The Application of Transformer Architectures to Sequence-to-Sequence Tasks

Executive Summary

The landscape of natural language processing (NLP) has been fundamentally reshaped by the advent of the Transformer architecture, particularly in its application to sequence-to-sequence (seq2seq) tasks. This report provides a comprehensive analysis of this paradigm, tracing its evolution from early recurrent neural networks to the sophisticated models of today. A detailed examination of canonical encoder-decoder architectures, including the generalist T5, the denoising autoencoder BART, and the specialized PEGASUS, reveals that a model's performance is profoundly shaped by its pre-training objective.

The report explores the practical implementation of these models for diverse applications such as text summarization, machine translation, and conversational AI, highlighting the crucial role of libraries like Hugging Face in democratizing access to these powerful tools. A critical evaluation of architectural trade-offs—comparing encoder-only, decoder-only, and encoder-decoder designs—demonstrates that the optimal model choice is not universal but is a strategic decision contingent on the specific task and available computational resources. The report concludes by identifying cutting-edge alternatives like Mamba and Griffin, which challenge the dominance of the Transformer's quadratic complexity by offering linear-time scaling for long sequences, a significant step toward more efficient and sustainable AI. The core finding is that effective model selection and deployment require a nuanced understanding of a model's design, its intended purpose, and the practical challenges of implementation.

1. The Foundational Principles of Transformer-based Seq2Seq

1.1 The Evolution of Sequence-to-Sequence Modeling

The development of sequence-to-sequence modeling has been a journey of overcoming fundamental computational bottlenecks. Early models relied on Recurrent Neural Networks (RNNs) and their more advanced variants, such as Long Short-Term Memory (LSTMs).[1] These architectures processed sequential data by ingesting one element at a time, making parallel processing impossible and leading to inherently slow training times.[1] This serialization was a major limitation.

A more critical problem for these early models was the "fixed-length encoding vector" or "bottleneck" issue.[1] The encoder, an RNN, was tasked with compressing an entire input sequence into a single, fixed-size vector. While theoretically able to propagate information arbitrarily far, in practice, this fixed vector was incapable of retaining all relevant information from long input sequences, leading to significant information loss and a degradation in performance.[1] This limitation spurred the development of new approaches.

The attention mechanism was introduced as an enhancement to this structure in 2014 to address the fixed-length encoding problem.[3] This innovation allowed the decoder to selectively "focus on different parts of the input sequence during the decoding process" instead of relying solely on the compressed context vector.[3] This breakthrough was followed by a landmark paper in 2017, titled

Attention Is All You Need, which marked a profound shift by proposing a novel architecture that entirely dispensed with recurrence and convolutions, making attention the sole mechanism for information processing.[4] This architectural pivot enabled massive parallelization, fundamentally changing the economics of AI training and directly paving the way for the modern era of Large Language Models (LLMs).[4] The core principle was that efficiency and scalability, not just accuracy, were the primary drivers of progress.

1.2 The Encoder-Decoder Paradigm

The standard Transformer architecture for seq2seq tasks is a two-part model comprising an encoder and a decoder.[3] This design is particularly well-suited for tasks where the input and output sequences have different lengths or are not directly correlated, such as in text

summarization, machine translation, and automatic speech recognition.[2]

- **The Encoder:** The encoder's primary role is to process the input sequence and produce a rich, abstract representation. It reads the entire input and "encodes that spectrogram to form a sequence of encoder hidden states that extract important features" from the input.[6] This bidirectional processing allows it to capture a deep, contextual understanding of the entire input, which is a key advantage over unidirectional models.[8] This abstract representation effectively encodes the "meaning" of the input sequence as a whole.[6]
- **The Decoder:** The decoder's role is to take the encoder's hidden states and autoregressively generate the output sequence one token at a time.[3] It functions like a language model, predicting the next token based on the input context provided by the encoder and the tokens it has already generated.[6] A crucial architectural constraint is that the decoder's attention is "causal," meaning it is not allowed to "look into the future" at subsequent tokens.[6] This ensures that the generation process is a realistic, step-by-step prediction, which is essential for creative or translational tasks.

1.3 The Attention Mechanism

The power of the Transformer architecture lies in its sophisticated attention mechanisms, which enable it to efficiently capture complex dependencies.

- **Self-Attention:** At the core of both the encoder and decoder is the self-attention mechanism.[4] For each token in a sequence, the model computes three vectors: a Query (Q), a Key (K), and a Value (V).[4] The Query vector represents the information a specific token is "seeking," while the Key vectors represent the information that each token contains.[9] The dot product of a token's Query with all other tokens' Keys determines "alignment scores," which are then converted into attention weights.[9] These weights are used to form a weighted sum of the Value vectors, allowing each token to form a new representation based on a contextual understanding of all other tokens in the sequence.[9]
- **Multi-Head Attention:** The original *Attention Is All You Need* paper introduced multi-head attention to enhance this process.[4] This technique involves running the self-attention mechanism multiple times in parallel with different linear projections.[4] By doing so, the model can simultaneously focus on different aspects of the relationships between words, capturing a more varied and diverse set of perspectives on the input embeddings.[4]
- **Cross-Attention:** A key distinction in the decoder is the presence of a cross-attention mechanism.[6] This mechanism allows the decoder to "look at the encoder's representation of the input sequence".[6] Specifically, the decoder's queries attend to the encoder's keys and values, ensuring that each generated token is directly informed by

and grounded in the information extracted from the input sequence.[6] This is the primary way the encoder's output influences the decoder's generation.

2. A Deep Dive into Canonical Seq2Seq Architectures

The effectiveness of a Transformer model for a specific task is not solely determined by its core architecture but also by its pre-training objective. The following analysis examines three of the most prominent encoder-decoder models, revealing how their distinct pre-training philosophies shape their capabilities.

2.1 T5 (Text-to-Text Transfer Transformer)

The T5 model represents a generalist approach to NLP. Its most significant contribution is the unified "text-to-text" framework, which reframes every NLP problem as a text generation task.[10] This is achieved by prepending the input with a task-specific prefix, such as

"summarize:" for summarization or "translate English to German:" for translation.[10] This eliminates the need for unique, task-specific architectures and enables a single model to handle a diverse range of tasks, from translation to question answering, with only minor adjustments.[10]

The T5 architecture is a standard encoder-decoder Transformer with a range of sizes from 60 million to 11 billion parameters.[10] It employs a few specific design choices, such as using relative scalar embeddings that allow inputs to be padded on either side.[10] In a comparative study on business news summarization, T5 excelled, demonstrating superior performance in ROUGE-1 and METEOR scores compared to both BART and PEGASUS.[12] This suggests that T5 has a stronger ability to align its generated vocabulary and phrasing with human-written summaries, a testament to the power of its generalist pre-training.

2.2 BART (Bidirectional and Auto-Regressive Transformers)

BART’s innovation lies in its denoising autoencoder objective, a pre-training strategy that combines the strengths of both BERT and GPT.[12] The model is trained by corrupting an input

text using various "noising" functions and then learning to reconstruct the original, uncorrupted version.[14] This process is particularly effective for generation tasks because it forces the model to reason about sentence-level structure and long-range dependencies.[15]

The pre-training corruption schemes are diverse and strategic, including:

- **Token Masking:** Similar to BERT, random tokens are replaced with a `` token.[15]
- **Text Infilling:** Spans of text with lengths drawn from a Poisson distribution are replaced with a single `` token. This forces the model to predict the number and content of the missing words, a task directly relevant to generation.[15]
- **Sentence Permutation:** Sentences within a document are randomly shuffled, and the model must learn to restore the correct order.[15]

BART's architecture, a standard encoder-decoder, inherits a dual nature from this pre-training: a bidirectional encoder for comprehension (like BERT) and an autoregressive decoder for generation (like GPT).[12] In the comparative summarization study, BART demonstrated a strong capacity for faithful content reproduction, achieving the highest ROUGE-P (precision) score among the models, indicating a precise word overlap with reference summaries.[12]

2.3 PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive Summarization)

PEGASUS adopts a highly task-specific pre-training objective known as "Gap Sentence Generation" (GSG).[17] The core idea is that a pre-training objective that closely mimics the downstream task will yield better fine-tuning performance.[17] For PEGASUS, this means important sentences are removed from an input document, and the model is tasked with generating those missing sentences as the output.[17] This objective is a near-perfect analog for abstractive summarization.

This targeted approach has proven remarkably effective. PEGASUS achieved state-of-the-art performance on a wide range of summarization datasets.[17] A particularly compelling finding was its efficiency in low-resource scenarios.[17] The model was able to achieve near state-of-the-art results with as few as 1,000 fine-tuning examples, outperforming strong baselines that used significantly more data.[17] This demonstrates the power of a well-designed pre-training task to drastically reduce the need for large, costly supervised datasets.

The comparison of these three models reveals a key difference in philosophy: T5 is a generalist designed for broad adaptability, BART is a balanced model for both generation and

comprehension, and PEGASUS is a specialist optimized for a single, high-stakes task. The performance of these models shows that the pre-training objective is a crucial differentiator, often more influential than the raw architectural design in shaping a model's final capabilities and resource efficiency.

Table 1: Comparative Analysis of Canonical Encoder-Decoder Models

Model Name	Core Pre-training Objective	Architectural Strengths	Primary Use Cases	Comparative Performance (ROUGE-1 / METEOR)
T5	Text-to-Text Transfer Framework [10]	Unified architecture for diverse tasks via prefixes [10]	Translation, Summarization, Question Answering	0.354 / 0.35 (Highest on these metrics) [12]
BART	Denoising Autoencoder [12]	Bidirectional encoder (BERT-like) + Autoregressive decoder (GPT-like) [15]	Generation, Translation, Comprehension, Summarization	0.308 / 0.28 (Strong performance) [12]
PEGASUS	Gap Sentence Generation [17]	Task-specific pre-training for abstractive summaries [18]	Abstractive Summarization	0.245 / 0.25 (Good, but lower than others) [12]

3. Practical Applications and Implementation Best Practices

3.1 Text Summarization

Text summarization is a critical NLP task, and modern Transformer models excel at the abstractive approach, which involves generating new, concise sentences to convey the core meaning of a document.[20] This is in contrast to the traditional extractive method, which simply copies the most important sentences.[20]

For abstractive summarization, models like T5, BART, and PEGASUS are ideal. Implementing a summarization pipeline is streamlined by using the Hugging Face transformers library, which provides a high-level pipeline abstraction.[21] This abstraction handles complex sub-processes like tokenization and model loading, making powerful models accessible with minimal code.[21]

A case study of a summarizer using the BART model, for instance, involves a simple, step-by-step process:

1. Install the transformers library using !pip install transformers.[20]
2. Initialize the summarization pipeline by specifying the task and a pre-trained model checkpoint, such as "facebook/bart-large-cnn".[21]
3. Define the input text to be summarized.
4. Call the summarizer object, passing the text and specifying generation parameters such as max_length and min_length to control the output size.[21]
5. Print the decoded output, which is the final human-readable summary.[20]

This process highlights a significant trend: the democratization of AI. The modular and well-documented nature of libraries like Hugging Face abstracts away the complexity of managing a Transformer's full lifecycle, enabling a wider range of developers to apply these powerful models to real-world problems.

3.2 Machine Translation

Machine translation remains the canonical and most common application of the seq2seq paradigm. The process involves an encoder that processes the source language and a decoder that generates the target language.[23] Modern Transformer-based models have elevated the performance of this task to a new standard.

The implementation process with Hugging Face mirrors that of summarization. A specialized pipeline, such as "translation_en_to_de", can be used, and a model like Helsinki-NLP/opus-mt-en-nl can be loaded.[22] The process involves tokenizing the input text, using the model to generate a sequence of token IDs, and then decoding the output back into

human-readable text.[22]

3.3 Conversational AI

While early chatbots relied on retrieval-based models that output predefined responses [24], modern conversational AI is powered by generative models that can produce novel, contextually relevant text. The Hugging Face

TextGenerationPipeline offers a robust framework for building such chatbots.[25]

A key best practice is to format the conversation history as a list of dictionaries, with specific roles for system (to define the model's behavior) and user (for user input).[26] This structure provides essential context for the model to generate a coherent and consistent response.

However, moving from a research paper to a practical application introduces significant computational challenges. Large models can have a substantial memory footprint; for example, an 8 billion parameter model can require approximately 32 GB of memory in full precision.[25] To address this, several optimization techniques are crucial:

- **Quantization:** This process reduces the precision of the model weights (e.g., from 32-bit to 8-bit or 4-bit), which can significantly reduce memory usage without a proportional drop in performance.[25]
- **Data Type Reduction:** Changing the data type to torch.bfloat16 can save memory while maintaining reasonable performance.[26]
- **Hardware Acceleration:** Using the device_map="auto" parameter ensures that the model is loaded on the fastest available device, optimizing resource allocation.[26]

These practical considerations demonstrate that a model’s real-world utility is often bottlenecked by engineering challenges and resource constraints, highlighting that the model itself is only one part of the solution.

4. Architectural Trade-offs and Computational Considerations

4.1 The Great Debate: Encoder-Only vs. Decoder-Only vs. Encoder-Decoder

The choice of Transformer architecture is not arbitrary; each design is purpose-built for a specific class of problems. A nuanced understanding of their trade-offs is crucial for effective model selection.

- **Encoder-Only Models (e.g., BERT, RoBERTa):** These models process the entire input sequence bidirectionally, allowing every token to attend to all others.[8] This design is optimized for deep contextual understanding. They are the ideal choice for understanding-based tasks like text classification (e.g., sentiment analysis), Named Entity Recognition (NER), and semantic search.[8] A key advantage is their fast inference for these tasks due to the parallel processing of all tokens.[8] However, they are not suitable for text generation, as they are not designed to produce new content.[8]
- **Decoder-Only Models (e.g., GPT, LLaMA):** These models are designed for generative tasks. They operate autoregressively, predicting one token at a time from left to right, and use causal masking to prevent them from seeing future tokens.[8] This sequential generation process makes them highly effective for tasks like text completion, story writing, and chatbots.[8] While they scale well and have become the standard for modern LLMs, their sequential generation can be slower for inference on a per-token basis compared to the parallel processing of encoder-only models.[8]
- **Encoder-Decoder Models (e.g., T5, BART):** This hybrid architecture combines a bidirectional encoder with an autoregressive decoder.[8] The encoder provides a comprehensive understanding of the input, while the decoder generates the output. This separation of concerns makes them the best choice for a wide range of seq2seq tasks, including machine translation and summarization, where a transformation from one text format to another is required.[8] They are more flexible than decoder-only models for tasks that require a full input context, but they can be more computationally expensive than either a standalone encoder or decoder.[8]

The ultimate conclusion is that there is no single "best" architecture. A practitioner must first deeply understand the nature of the problem—whether it's an understanding task, a generation task, or a transformation task—before selecting the model that is architecturally fit for purpose.

Table 2: Architectural Trade-offs for Different Transformer Model Types

Architecture Type	Typical Models	Core Function	Best Use Cases	Key Strengths	Primary Trade-offs

Encoder-Only	BERT, RoBERTa	Bidirectional Understanding	Classification, NER, Retrieval, Semantic Search	Fast inference due to parallel processing of all tokens [8]	Cannot generate text [8]
Decoder-Only	GPT, LLaMA	Autoregressive Generation	Text Completion, Chatbots, Code Generation	Best for creative and conversational generation tasks [8]	Less efficient for understanding tasks and sequential inference is slower [8]
Encoder-Decoder	T5, BART, PEGASUS	Sequence Transformation	Machine Translation, Summarization, Paraphrasing	Flexible, handles tasks with variable input/output lengths [8]	Can be computationally more expensive than the other two individually [8]

4.2 Computational Efficiency and Scaling Challenges

A major challenge for Transformers at scale is the computational cost of the self-attention mechanism, which scales quadratically with sequence length ($O(n^2)$).[1] This makes processing long documents computationally expensive and a significant barrier for applications in fields like legal document analysis or academic research.[28]

Furthermore, to avoid re-calculating keys and values at each decoding step, these are stored in a Key-Value (KV) cache, which grows linearly with sequence length ($O(n)$).[29] For long sequences, this cache can consume a significant amount of memory, further limiting a model's ability to handle extensive context.[29]

To address these issues, various optimization strategies have been developed:

- **Quantization:** This involves reducing the precision of the model's weights and activations (e.g., from 32-bit to 8-bit or 4-bit).[25] This significantly reduces memory usage and can speed up inference without a large loss of accuracy.[25]
- **Pruning and Knowledge Distillation:** These techniques reduce the model's size by removing redundant parameters or by training a smaller model to mimic the performance of a larger one.[28]
- **Hardware-Aware Optimizations:** Strategies that leverage specific hardware features, such as memory tiling and cache-friendly fusion strategies, can lead to substantial reductions in latency.[28]

4.3 Generation Strategies

The quality and speed of a model's output are heavily influenced by the decoding strategy used. Beyond the simple, default greedy search, several advanced methods are available:

- **Greedy Search:** This is the simplest method, where the model selects the token with the highest probability at each step. While fast, it can lead to repetitive or suboptimal outputs.[30]
- **Beam Search:** A more sophisticated approach that maintains multiple generated sequences (beams) at each step and selects the one with the highest overall probability at the end of the process.[30] This allows the model to "look ahead" and choose a sequence that might not have the most probable initial token but ultimately leads to a better result. It is particularly well-suited for input-grounded tasks like translation and speech recognition.[30]
- **Sampling:** This method introduces randomness by selecting a token based on its probability distribution, which helps to reduce repetition and produce more creative and diverse outputs.[30]
- **Speculative Decoding:** This is an advanced technique for accelerating inference. A smaller, faster "assistant" model generates a set of candidate tokens, which the main, larger model then verifies in a single forward pass.[30] This can significantly speed up token generation, particularly for large language models where each token generation is computationally expensive.[30]

The performance of a Transformer in a real-world application is not solely a function of its architecture. An efficient implementation pipeline that includes a suitable decoding strategy and memory optimizations is equally important. This highlights that model selection is just one piece of a larger engineering challenge, where hardware, algorithms, and practical constraints must all be considered.

5. Emerging Trends and Conclusion

5.1 A New Frontier: Transformer Alternatives

The quadratic complexity of the self-attention mechanism remains the primary bottleneck for Transformer models, particularly when processing long sequences. This limitation has spurred a new wave of research into alternative architectures that challenge the foundational premise of the *Attention Is All You Need* paper. These new models suggest that a hybrid or entirely different mechanism may be superior, especially when scalability to extremely long contexts is the priority.

- **Mamba (State Space Models):** Mamba is a deep learning architecture that has emerged as the first true competitor to the Transformer for language modeling.[31] It is based on a State Space Model (SSM) that compresses the entire sequence into a fixed-size hidden state.[32] This design allows Mamba to achieve linear scaling in both time and memory for long sequences, a significant advantage over the quadratic complexity of Transformers.[29] The architecture's ability to selectively focus on or ignore specific parts of the past input history gives it a capability previously unique to Transformers.[31]
- **Griffin (Hybrid Architectures):** Griffin is a hybrid model that mixes gated linear recurrences with local attention to address the computational bottlenecks of global attention.[33] Its architecture combines a recurrent block with a local Multi-Query Attention (MQA) block, leveraging the fast inference and efficient scaling of recurrence while retaining the benefits of attention for local dependencies.[33] Griffin has demonstrated performance comparable to leading decoder-only models like Llama-2 while being trained on a fraction of the data.[33] The emergence of architectures like Mamba and Griffin represents a return to the principles of recurrence, but with critical modern improvements that address the historical weaknesses of RNNs. This new wave is driven by a need for efficiency at scale, fundamentally re-thinking how sequence models can be built.

5.2 Conclusion and Actionable Recommendations

The choice of a Transformer model for a sequence-to-sequence task is a sophisticated decision that extends far beyond simply selecting the largest or most popular model. The analysis presented in this report provides a structured framework for making an informed choice based on the nature of the task and the practical constraints of the application.

1. **Define the Task:** The first step is to categorize the problem. If the goal is a transformative task, such as translation or summarization, an encoder-decoder architecture is the most natural fit. For understanding-based tasks like classification, an encoder-only model is a more efficient choice. For pure text generation, a decoder-only model is the standard.
2. **Evaluate the Pre-training Objective:** For encoder-decoder tasks, the model's pre-training objective is a primary differentiator. T5 is recommended for its unified flexibility across diverse tasks. BART is a strong choice when the task requires a balance of comprehension and generation. PEGASUS is the optimal model for abstractive summarization, especially in low-resource settings.
3. **Assess Resource Constraints:** The computational cost of Transformer models, particularly the quadratic scaling of attention and the memory footprint of the KV cache, cannot be overlooked. For memory-constrained environments or applications requiring high inference speed, model quantization and advanced decoding strategies like speculative decoding are essential.
4. **Embrace Emerging Architectures:** The field is in a state of rapid evolution. The emergence of architectures like Mamba and Griffin, which offer linear scaling and address the core limitations of the Transformer, indicates a new frontier in sequence modeling. For applications involving extremely long sequences or where computational efficiency is a critical business metric, these models represent a compelling and potentially superior alternative.

In conclusion, the era of the Transformer has brought unprecedented capabilities to seq2seq modeling. However, the future of the field is moving toward more efficient, specialized, and purpose-built architectures. The key to success lies in a deep, strategic understanding of a model's design, rather than a superficial reliance on its popularity or size.

Works cited

1. Transformer (deep learning architecture) - Wikipedia, accessed on August 25, 2025, https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)
2. NLP From Scratch: Translation with a Sequence to Sequence Network and Attention, accessed on August 25, 2025, https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html?highlight=glove
3. Seq2seq - Wikipedia, accessed on August 25, 2025, https://en.wikipedia.org/wiki/Seq2seq
4. Attention Is All You Need - Wikipedia, accessed on August 25, 2025, https://en.wikipedia.org/wiki/Attention_Is_All_You_Need

5. Attention is All you Need - NIPS, accessed on August 25, 2025, https://papers.nips.cc/paper/7181-attention-is-all-you-need
6. Seq2Seq architectures - Hugging Face Audio Course, accessed on August 25, 2025, https://huggingface.co/learn/audio-course/chapter3/seq2seq
7. (PDF) The evolution, applications, and future prospects of large language models: An in-depth overview - ResearchGate, accessed on August 25, 2025, https://www.researchgate.net/publication/377932211_The_evolution_applications_and_future_prospects_of_large_language_models_An_in-depth_overview
8. Choosing between Encoder,Decoder and Encoder-Decoder Models ..., accessed on August 25, 2025, https://medium.com/@yed.pavankumar/choosing-between-encoder-decoder-and-encoder-decoder-models-fa03ea5df2ff
9. What is a Transformer Model? - IBM, accessed on August 25, 2025, https://www.ibm.com/think/topics/transformer-model
10. T5 - Hugging Face, accessed on August 25, 2025, https://huggingface.co/docs/transformers/model_doc/t5
11. Simplified encoder-decoder transformer architectures used by BART ..., accessed on August 25, 2025, https://www.researchgate.net/figure/Simplified-encoder-decoder-transformer-architectures-used-by-BART-and-T5_fig1_347234294
12. Summarizing Business News: Evaluating BART, T5, and PEGASUS ..., accessed on August 25, 2025, https://www.iieta.org/journals/ria/paper/10.18280/ria.380311
13. Summarizing Business News: Evaluating BART, T5, and PEGASUS for Effective Information Extraction - IIETA, accessed on August 25, 2025, https://www.iieta.org/download/file/fid/132319
14. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension - Meta Research, accessed on August 25, 2025, https://research.facebook.com/publications/bart-denoising-sequence-to-sequence-pre-training-for-natural-language-generation-translation-and-comprehension/
15. Papers Explained 09: BART. BART is a denoising autoencoder built... | by Ritvik Rastogi | DAIR.AI | Medium, accessed on August 25, 2025, https://medium.com/dair-ai/papers-explained-09-bart-7f56138175bd
16. Mike Lewis et. al., accessed on August 25, 2025, https://ysu1989.github.io/courses/au20/cse5539/BART.pdf
17. PEGASUS: A State-of-the-Art Model for Abstractive Text Summarization - Google Research, accessed on August 25, 2025, https://research.google/blog/pegasus-a-state-of-the-art-model-for-abstractive-text-summarization/
18. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization - arXiv, accessed on August 25, 2025, https://arxiv.org/abs/1912.08777
19. FACTPEGASUS: Factuality-Aware Pre-training and Fine-tuning for Abstractive Summarization - ACL Anthology, accessed on August 25, 2025,

https://aclanthology.org/2022.naacl-main.74.pdf
20. Text-Summarization Case Study. Why Text Summarization? | by ..., accessed on August 25, 2025, https://medium.com/@raguwing/text-summarization-case-study-d70f82e18728
21. How to Build A Text Summarizer Using Huggingface Transformers, accessed on August 25, 2025, https://www.freecodecamp.org/news/how-to-build-a-text-summarizer-using-huggingface-transformers/
22. Hugging Face Pre-trained Models: Find the Best One for Your Task - neptune.ai, accessed on August 25, 2025, https://neptune.ai/blog/hugging-face-pre-trained-models-find-the-best
23. www.geeksforgeeks.org, accessed on August 25, 2025, https://www.geeksforgeeks.org/nlp/machine-translation-with-transformer-in-python/#:~:text=Transformer%20is%20a%20deep%20learning,(English%20in%20our%20example).
24. Chatbot Tutorial - PyTorch documentation, accessed on August 25, 2025, https://docs.pytorch.org/tutorials/beginner/chatbot_tutorial.html
25. Chat basics - Hugging Face, accessed on August 25, 2025, https://huggingface.co/docs/transformers/main/conversations
26. Chat basics - Hugging Face, accessed on August 25, 2025, https://huggingface.co/docs/transformers/conversations
27. Encoder-Decoder Transformers vs Decoder-Only vs Encoder-Only: Pros and Cons - YouTube, accessed on August 25, 2025, https://www.youtube.com/watch?v=MC3qSrsfWRs
28. Energy-Efficient Transformer Inference: Optimization Strategies for Time Series Classification - arXiv, accessed on August 25, 2025, https://arxiv.org/html/2502.16627v2
29. Transformer alternatives in 2024 - Nebius, accessed on August 25, 2025, https://nebius.com/blog/posts/model-pre-training/transformer-alternatives-2024
30. Generation strategies - Hugging Face, accessed on August 25, 2025, https://huggingface.co/docs/transformers/generation_strategies
31. What Is A Mamba Model? | IBM, accessed on August 25, 2025, https://www.ibm.com/think/topics/mamba-model
32. Mamba Explained - The Gradient, accessed on August 25, 2025, https://thegradient.pub/mamba-explained/
33. Griffin: Mixing Gated Linear Recurrences with Local Attention ... - arXiv, accessed on August 25, 2025, https://arxiv.org/abs/2402.19427
34. Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models, accessed on August 25, 2025, https://arxiv.org/html/2402.19427v1