LLaMA-3:** FlashAttention is integrated to handle long chains and accelerate reasoning/training.
- **GPT-4 and large models:** FlashAttention helps reduce memory costs, allowing longer context processing in tasks such as dialogue or summary.
- **Grok (xAI):** FlashAttention can be used to optimize performance on natural language tasks, especially when it is necessary to handle long chains.
- **Long chain model:** FlashAttention is popular in models such as Longformer, Transformer-XL, or long document processing applications.
- **Natural language tasks:** Support for texting, translating, answering questions, and long-term context requirements, with higher efficiency than the original Self-Attention.

1.5 Linear Attention

⁎ **Mechanism of operation and main ideas:**

Linear Attention is a variant of Self-Attention designed to reduce magnetic calculation complexity $O(n^2)$ down $O(n)$, where n is the string length. Instead of calculating the magic matrix size n x n, Linear Attention uses a Kernel mapping to represent the scalar accumulation between Query (Q) and Key (K) in the form of a linear calculation, helping to handle long chain more effectively.

Mechanism of operation:

- In the original Self-Attention, the Attention matrix is calculated by:

$$Attention(Q,K,V) = softmax\left(\frac{QxK^T}{\sqrt{d_k}}\right) \times V$$

This requires calculation QxK^T (size n × n) and apply Softmax.

- Linear Attention restructures calculation by replacing Softmax with a Kernel function ϕ, allow for permission to change the order of calculation:

$$LinearAttention(Q,K,V) = \phi(\mathrm{Q})\left(\sum_i \phi(K_i)V_i^T\right)$$

In there, ϕ is a mapping function (usually nonlinear function like Elu (x) + 1 or exp (x)). K_i, V_i are vector Key and Value of the token i.

- Key insight: instead of calculating QxK^T first, Linear Attention calculated the sum $\sum_i \phi(K_i)V_i^T$ (a matrix d x d) first, then multiply with $\phi(\mathrm{Q})$. This avoids creating matrix n x n, reduce complexity.
- Multi-Head: Linear Attention can be applied in Multi-Head mode, similar to Self-Attention, but each head uses linear calculations.

⁎ **Computational Complexity:**

- Traditional Attention: $O(n^2d)$, due to the calculation of Attention for every pair of tokens.

- Linear Attention: $O(n^2d)$, because just mapping and accumulating vectors, no need to save the entire Matrix.
- Meaning: When n is much greater than d (usually true to LLM), Linear Attention helps to save significant resources.

⁕ **Advantage:**

- Save memory and accelerate: Allows much longer chain treatment without memory overflow, suitable for actual LLM applications.
- Suitable for inference real-time: Because it only needs to save a fixed size (d × d), very suitable for the text of each token (Autoregressive decoding).
- Easy to integrate: Can be applied to Attention based models with small changes in architecture.

⁕ **Disadvantage:**

- Reducing the quality of representation: Compressing information into a fixed state (d x d) makes it difficult for the model to store the entire context when the chain is too long, which can reduce performance compared to traditional attention.
- Unable to keep all complex relationships: Some information about the relationship between tokens may be lost due to state memory limitations.
- Performance on benchmark: Despite the efficiency of speed and memory, Linear Attention often gives less than Softmax Attention on tasks that need deeply understood context.

⁕ **Compare with the original Self-Attention in transformer:**

- **The Similarity:**
  - Both learn relationships between tokens based on Q, K, V.
  - Can be applied in Multi-Sead mode.
- **The Difference:**

- Complexity: Linear Attention decreases from $O(n^2)$ down $O(n)$, more suitable for long string.
- How to calculate Attention: Self-Attention uses Softmax and matrix n x n, while the Linear Attention uses Kernel and linear calculations.
- Quality: Self-Attention is often superior in the tasks that require accurate performances, while the Linear Attention may lose information due to Kernel Approximation.
- Application: Self-Attention is the standard in Transformer; Linear Attention is more popular in long-chain processing models (such as Performer).

⁕ **Application in modern LLM models:**

- **Performer:** The pioneering model uses Linear Attention, optimized for long string requirements such as handling documents or biological data.
- **Long-context LLMs:** Linear Attention is tested in models such as Longformer or Transformer-XL to handle longer context without significant costs.
- **Combined with other techniques:** Linear Attention can be integrated with Flashattention or GQA to optimize more, although less common in large LLMSs like LLAMA or GPT-4.
- **Long chain application:** Suitable for tasks such as long document summaries, programming code analysis, or real-time data processing.
- **Research:** Linear Attention is still being discovered to improve performance and quality, with the potential to replace Self-Attention in future models.

1.6 Sparse Attention

⁕ **Mechanism of operation and main ideas:**

Sparse Attention is a variant of Self-Attention designed to reduce calculation complexity and magnetic memory $O(n^2)$ down to lower levels (usually close $O(n\log n)$ or $O(n)$) by calculating the Attention for a set of tokens pairs instead of all pairs. This is especially useful for long chains, where the Attention matrix n x n becomes too expensive.

Mechanism of operation:

- In the original Self-Attention, each token pays attention to all the other tokens, creating an Attention matrix size n x n:

$$Attention(Q,K,V) = softmax\left(\frac{QxK^T}{\sqrt{d_k}}\right)\times V$$

  This requires calculation QxK^T (size n × n) and apply Softmax.
- In Sparse Attention, only a few elements in the matrix QxK^T are calculated, based on a sparset pattern. Other elements are set by zero or ignore.
- Common samples:
  - Sliding Window Attention: Each token only pays attention to a window of nearby tokens (For example: w token before and after).
  - Dilated Window Attention: Similar to Sliding Window, but with dance (dilation) to expand the scope of attention without increasing costs.
  - Global Attention: Some tokens (such as the first token or special token) pay attention to all tokens, and vice versa.
  - Random Attention: Randomly select some tokens to pay attention, reduce costs but may lose information.
- Deploying: Attention matrix is mask to refer to the tokens in the sample, or use specialized algorithms to ignore unnecessary calculations.

- Multi-Head: Sparse Attention is often applied in Multi-Head mode, with each head that can use different samples to increase diversity.

⁕ **Computational Complexity:**

- Traditional Self-Attention: $O(n^2)$, with n is the string length, because must be calculated for every pair of tokens.
- Sparse Attention: The complexity decreases to $O(n\ x\ k)$, with k is the number of Attention connections each token (k << n). For example, with a chain of 1,000 tokens, Self-Attention calculates 1 million pairs, Sparse Attention only needs 100,000 pairs if k = 100, 90% discount on calculation costs and memory.
- Double Sparset: Even reducing costs by keeping only the largest Attention Score after the connection sparse.

⁕ **Advantage:**

- Memory saving and acceleration: sharply reducing memory costs and calculation time, allowing a much longer chain processing than traditional Self-Attention.
- Focus on important information: The model learns how to pay attention to important tokens, increase efficiency for tasks such as text summary, long-term analysis, large documents processing.
- Expansion: Allows LLMS to handle the chain tens of thousands of tokens, suitable for large-scale practical applications.

⁕ **Disadvantage:**

- The information may be missed: If the Pattern Sparse is not suitable, the model can ignore the important relationships between the tokens.
- Need to tweak Pattern: Choosing or learning Pattern Sparse is suitable for each task is a challenge.
- It is difficult to optimize the simultaneously: If the Pattern Sparse is the same for all heads, the model may be limited in the ability to perform; If different, increase the implementation complexity.

⁕ **Compare with the original Self-Attention in transformer:**

- **The Similarity:**
  - Both are based on Q, K, V and Scaled Dot-Product Attention.
  - Can be applied in Multi-Sead mode.
- **The Difference:**
  - Complexity: Sparse Attention decreases from $O(n^2)$ down $O(n\ x\ w)$, more suitable for long string.
  - Scope of attention: Self-Attention pays attention to all tokens; Sparse Attention only pays attention to a subset of a sparse model.
  - Quality: Self-Attention is often superior in the tasks that require all overall contexts, while Sparse Attention prioritizes effectively.
  - Application: Sparse Attention is popular in long-chain processing models (such as Longformer, Bigbird), while Self-Attention is a standard in Transformer.

⁕ **Application in modern LLM models:**

- **Longformer:** Use Sliding Window Attention Slings combine Global Attention to process long documents, such as summary or text analysis.
- **BigBird:** Combining Slings Window, Global Attention, and Random Attention to achieve high efficiency with long chains, used in tasks such as QA and classified.
- **Transformer-XL:** Use the same mechanism as Sparse Attention to handle longer context through repeat memory.
- **Long chain:** Sparse Attention is suitable for tasks such as processing code, long documents, or real-time data.
- **Combined with other techniques:** It is integrated with FlashAttention to optimize more on GPU, or with GQA/MQA to reduce costs in large LLMS.

- **Research:** Sparse Attention is being discovered to replace Self-Attention in future models, especially in high-performance requirements.

1.7 Rotary Positional Embedding (RoPE)

⁕ **Mechanism of operation and main ideas:**

Rotary Positional Embedding (ROPE) is a location (Positional encoding) designed to integrate Token's location information into the Self-Attention mechanism effectively. Instead of adding the input vector like a static Positional Encoding in the original transformer, RoPE applies a rotation to the query (Q) and key vector (k) based on the token position, helping the model learn relatively relative location relationships.

Mechanism of operation:

- RoPE encodes the position of token **m** with a rotating matrix R_m, defined based on the rotation angle depends on the location **m** and wave frequency (frequences).
- Each vector Q and K (size d_k) is divided into pairs of dimensions, and rotating matrix R_m is applied to rotate these pairs:

$$R_m = \begin{bmatrix} \cos(m\theta_1) & -\sin(m\theta_1) & 0 & 0 & \cdots \\ \sin(m\theta_1) & \cos(m\theta_1) & 0 & 0 & \cdots \\ 0 & 0 & \cos(m\theta_2) & -\sin(m\theta_2) & \cdots \\ 0 & 0 & \sin(m\theta_2) & \cos(m\theta_2) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

  In there, $\theta_i = 10000^{\frac{-2i}{d_k}}$ is the wave frequency, gradually decreasing in the direction i.

- Vector Q_m and K_m at location **m** rotate:

$$Q'_m = R_m Q_m$$
$$K'_m = R_m K_m$$

- The scalar in the Self-Attention becomes:

$$(Q'_m)^T K'_n = (R_m Q_m)^T (R_n K_n) = Q_m^T R_m^T R_n K_n$$

  Matrix $R_m^T R_n$ relative distance encoding m - n, helping the model to learn location relationships without adding a separate location vector.
- Multi-Head: RoPE is applied independently to each head in the Multi-Head Attention, ensuring consistency between heads.

⁕ **Outstanding features of RoPE:**

- **Relatively natural placement**: When calculating dot-prooduct between query at location m and key at location n, the result is equivalent to the application of a relatively dependent rotation n - m, helping the model to learn relative location relationship without adding separate embedding class.
- **Continuous and stable**: RoPE maintains the smoothness in the embedding space when changing the token position, making the generalized model better with longer chains than the training length.
- **Good support for KV cache in Inference:** Different from traditional relative embedding, RoPE allows effective storage and reuse cache key and value in the process of producing Autoregressive text.
- **Easy to integrate:** RoPE only requires changes Q and K before calculating the Attention, no additional parameters or complex classes.

⁕ **Computational Complexity:**

- Rope does not increase the calculation complexity of Self-Attention (still $O(n^2)$ with n is the string length).
- The rotation is done in pairs of Embedding directions, which can be effective in parallel on the GPU.
- Therefore, RoPE is an effective way to embed position, without clogging.

⁕ **Advantage:**

- Combining both the advantages of Absolute and Relative Positional Embedding.
- Helps the model of relatively intuitive and effective position.
- Keep stability and overall ability with long chains.
- Increasing Inference performance thanks to good KV cache support.
- Successfully applied in many modern LLM such as LLAMA, GPT-4, Mistral, Phi-2 ...

⁕ **Disadvantage:**

- A little more complex than the traditional Absolute Positional Embedding.
- It is necessary to calculate the rotation for each position and each pair of dimensions, but this cost is not significant compared to the entire model.
- Some of the deeper mathematical parts of RoPE can be confusing and need to refer to the original article to understand.

⁕ **Compare with the original Self-Attention in transformer:**

- **The Similarity:**
  - RoPE is integrated into Self-Attention, without changing the core mechanism of Scaled Dot-Product Attention.
  - Both use Q, K, V to learn the relationship between tokens.
- **The Difference:**
  - Location encryption: Original Self-Attention using static Positional Encoding (Sinusoidal or learned) added to the input, while the RoPE applies directly to rotate onto Q and K.
  - Location properties: RoPE encodes relative location, while Positional encoding encodes the position absolutely.
  - Long-string efficiency: The generalized RoPE is better for the long chain thanks to the relative properties, while the static

Positional encoding may have difficulty with the chain exceeding the training length.

- Deploying: RoPE requires amendments to calculation Q, K, while Positional Encoding only needs to add to the input embeding.

⁕ **Application in modern LLM models:**

- **LLaMA và LLaMA-2:** RoPE is used as the main location encryption method, helping to improve performance on long-standing tasks, such as texting and answering questions.
- **Grok (xAI):** Some of Grok variants can use RoPE to take advantage of the overall ability for long chains.
- **Mistral and Mixtral:** Use RoPE to support long chain processing and improve the effectiveness of reasoning.
- **Long chain task:** RoPE is suitable for tasks such as long document summaries, programming code analysis, or dialogue with long context.
- **Combined with other techniques:** RoPE is usually integrated with FlashAttention, GQA, or MQA to optimize performance in large LLMS.

CHAPTER 2. OPTICAL CHARACTER RECOGNITION

2.1 Introduction and Theoretical Basis

Optical Character Recognition (OCR) is a fundamental technology in computer vision aimed at converting differrnce types of documents, such as scanned paper documents, PDFs, or images taken by a camera, into editable and searchable data. It plays a vital role in various applications, including digitizing historical archives, autotmating data entry from invoices and forms, extracting information from signage and analyzing content in digital media. Traditionally, the OCR task is divided into two main sequential stages: Text Detection and Text Recognition. Text Detection focuses on locating the bounding boxes or polygons around text regions within image. Text Recognition then takes these detected regions as input and translate the visual characters into a machine-readable text format.

The core of our OCR system for text recognition utilizes a modern Encoder-Decoder framework. This architecture has preoven highly effective in various sequence to sequence tasks, including machine translation and image captioning, making it well suited for converting a sequence of image features into a sequence of text tokens. The role of the Encoder is to precess the input image and generate a compact, high-level feature representation. For OCR, the CNN Encoder extracts visual features that are relevant for identifying characters and words. The Decoder is tasked with generating the output text sequence based on the visual features provided by the Encoder. We use a Transformer Decoder for its powerful sequence modeling capabilities. Our Transformer Decoder takes the partially generated text sequence as input. It utilizing token embeddings to convert discrete token indices into continuous vector representations. Positional Encoding is added to these embeddings to inject information about the tokens' positions within the sequence, compensating for the permutaiton- invariant nature of attention.

The Attention mechanism is central to how the Decoder effectively uses the Encoders output. Within the Transformer Deocder, two key attentoin type are employed:

- Masked Self-Attention: Applied within the Decoder, allowing it to weigh the importance of previously generated tokens when predicting the next one. The “mask” aspect ensures that prediction for a token at a given position only depends on the tokens that precede it in the sequence, which is essential for autoregressive sequence generation.
- Cross-Attention: This mechanism connects the Decoder to the Encoder’s output. At each step of text generation, the Decoder queries the Encder’s visual features. This allows the Decoder to focus its attention on the most relevant parts of the image feature sequence when determining which character to output next.

2.2 Implement Details

2.2.1 Dataset and preprocessing

The project utilizes the MC-OCR dataset. The raw annotation data provided in a CSV file, contains information about image IDs, raw polygon strings and concatenated text strings for all text instances within each image.

The initial preprocessing step involves loading this raw CSV and carefully parsing the *anno_polygons* and *anno_texts* columns. Since *anno_polygons* are a string representation of a list of dictionaries and *anno_texts* is a concatenated string separated by '|||', we use *ast.literal_eval* and string splitting to extract individual Ground Truth (GT) polygon-text pairs. Instances with invalid or empty GT text or malformed polygon data are filtered out. This process generates a processed DataFrame (*df_processed_gt*) where each row represents a single GT text instance with its corresponding image ID, GT polygon points (as a list of [x, y] pairs), and cleaned GT text.

This df_processed_gt DataFrame is then split into training, validation, and test sets (80%, 10%, 10% respectively) based on the number of instances. This ensures that the Recognition model is trained and evaluated on distinct subsets of the Ground Truth data.

The OCRDataset class is designed to load and preprocess individual samples for the Recognition model training. For each GT instance (a row in the split DataFrames), it loads the original full image, calculates the bounding box tightly enclosing the GT polygon, crops this region from the grayscale image, and resizes it to a fixed size (IMG_HEIGHT x IMG_WIDTH) using bilinear interpolation. The cropped image is then converted to a PyTorch tensor and normalized. The corresponding GT text is tokenized into a sequence of indices, including [SOS] and [EOS] tokens, and padded to a fixed MAX_LEN using the [PAD] token. The collate_fn function is used in the DataLoader to batch these processed image tensors and token sequences, handling padding and creating appropriate masks for the Transformer Decoder.

2.2.2 Vocaburary Building

The vocabulary for the Recognition model is constructed from the set of all unique characters found in the Ground Truth text instances across the *entire df_processed_gt*. This ensures the model can generate any character present in the dataset. Special tokens ([SOS], [EOS], [PAD], [UNK]) are added to the vocabulary. Dictionaries mapping characters to indices (*char_to_idx*) and indices back to characters (*idx_to_char*) are created, and the total vocabulary size (VOCAB_SIZE) is determined.

2.2.3 Model Architecture

The Recognition model is an OCRModel comprising the EncoderCNN and DecoderTransformer, as detailed in the theoretical basis. The EncoderCNN processes the input image region (after cropping and resizing) into a sequential visual feature representation. The DecoderTransformer takes this feature sequence and the shifted

Ground Truth text sequence as input to predict the probability distribution of the next token. Positional encoding is applied to both the image features and the text token embeddings. The model is instantiated and moved to the specified DEVICE (GPU or CPU).

2.2.4 Traning Setup

The Recognition model is trained using the following setup:

- **Loss Function**: CrossEntropyLoss is used to measure the discrepancy between the predicted token probabilities and the actual target token indices. The ignore_index=PAD_TOKEN ensures that padding tokens do not contribute to the loss calculation.
- **Optimizer**: The optim.AdamW optimizer is employed with a learning rate of LEARNING_RATE (1e-4) and a weight decay of 0.01. AdamW is chosen for its effectiveness in training deep learning models.
- **Training Loop**: The train epoch function handles one epoch of training. It sets the model to training mode, processes batches from the training DataLoader, performs the forward pass, calculates the loss, backpropagates gradients, and updates model weights using the optimizer. Gradient clipping *(torch.nn.utils.clip_grad_norm_)* is applied to mitigate potential gradient exploding issues.
- **Evaluation**: The evaluate recognition function assesses the model's performance on the validation set after each training epoch. It sets the model to evaluation mode, disables gradient computation, and calculates the average loss, Character Error Rate (CER), and Sequence Accuracy. Greedy decoding is performed within this function to generate predicted text sequences for CER and Accuracy calculations.

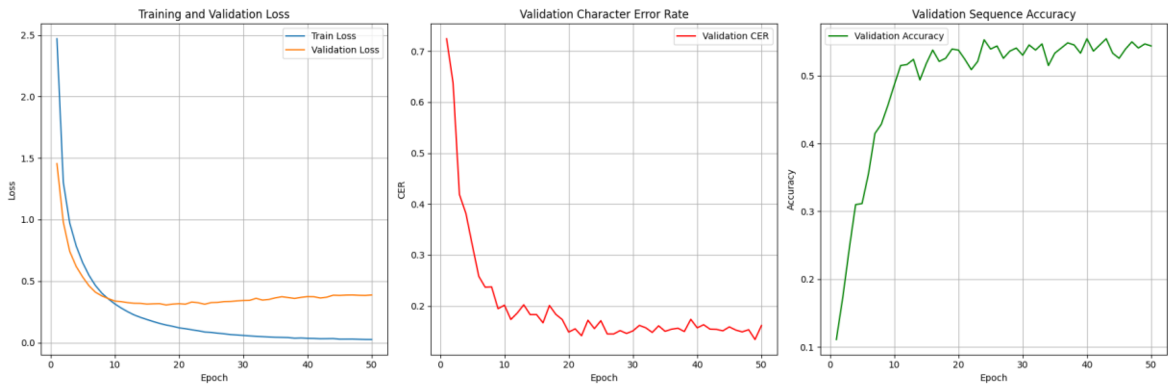
2.2.5 Intergration with PaddleOCR Detection

For the End-to-End OCR pipeline, we integrate our custom Recognition model with the PaddleOCR library. The PaddleOCR component is initialized separately using PaddleOCR*(lang='vi', det=True, rec=False, use_angle_cls=False*). The *det=True* parameter enables its text detection capability, while *rec=False* disables its built-in recognition module, as we will be using our own trained model for recognition. This initialized ocr_detector object is used in the inference phase to take a full input image and output a list of detected text region polygons.

2.3 Results, Evaluation and Inference

2.3.1 Recognition Model Training Process

The Recognition model was trained on the Ground Truth training set for EPOCHS (50) epochs. The training process was monitored by tracking the loss on both the training and validation sets, as well as the Character Error Rate (CER) and Sequence Accuracy on the validation set after each epoch. The model achieving the lowest validation CER during training was saved as the best model.



Plots of train and val Loss, CER, and Validation Accuracy over epochs.

As show in the plots above, the training loss go down over the epochs. The validation loss also decreased, after this initial rapid decrease the validation loss fluctuating between approximately 0.3 and 0.4 for the remainder of the training duration. The Validation CER start from a high value and experienceda sharp decline in the first 10 epochs, the lowest validation CER achieved appears to be around 0.14 to 0.15. The valition Sequence Accuracy followed an inverse trend to the CER, start from a low point around 0.1. It rapidly increased during the early epochs, then fluctuated between 0.5 and 0.55, reflecting the model’s improving ability to transcribe entire text lines accurately.

2.3.2 Recognition Model Evaluation

To quantitatively assess the performance of our trained Recognition model independently, we evaluated the best saved model on a separate Ground Truth test set *(test_loader_rec*). This set consists of image regions cropped based on accurate Ground Truth polygons from images that were not used during either training or validation. This evaluation specifically measures how well the model transcribes text when provided with ideal inputs.

The evaluation metrics on the Test Set are as follows:

- Test Loss: **0.4042**
- Test CER: **0.1502**
- Test Accuracy: **0.5903**

The achieved Test CER of 0.1502 indicates that, on average across the test instance the recognition model make 15-character errors per 100 characters in the Ground Truth text. The Test Accuracy of 0.5903 mean that 59% of the Ground Truth text instances in the test set were predicted exactly correctly by the model. These results demostrate the model’s string ability to recognize Vietnamese text from isolated and correctly cropped regions.

2.4 Discussion and Conclusion

The training process showed clear learning, with both training and validation loss decreasing significantly, and validation CER decreasing while accuracy increased. The validation metrics reached a stable state after approximately 15-20 epochs, suggesting that the model had converged sufficiently on the validation data. The best Validation CER achieved was around 0.15, indicating a reasonable level of character-level accuracy on unseen, but perfectly cropped, validation text instances.

The quantitative evaluation on the separate Ground Truth test set confirmed the model's recognition capabilities. A Test CER of 0.1505 and Test Accuracy of

0.5903 demonstrate that our trained model can accurately transcribe a significant portion of text regions when provided with precise bounding box information derived from Ground Truth. These results align well with the validation performance, suggesting good generalization to unseen Ground Truth instances.

The End-to-End inference pipeline qualitatively showcased the system's ability to operate on full images. By visually inspecting the output on sample test images and the user-provided image, we could observe the performance of both the detection and recognition stages. PaddleOCR demonstrated its effectiveness in locating text regions on the tested images, identifying most prominent text lines. Our Recognition model then successfully transcribed the text within many of these detected regions.

In conclusion, this project successfully implemented and demostrated a core image to text recognition system using a modern CNN-Transformer architecture trained on data. By integrating it with a pre-trained text detector, a functional End-to-End OCR pipeline was realized, capable of processing full images and visualizing the detected text regions alongside their predicted content. The quantitative results on the recognition test set highlight the model’s capability, while the End-to-End demostration provides a practical application, paving the way for furthers enhancements to address the identified limitations.

REFERENCES

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. (2 Aug 2023). *Attention Is All You Need.*

Bais, G. (24th April, 2025). *Building Deep Learning-Based OCR Model: Lessons Learned*. Được truy lục từ https://neptune.ai/blog/building-deep-learning-based-ocr-model

Doanh C. Bui; Dung Truong; Nguyen D. Vo; Khang Nguyen. (19-21 August 2021). *MC-OCR Challenge 2021: Deep Learning Approach for Vietnamese Receipts OCR.* IEEE.

ironscales. (không ngày tháng). Được truy lục từ What is OCR Deep Learning?: https://ironscales.com/glossary/ocr-deep-learning

Manh, B. Q. (2023, 7). *viblo*. Được truy lục từ https://viblo.asia/p/bai-toan-phat-hien-chu-text-detection-va-mo-hinh-db-phan-2-L4x5xqLOKBM

Naminas, K. (Published May 25, 2023). Được truy lục từ OCR Deep Learning: The Curious Machine Learning Case: https://labelyourdata.com/articles/ocr-with-deep-learning

Nguyen, Hoai Viet and Bao Doan, Linh and Trinh, Hoang Viet and Huy Phan, Hoang and Thanh, Ta Minh . (2021). *http://eprints.lqdtu.edu.vn/id/eprint/10294/*. Được truy lục từ http://eprints.lqdtu.edu.vn/id/eprint/10294/

Xuan Son Vu, Quang Anh Bui, Nhu-Van NGUYEN, Hai Thi Tuyet Nguyen . (không ngày tháng). *Receipts, Mobile-Captured Image Document Recognition for Vietnamese.* Được truy lục từ https://www.rivf2021-mc-ocr.vietnlp.com/

Xuan-Son Vu; Quang-Anh Bui; Nhu-Van Nguyen; Thi Tuyet Hai Nguyen; Thanh Vu. (19-21 August 2021). *MC-OCR Challenge: Mobile-Captured Image Document Recognition for Vietnamese Receipts.* IEEE.