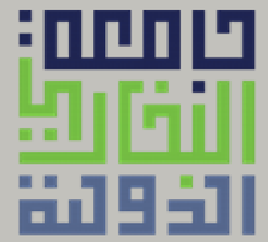
# AI-powered Research Paper Summarizer for Academic Use

## Utilizing the Pegasus Model with ArXive Dataset



.BUKHARY INTERNATIONAL UNIVERSIT

Project Presentation

Md Farman Ali
TAWFIG TAIB YASSEN
S M Asiful Islam Saky
Habib Marzooq Mohamadu

# Introduction

Today, so many research papers are published every day. This makes it challenging for students, teachers, and researchers to keep up and fully understand. Reading long and often complex papers takes a great deal of time and effort.

Our solution is an AI-powered summarization tool that uses Natural Language Processing (NLP) to automatically read and generate clear summaries of academic papers .

# Problem Statement

. Researchers struggle to quickly find the main ideas .

. Most tools don’t give summaries for each section like abstract, methods, or results.

• Need for an intelligent, domain-aware summarization system.

# Research Objectives

1.To build an AI tool that summarizes research papers into short, clear summaries.

2. To develop a tool that summarizes key sections of research papers, including the Introduction, Methodology, Results, and Conclusions.

3. To help researchers access these summaries easily to save time and work more efficiently.

# Literature Review

# Comparison Between Existing Models and Proposed Model

| **Criteria** | **Existing Models (e.g., BERT, GPT-3, T5)** | **Proposed Model (Pegasus-based for Scientific Papers)** |
|---|---|---|
| Summarization Type | Primarily abstractive; some support hybrid approaches (e.g., Dou et al., 2021) | Abstractive (section-wise) |
| Architecture | Transformer-based (BERT, GPT-3, T5) (e.g., Raffel et al., 2020; Lewis et al., 2020) | Transformer-based (Fine-tuned Pegasus model) |
| Domain Specialization | General-purpose; often struggle with technical or domain-specific language ( Shaib et al., 2023) | Specialized in scientific literature, especially ML and NLP papers |
| Section-wise Summary Generation | Rarely implemented; summaries are typically full-text based (e.g., Gidiotis & Tsoumakas, 2020) | Supports targeted summarization (e.g., Introduction, Methodology, etc.) |
| Context Understanding | High for general context, lower in technical or structured texts (e.g., Supriyono et al., 2024) | Tuned for deeper scientific context understanding |
| Real-world Use Cases | News summarization, chat summarization, medical reports (e.g., Shaib et al., 2023; Dou et al., 2021) | Academic research assistance, literature reviews, automated paper digestion |

# Proposed Methodology

# Dataset Description

- Sourced from ArXiv (neuralwork/arxiver dataset).
- Focused on papers from Machine Learning and NLP fields.
- Includes structured markdown and abstracts.
- Ideal for supervised training using section-wise summarization.


```
"author" : string "[{'name': 'Ahmed Osman'}, {'name': 'Wojciech Samek'}]"
"day" : int 1
"id" : string "1802.00209v1"
"link" :
string "[{'rel': 'alternate', 'href': 'http://arxiv.org/abs/1802.00209v1', 'type': 'te
{'rel': 'related', 'href': 'http://arxiv.org/pdf/1802.00209v1', 'type': 'application
'pdf'}]"
"month" : int 2
"summary" :
string "We propose an architecture for VQA which utilizes recurrent layers to generate
textual attention. The memory characteristic of the proposed recurrent attention uni
joint embedding of visual and textual features and enables the model to reason relat
several parts of the image and question. Our single model outperforms the first plac
VQA 1.0 dataset, performs within margin to the current state-of-the-art ensemble mod
experiment with replacing attention mechanisms in other state-of-the-art models with
implementation and show increased accuracy. In both cases, our recurrent attention m
performance in tasks requiring sequential or relational reasoning on the VQA dataset
"tag" :
string "[{'term': 'cs.AI', 'scheme': 'http://arxiv.org/schemas/atom', 'label': None},
'scheme': 'http://arxiv.org/schemas/atom', 'label': None}, {'term': 'cs.CV', 'scheme
'http://arxiv.org/schemas/atom', 'label': None}, {'term': 'cs.NE', 'scheme':
'http://arxiv.org/schemas/atom', 'label': None}, {'term': 'stat.ML', 'scheme':
'http://arxiv.org/schemas/atom', 'label': None}]"
```

00

# Problem Definition

- Dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, where:
  - $x_i$: Markdown-formatted body text of academic paper $i$
  - $y_i$: Human-written abstract for paper $i$
- Pretrained summarization model $M \leftarrow \text{Pegasus-large}$
- Tokenizer $T \leftarrow \text{PegasusTokenizer}$

01

# Preprocessing

**For each** paper $d_i \in D$:

1.1 Extract key sections:

$S_i \leftarrow \text{extract}(d_i, \text{sections} = \{\text{Intro, Method, Results, Conclusion}\})$

1.2 Retrieve the reference abstract $T_i \leftarrow \text{get\_abstract}(d_i)$

1.3 Tokenize input and target:

$x_i \leftarrow \text{tokenize}(S_i, \text{max\_len} = 512)$

$y_i \leftarrow \text{tokenize}(T_i, \text{max\_len} = 128)$

1.4 Apply padding and attention masks

02

## Model Initialization

2.1 Load Pegasus model $f_{\text{pegasus}}(\cdot;\theta)$ with pre-trained weights

2.2 Set training parameters: learning rate, batch size, number of epochs

03

## Training

**For** each epoch:

**For** each batch $(x_b, y_b)$:

3.1 Generate predicted summary $\hat{y}_b = f_{\text{pegasus}}(x_b;\theta)$

3.2 Compute loss: $\mathcal{L}_{\text{CE}} = \text{CrossEntropy}(y_b, \hat{y}_b)$

3.3 Update model parameters:

$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_{\text{CE}}$

04

## Evaluation

**For** each test document $d_j$:

4.1 Generate summary $\hat{y}_j = f_{\text{pegasus}}(x_j; \theta)$

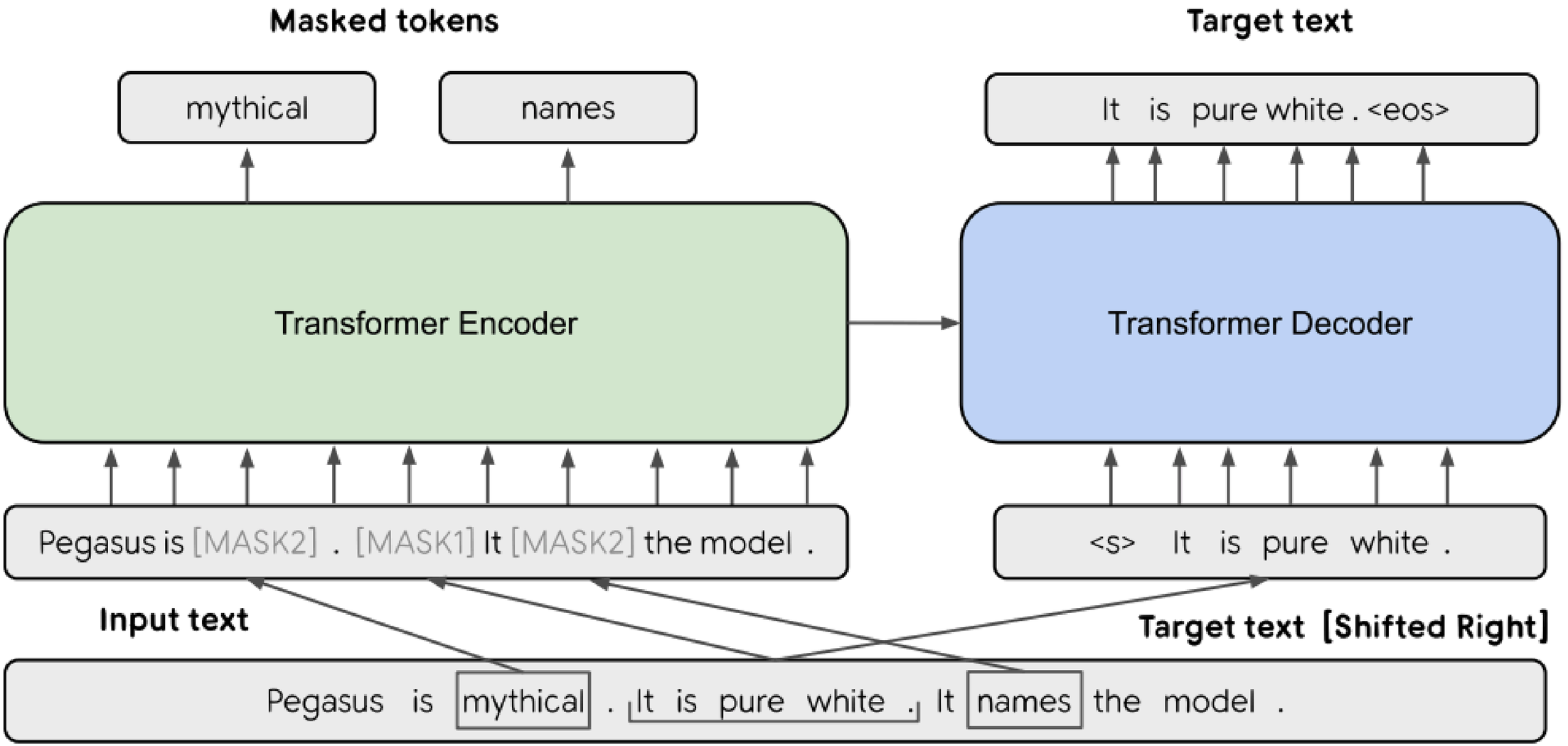
4.2 Compute:

- ROUGE Score: $\text{ROUGE}(\hat{y}_j, y_j)$
- BLEU Score: $\text{BLEU}(\hat{y}_j, y_j)$
- BERTScore: $\text{BERTScore}(\hat{y}_j, y_j)$

## Output:

- **Trained Pegasus model fine-tuned for scientific paper summarization**
- **Performance metrics: ROUGE-1, ROUGE-2, BLEU, BERTScore-F1**

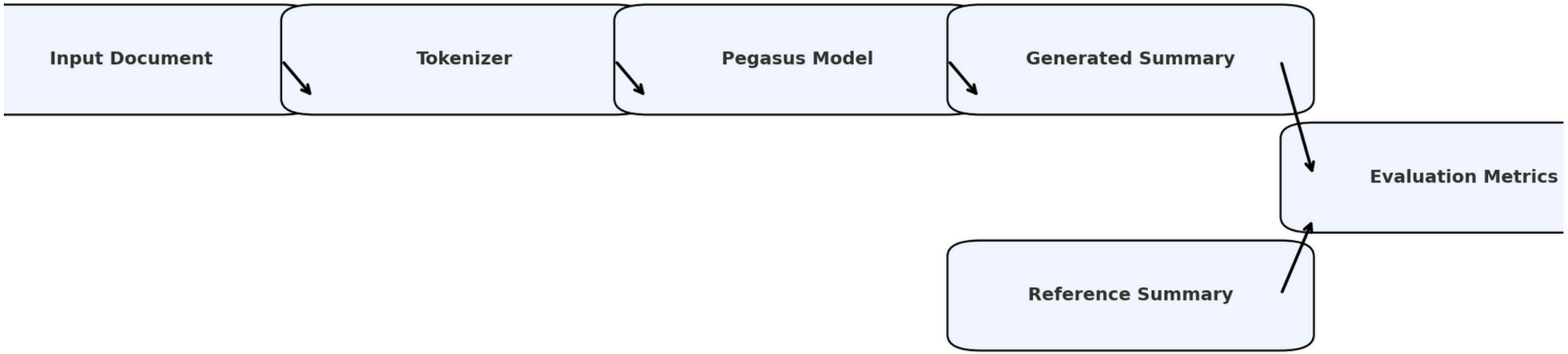
# PEGASUS MODEL ARCHITECTURE



# Evaluation Metrics

Evaluating Summarization Quality

- **ROUGE - Measures n-gram overlaps (recall-focused).**
- **BLEU - Measures precision of n-grams (precision-focused).**
- **BERTScore - Measures semantic similarity using embeddings.**
- **Combined metrics give robust quality analysis.**



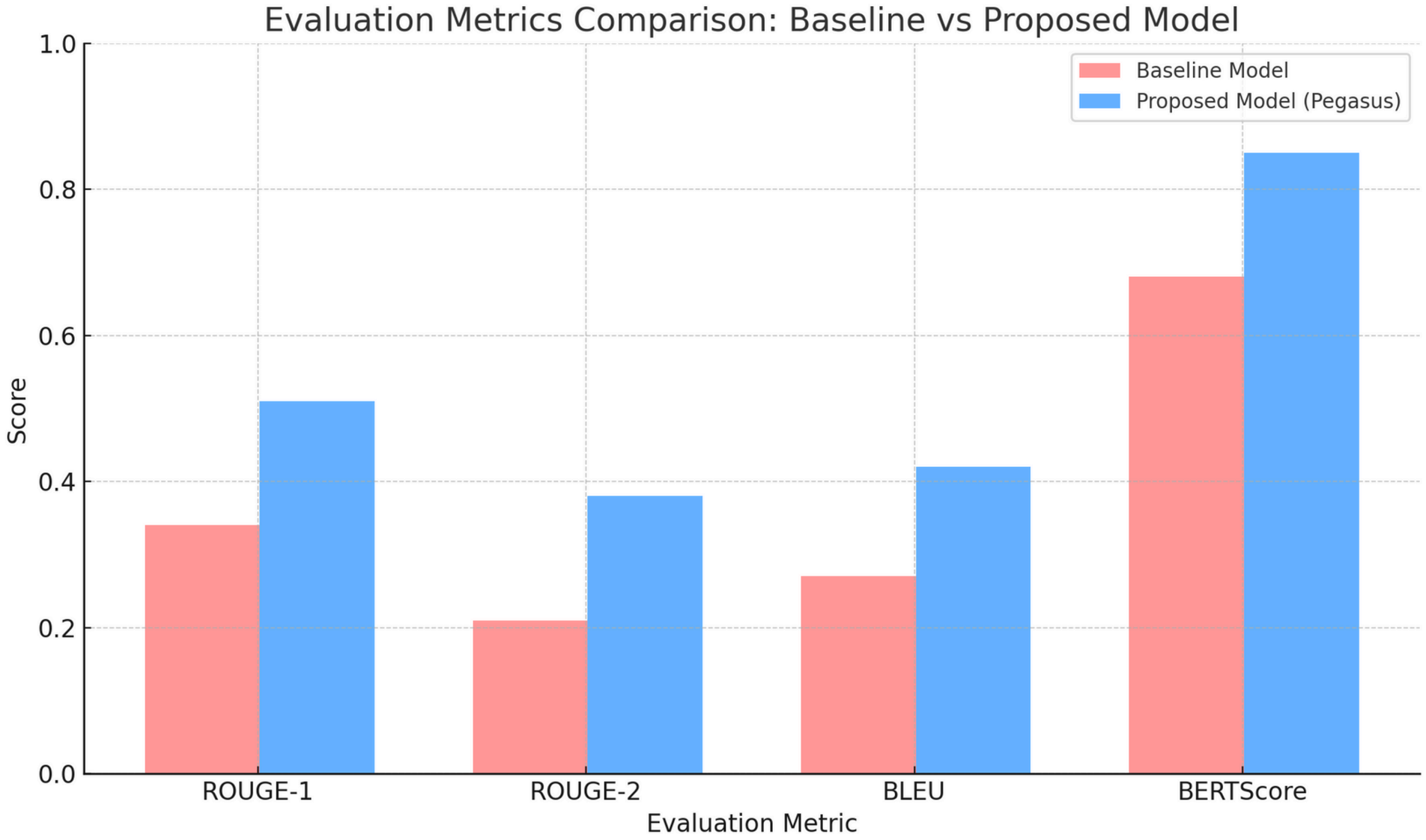
# 1 Result

```
    computation [32], while also improving model performance in case of the l
al
    constraint of sequential computation, however, remains.
    Attention mechanisms have become an integral part of compelling sequence
ction models in various tasks, allowing modeling of dependencies without rega
in
    the input or output sequences [2, 19]. In all but a few cases [27], howev
echanisms
    are used in conjunction with a recurrent network.
    In this work we propose the Transformer, a model architecture eschewing r
d
    relying entirely on an attention mechanism to draw global dependencies be
ut.
    The Transformer allows for significantly more parallelization and can rea
e art in
    translation quality after being trained for as little as twelve hours on
```

## Main Text

Generated Summary for 'Attention is All You Need': Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly establishe d as state of the art approaches in sequence modeling and transduction problems such as langua ge modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15]. Re current models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hi dden states ht, as a function of the previous hidden state ht

# Evaluation


Evaluation Metrics Comparison: Baseline vs Proposed Model
Baseline Model
Proposed Model (Pegasus)
Score
1.0
0.8
0.6
0.4
0.2
0.0
ROUGE-1
ROUGE-2
BLEU
BERTScore
Evaluation Metric

# Conclusion

Summary

- Effective use of Pegasus model for scientific summarization.
- Demonstrated performance gains over baseline models.
- •Suitable for aiding researchers in literature reviews.

Future work:

- Section-wise fine-tuning
- multi-document input
- GUI integration.

# References


1. Ahmad, M., Afzal, H., & Latif, S. (2021). Deep learning approaches for automatic text summarization: A survey. IEEE Access, 9, 16036–16051. https://doi.org/10.1109/ACCESS.2021.3052743
2. Chen, Y., & Bansal, M. (2021). Improving abstractive summarization with sentence-level content planning. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 646–658. https://doi.org/10.18653/v1/2021.naacl-main.52
3. Dou, Z., Liu, P., Li, W., Sun, M., & Niu, Y. (2021). GSum: A general framework for guided abstractive summarization. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 4830–4842. https://doi.org/10.18653/v1/2021.naacl-main.385
4. Giarelis, N., Mastrokostas, C., & Karacapilidis, N. (2023). Abstractive vs. extractive summarization: An experimental review. Applied Sciences, 13(13), 7620. https://doi.org/10.3390/app13137620
5. Gidiotis, A., & Tsoumakas, G. (2020). A divide-and-conquer approach to the summarization of long documents. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 28, 3029–3040. https://doi.org/10.1109/TASLP.2020.3011774
6. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., & Levy, O. (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703
7. Liu, Y., & Lapata, M. (2022). Multi-document summarization with BERT and pointer-generator networks. Transactions of the Association for Computational Linguistics, 10, 161–177. https://doi.org/10.1162/tacl_a_00428
8. Muia, C., & Ndung, R. (2024). A comparative study of transformer-based models for text summarization of news articles. Academia.edu. https://www.academia.edu/117474569
9. Ramesh, G. S., Manyam, N. V., Mandula, N. V., Myana, N. P., Macha, N. S., & Reddy, N. S. (2022). Abstractive text summarization using T5 architecture. In S. K. Bhatia et al. (Eds.), Algorithms for intelligent systems (pp. 535–543). Springer. https://doi.org/10.1007/978-981-16-7389-4_52

# Appendix

```
# Disable WANDB logs and other non-essential settings
os.environ["WANDB_DISABLED"] = "true"

# Load the original model and tokenizer (no additional layers)
model_name = "google/pegasus-large"
tokenizer = PegasusTokenizer.from_pretrained(model_name)
model = PegasusForConditionalGeneration.from_pretrained(model_name)

# Set device to GPU if available, otherwise CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Load a small subset of the dataset to test stability, you can try with 1000 samples
dataset = load_dataset("neuralwork/arxiver", split="train[:2000]")

# Preprocessing function
def preprocess_function(examples):
    inputs = tokenizer(examples['markdown'], max_length=512, truncation=True,
padding="max_length")
    targets = tokenizer(examples['abstract'], max_length=128, truncation=True,
padding="max_length")
    return {
        "input_ids": inputs["input_ids"],
        "attention_mask": inputs["attention_mask"],
        "labels": targets["input_ids"]
    }

# Preprocessing the dataset
tokenized_dataset = dataset.map(preprocess_function, batched=True)
```

```
# Setting training arguments
training_args = Seq2SeqTrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=1,  # Small batch size
    per_device_eval_batch_size=1,
    weight_decay=0.01,
    num_train_epochs=1,
    predict_with_generate=True,
    fp16=False,  # Disable mixed precision for stability
    dataloader_num_workers=0,
    report_to="none"
)

# Initializing Trainer
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    eval_dataset=tokenized_dataset,
    tokenizer=tokenizer
)

# Training the model
trainer.train()

# Saving the model after training
model.save_pretrained("./saved_model")
```

# Thank You!

# Any Question?