

Enhanced Pegasus Architecture for News Summarization

Overview: We build on the standard PEGASUS encoder–decoder transformer and augment it with five advanced components for richer understanding and generation. The base model remains a multi-layer Transformer (stacked self-attention and feed-forward blocks) pre-trained with the *gap-sentence generation* objective. We then integrate hierarchical attention, external memory access, advanced positional encodings, and global context modules. The result is a deep architecture optimized for news domains (long, multi-topic articles) with heavy emphasis on **training-time effectiveness** (multi-task pretraining, complex attentions) rather than minimal inference cost.

Paper link -

https://homepage.divms.uiowa.edu/~jruser/asad_1113.pdf#:~:text=new%20self%02supervised%20objective,all%2012%20downstream%20datasets%20measured

Pegasus Backbone and Pretraining

1. Transformer Encoder–Decoder Backbone

1.1. Token Embedding + Positional Encoding

- **Token Embeddings:** Each input token is mapped to a learned embedding vector of dimension d (often 1024 or 768).
- **Positional Encodings:** To give the model a sense of word order, PEGASUS adds *absolute sinusoidal* positional encodings to these embeddings. In our enhanced model we'll swap these for *rotary/relative* encodings, but the original PEGASUS uses standard add-and-normalize layers.

1.2. Encoder Layers

Each of the L encoder layers (typically $L = 12$ or 16) consists of:

1. **Multi-Head Self-Attention**
 - Splits embeddings into H heads; each head computes $\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k}) V$
 - Enables tokens to attend to all other tokens in the same layer.
2. **Add & LayerNorm**
 - Residual connection around the attention block, followed by layer normalization.
3. **Position-Wise Feed-Forward**

- Two linear transforms with a GeLU nonlinearity in between:

$$\text{FFN}(x) = W_2 \text{GeLU}(W_1 x + b_1) + b_2$$

$$\text{FFN}(x) = W_2 \text{GeLU}(W_1 x + b_1) + b_2$$
- 4. **Add & LayerNorm**
 - Another residual + normalization around the FFN.

Stacking these yields deep contextual representations. All encoder layers share the same dimension d and head count H ; their weights are independent.

1.3. Decoder Layers

The decoder also has L layers, each containing:

1. **Masked Multi-Head Self-Attention**
 - Like the encoder's self-attention, but causally masked so each position can only attend to previous positions (ensuring autoregressive generation).
2. **Add & LayerNorm**
3. **Encoder–Decoder Cross-Attention**
 - Queries come from the decoder's current states; keys/values come from the encoder's final hidden states.
 - Letting the decoder “look back” at the encoded document.
4. **Add & LayerNorm**
5. **Feed-Forward + Add & LayerNorm**
 - Same two-layer FFN and normalization as the encoder.

Finally a linear + softmax projects decoder outputs into vocabulary logits.

2. Summarization-Tailored Pretraining: Gap-Sentence Generation (GSG)

2.1. Motivation

Standard language-model pretraining (e.g., masked tokens or left-to-right LM) does not directly teach “condense this paragraph into a sentence.” PEGASUS's insight: if you mask out entire *important* sentences and train the model to generate those missing sentences from the remaining text, you force it to learn summarization.

2.2. Constructing Pretraining Examples

1. **Document Sampling:** Take a large corpus of documents (e.g., news articles).
2. **Gap Sentence Extraction:** Use a simple salience heuristic (e.g., TF-IDF or lead-n) to pick k sentences that are likely summary-worthy.

3. **Masking:** Replace those k sentences in the source with special **<MASK>** tokens.
4. **Input/Target Pair:**
 - **Input:** The document with k **<MASK>** placeholders.
 - **Target:** The concatenation of the k extracted sentences (in their original order).

2.3. Pretraining Objective

Minimize the cross-entropy between the decoder's output distribution and the target sentences, conditioned on the masked document:

$$\text{LGSG} = -\sum_{t=1}^T \text{target} \log P_{\theta}(y_t | y_{<t}, \text{MaskedDoc}).$$

$$\text{LGSG} = -\sum_{t=1}^T \log P_{\theta}(y_t | y_{<t}, \text{MaskedDoc}).$$

Because the decoder must generate fluent, coherent sentences that accurately reflect the masked content, the encoder learns to compress salient information into a fixed representation.

2.4. Long-Input Pretraining

News articles can be thousands of tokens long. PEGASUS extends the standard 512-token limit by:

- **Sliding-Window Masking:** Randomly masking sentences across the long document.
- **Truncation Strategies:** Training on segments up to 1–2K tokens, sometimes chaining segment masks so the model sees extended contexts.

This prepares the network to handle real-world articles (4K+ tokens) at fine-tuned summarization time.

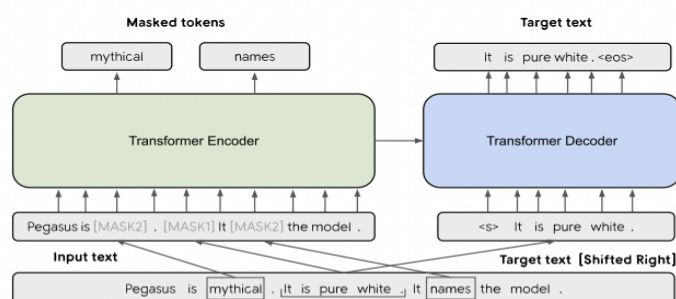


Figure 1: The base architecture of PEGASUS is a standard Transformer encoder-decoder. Both GSG and MLM are applied simultaneously to this example as pre-training objectives. Originally there are three sentences. One sentence is masked with [MASK1] and used as target generation text (GSG). The other two sentences remain in the input, but some tokens are randomly masked by [MASK2] (MLM).

Hierarchical Text Attention (HTA)

Hierarchical Text Attention (HTA) enhances a standard transformer encoder by explicitly modeling the natural structure of a document—words form sentences, and sentences form the document—through two successive attention stages:

1. Word-Level Focus

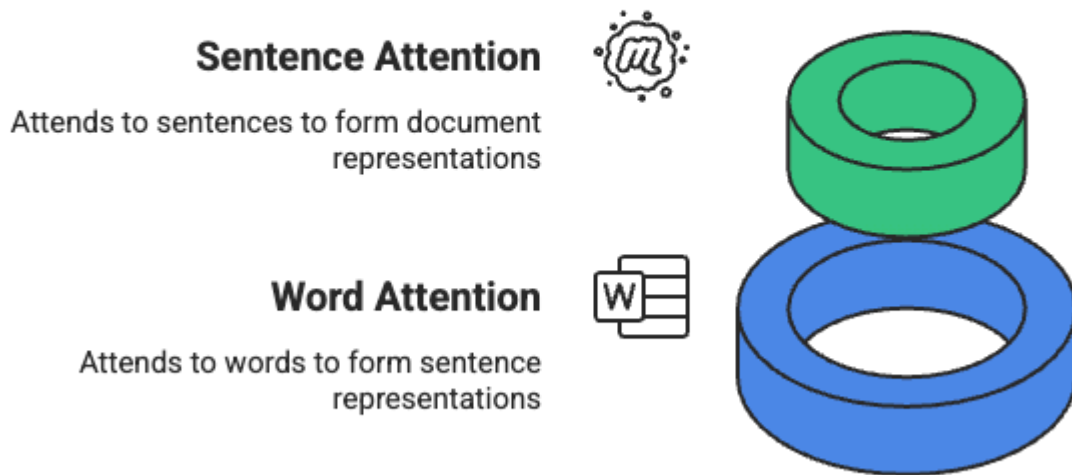
- Within each sentence, the model performs self-attention over the sequence of token embeddings.
- This stage learns which words in a sentence carry the most important meaning, dynamically assigning higher weight to key terms (e.g., proper nouns, action verbs, salient descriptors) and down-weighting common or filler words.
- The result is a single vector representation per sentence that captures its most significant content.

2. Sentence-Level Focus

- Taking the sentence vectors from the first stage, the model then applies a second self-attention pass across the sentences in a document.
- Here, it learns which sentences contribute most to the document's overall theme or message—such as identifying leads in a news article or pinpointing pivotal summary sentences.
- This yields a single, holistic document vector that distills the essence of the entire input.

By separating attention into these two levels, HTA allows the model to **differentiate importance at both micro (word) and macro (sentence) scales**, leading to richer and more interpretable representations. In practice, integrating HTA into Pegasus means that before any global or memory-augmented attention layers, the encoder already produces contextually weighted sentence and document embeddings—ensuring that the most critical information rises to the top for downstream summarization or generation.

Hierarchical Text Attention



Made with  Napkin

Advanced Positional Encodings



Advanced Positional Encodings replace the traditional approach of adding fixed, absolute position vectors to token embeddings with more flexible, relative schemes—most notably *rotary* or *relative* embeddings. In standard transformers, each token in the sequence receives a unique positional vector based only on its absolute position, which can limit the model's ability to generalize beyond the lengths seen during training. Rotary embeddings, in contrast, intertwine position information directly into the attention mechanism: rather than adding separate vectors, they “rotate” each token's query and key vectors by an angle determined by the distance between tokens. This means every pairwise attention score naturally reflects how far apart two tokens are, without requiring an explicit bias term for each relative distance.

Because these rotary or relative encodings focus on distances instead of fixed positions, they exhibit two key advantages for long-document summarization. First, they **generalize gracefully to sequences longer** than those encountered during training—there is no upper bound on embedding tables to overflow, and the same rotation logic applies whether tokens are near or far apart. Second, they **more accurately capture local vs. long-range dependencies**: close tokens will share similar rotary phases (and hence stronger relative position cues), while distant tokens will naturally attenuate each other's influence through phase differences. Empirical studies have shown that models using rotary or learned relative biases

outperform their sinusoidal counterparts on tasks requiring long-range reasoning and summarization of lengthy texts.

In practice, integrating advanced positional encodings into a Pegasus-based model simply means substituting the initial embedding step: as raw token embeddings are generated, they are immediately combined (via a rotation operation) with their positional context before any self-attention takes place. From that point onward, every transformer layer—and by extension, downstream modules like hierarchical attention or memory cross-attention—receives representations that inherently carry precise, distance-aware ordering information. This small change at the embedding stage yields more robust performance when compressing and summarizing documents that span thousands of tokens.

Comparison of Positional Encoding Methods

	 Standard Positional Encodings	 Rotary Embeddings
Position Encoding	Absolute position vectors added	Position intertwined into attention
Generalization to Longer Sequences	Limited by training lengths	Gracefully generalizes to longer sequences
Dependency Capture	Less accurate for long-range	Accurately captures local vs. long-range
Integration	N/A	Substitute initial embedding step

Memory Augmentation (MA)




Memory Augmentation (MA) enriches the transformer’s representations with external knowledge, which is vital in news summarization to ground the model in factual information (dates, figures, relationships) and to prevent hallucination. In our design, instead of relying solely on the parameters learned during pretraining, certain encoder layers can “peek” into a separate memory bank—either a collection of retrieved news snippets or embeddings drawn from a structured knowledge graph.

- **Slot-Based Memory Bank:** We organize external information into a fixed number of “slots.” Each slot holds a vector representing a fact or a piece of context (for example, an entity’s biography or a recent related article). During encoding, the model runs an additional attention mechanism in parallel to its normal self-attention: it uses the current token or sentence representations as *queries* and the memory slots as *keys* and *values*. This produces a memory-augmented context vector that is then fused (added or concatenated) with the original self-attention output. The fusion lets the model seamlessly combine information from the input text with pertinent background knowledge.
- **Selective Memory Lookup:** To keep this mechanism efficient, we can first compute a lightweight similarity between input representations and all memory slots, then select only the top-K most relevant slots. A second pass of attention over just these filtered slots concentrates on the most useful facts without overwhelming the model. This two-step strategy—score then attend—limits compute and ensures the model retrieves only what truly matters for summarizing the current article.
- **Knowledge Graph Interface:** When a structured knowledge graph is available, entities and relations can be linearized or embedded into the memory slots. During training, we can add auxiliary tasks such as predicting missing edges in the graph or answering simple fact-lookup questions. These objectives encourage the model to learn how to read and incorporate KG information effectively. At summarization time, if the input mentions “Company X,” the model can automatically attend to the memory slot holding Company X’s profile, bringing in relevant details that may not appear explicitly in the text.
- **Integration in Pegasus:** We interleave memory-augmented attention sublayers at strategic depths—midway through and near the top of the encoder. Early in encoding, the model grounds itself in broad background context; later, it refines its summary-worthy representations with those enriched facts. Because these memory queries are fast matrix multiplications and involve only a small number of slots, the extra cost is moderate compared

to the gains in factual accuracy and coherence.

Empirical studies of memory-augmented transformers show large gains on tasks requiring world knowledge or cross-document reasoning, while preserving inference efficiency by capping memory size and using selective lookups. In summary, MA equips the summarizer with an external “fact bank,” ensuring its outputs remain anchored in reality and rich in pertinent details.

Comparison of Memory Augmentation Techniques

Characteristic	Slot-Based Memory Bank	Selective Memory Lookup	Knowledge Graph Interface	Integration in Pegasus
 Mechanism	Attention mechanism with memory slots	Similarity scoring and filtering of slots	Linearized or embedded entities and relations	Interleaved memory-augmented attention sublayers
 Process	Fuses memory-augmented context vector	Two-step strategy: score then attend	Predicts missing edges or answers fact questions	Refines summary-worthy representations with enriched facts
 Efficiency	Additional attention mechanism in parallel	Lightweight similarity computation	Auxiliary tasks during training	Fast matrix multiplications

Made with  Napkin

Global Text Contextualization (GTC)

Global Text Contextualization (GTC) endows the model with a broader, corpus-level view beyond any single document’s boundaries. In practice, the attached code implements GTC in three complementary ways, each designed to help the encoder learn relationships across multiple inputs or an entire news corpus:

1. Global Tokens for Sparse Attention

- The code injects a small set of learned “global” token embeddings at the start of each input batch. Internally these tokens are treated specially: they participate in a sparse attention pattern that lets them see every block of the input (typically fixed-size segments of ~256 tokens) and also broadcast information back to those blocks. In effect, each global token aggregates signals from all parts of the document cluster, producing a distilled “global context” vector at each layer.

Downstream layers then use that global context to inform local decisions—so when the model processes a paragraph in document A, it already knows what topics or entities have been mentioned in document B.

2. Hierarchical Document Layers

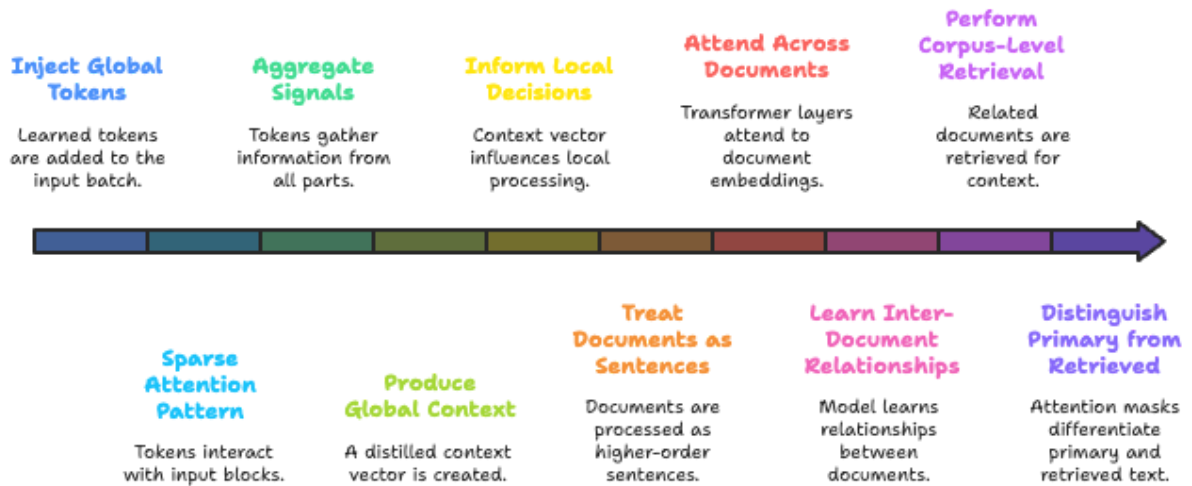
- Building on the word- and sentence-level HTA pipeline, the code then treats each entire document as if it were a “sentence” in a higher-order attention pass. After generating sentence embeddings for each document, a dedicated set of Transformer layers attends across those embeddings from all documents in a batch. This allows the model to learn inter-document relationships—spotting repeated themes, conflicting facts, or narrative threads that span multiple sources—instead of naively concatenating every document end-to-end.

3. Corpus-Level Retrieval

- As an optional step, the code can perform a lightweight retrieval over a larger news corpus at training time. Given the current input (one or more articles), it looks up related documents—by similarity on a precomputed index—and temporarily includes them as additional encoder inputs. Attention masks distinguish “primary” from “retrieved” text so that the model learns which context is core to the summary and which is background. This retrieval-augmented approach has been shown to strengthen factual consistency, because the model can resolve ambiguities or fill gaps by attending to true, unmasked source material rather than inventing content.

Across all three strategies, the common theme is that the encoder no longer operates in isolation on a single article. Instead, it weaves together information from within the document (via HTA), from external knowledge (via MA), and from the wider news ecosystem (via GTC). The result is a richly contextualized representation that helps the decoder produce summaries grounded not only in the input text but in a larger, coherent narrative of related reports and background facts.

Global Text Contextualization Process



Made with Napkin

Work Flow of Model

1. Input Tokenization & Rotary Positional Embedding

1. Tokenization

- Raw news text (one or more articles) is first split into tokens by the Pegasus tokenizer.
- We keep track of sentence and document boundaries so that later modules know which tokens belong together.

2. Embedding + Rotary PE

- Each token ID is mapped to a learned embedding vector.
- Immediately after lookup, we apply the **rotary positional transform**: each token's query/key vectors are rotated by angles determined by its position relative to others.

Storage: The resulting vectors (shape `[batch, seq_len, d_model]`) now carry both content and relative-position information, and are passed forward into the encoder.

2. Hierarchical Text Attention (HTA)

1. Sentence Splitting

- Using precomputed sentence boundaries, we view our long sequence as a list of sentences, each a contiguous subsequence of tokens.

2. Word-Level Self-Attention

- For each sentence in parallel, we apply a small Transformer block (or masked multi-head attention) that allows tokens within that sentence to attend to one another.
- We then pool (e.g. weighted sum via an internal attention head) across the sentence dimension to collapse each sentence into a single “sentence vector.”
- **Storage:** These sentence vectors (one per sentence, shape `[batch, num_sentences, d_model]`) are held in memory until the next step.

3. Sentence-Level Self-Attention

- We feed the list of sentence vectors into a second Transformer block that attends across sentences in the document (or across all documents in a multi-doc batch if we choose).
- Pooling across this output yields a single “document vector” per article (shape `[batch, d_model]`).
- **Storage:** Both the intermediate sentence vectors and the final document vectors are retained for later modules.

3. Memory Augmentation (MA)

1. Memory Bank Initialization

- A fixed-size bank of external “slots” is prepared. Each slot is a learned embedding representing a fact or a retrieved news snippet (shape `[M, d_model]`).

2. Cross-Attention Pass

- At one or more mid-level encoder layers, each document vector (or even each sentence vector) issues *queries* against the memory bank keys/values.
- This produces a “memory-context” vector that contains the weighted sum of the most relevant memory slots.
- **Gating/Selection:** Optionally, a lightweight scoring step filters the memory slots first, so only the top-K slots are actually attended over.

3. Fusion

- The new memory-context vector is fused (added or concatenated) with the HTA-generated document/sentence vectors.

- **Storage:** The fused vectors (now enriched with external knowledge) replace the original document vectors for downstream processing.

4. Global Text Contextualization (GTC)

1. Global Token Injection

- We prepend a handful of learned “global tokens” to each input (or separately to each document representation).
- These tokens are free to attend to—and be attended by—every token block in the sequence via a sparse block-local attention pattern.

2. Global Attention Layer

- In the highest encoder layers, we run a Transformer block where:
 - **Global tokens** attend to all sentence/document vectors (from HTA+MA).
 - **All sentence/document vectors** attend back to the global tokens.
- This two-way exchange yields:
 - A set of updated global token embeddings (each now summarizing whole-document or whole-batch context).
 - Document vectors that have directly absorbed signals from other documents or the entire input cluster.
- **Storage:** We pass both the updated document vectors and global token embeddings forward.

5. Decoder & Summary Generation

1. Preparing Encoder Outputs

- The final encoder outputs consist of:
 - The original token-level representations (with rotary PE).
 - HTA’s sentence/document vectors (enriched by MA).
 - Global token embeddings (carrying corpus-level context).

2. Transformer Decoder

- The decoder receives its own input IDs (previously generated summary tokens) and attends to the full set of encoder outputs via standard cross-attention.
- At each decoding step, it draws on:
 - Local details (via token-level keys/values).
 - Sentence-level structure (via HTA vectors).
 - Background facts (via memory-context vectors).
 - Cross-document themes (via global tokens).

3. Output

- The decoder emits the next summary token, repeating until the summary is complete.

