
# Exploring Neural Approaches for Tiny Story Generation

**DATS 6312: Natural Language Processing**
**Instructor:** Prof. Amir Hossein Jafari

**Group Name:** Group 5
**Team Members:**
Abhilasha Singh
Pranjal Wakpaijan
Shabnam Rafat Ummey Sawda

Master of Data Science
George Washington University

**Date:** December 8, 2025

**Table of Contents:**

# 1. Introduction

The automatic generation of coherent and engaging short stories remains a central challenge within the field of Natural Language Processing (NLP). Story generation requires more than language fluency; effective models must understand narrative structure, maintain thematic and character consistency, incorporate given constraints, and generate contextually appropriate continuations. Unlike tasks such as summarization or translation, narrative generation is inherently open-ended and demands creativity, long-range dependency modeling, and the ability to integrate user-provided cues such as keywords or story beginnings.

This project investigates a range of neural approaches for children's story generation, leveraging the Tiny Stories dataset from Hugging Face, a collection of synthetically generated, child-friendly narratives created using GPT-3.5 and GPT-4. The dataset is characterized by its simple vocabulary, concise narrative style, and clear story progression, making it an ideal testbed for studying how different model architectures capture narrative flow, sentence-level coherence, and adherence to constraints such as required keywords. Given an initial story prompt and a set of target vocabulary words, the goal is to produce a complete story continuation that is coherent, grammatically sound, age-appropriate, and semantically aligned with the prompt.

# 2. Dataset Description

## 2.1 Initial Dataset

This project uses a curated subset of the Tiny Stories dataset, consisting of 50 JSON files, each containing thousands of synthetically generated children's stories. These files were merged and preprocessed to create the final dataset used for model training and evaluation.

### 2.1.1 Dataset Source and Properties

Hugging Face Dataset: Tiny Stories
Data: 49 million
License: CDLA-Sharing-1.0
Language: English
Download Size: ~1 GB
Number of Rows: ~2.14 million stories (full dataset)

### 2.1.2 Dataset Structure

Each entry in the Tiny Stories dataset includes multiple fields that describe the story and its generation metadata. A typical JSON sample contains:

story: The full children's story written in simple words.
instruction: A structured object containing:

- prompt: A natural language instruction used to generate the story.
- words: A list of required words the story must include (e.g., ["shake", "guard", "blue"]).
- features: Additional stylistic or structural constraints (optional).
- summary: A short summary of the story.

source: The model that generated the story (GPT-3.5 or GPT-4).

## 2.2 Cleaned Dataset

After merging the 50 JSON files from the Tiny Stories collection, the dataset underwent a systematic preprocessing workflow to prepare it for model training and evaluation. These cleaning steps ensured consistent

formatting, removed unnecessary metadata, normalized text structure, and filtered out stories that did not meet the required quality thresholds. As a result, the cleaned dataset contains only well-structured, high-quality samples suitable for training neural story-generation models.

#### 2.2.1 Cleaned Dataset Properties

The final cleaned dataset was split into training, validation, and test sets following an 80/10/10 ratio. The resulting dataset sizes are:

Training Data: 79,192 samples
Validation Data: 9,899 samples
Test Data: 9,899 samples
The combined dataset size across all CSV files is approximately 160 MB (≈ 0.15 GB), making it compact enough for efficient experimentation while still large enough to train both small and large language models.

#### 2.2.2 Cleaned Dataset Structure

After preprocessing, the dataset contains the following columns (as produced by EDA.py):

**story** – normalized full story text
**words** – list of required words extracted from the instruction
**story_beginning_prompt** – first 10 words of each story used as prompt
**word_count** – total number of words in the story
The following fields were intentionally removed because they were not needed for the generation task: **instruction**, **summary**, and **source**.

## 3. Exploratory Data Analysis

This section describes the exploratory analysis performed on the TinyStories dataset to understand its structure, verify data quality, and prepare it for downstream modeling. The EDA pipeline was implemented using Python, leveraging packages such as pandas, NumPy, matplotlib, seaborn, and project-specific utility functions defined in utils.py.

### 3.1 Loading and Initial Inspection of Data

The dataset was first loaded by reading all JSON files stored in the /Data directory. This was accomplished using the custom utility function read_and_merge_json_files(), which performs the following operations:

Reads each JSON file sequentially
Concatenates all story entries into a single DataFrame
Removes duplicate records to ensure data quality
Returns the final merged dataset for further processing
This step consolidates all available story files into one unified dataset, making it suitable for analysis and model development.

### 3.2 Extracting Keywords and Generating Prompts

To prepare the dataset for prompt-based story generation, several preprocessing steps were applied:

The nested "instruction" field was parsed to extract the list of required words, which represent vocabulary constraints for each story.
A story beginning prompt was generated by extracting the first 10 words of every story. This serves as the input context for models such as GPT-2, FLAN-T5, LSTM-Attention, and SLM+GNN.

These transformations allow models to be conditioned on both a narrative beginning and specific vocabulary requirements.

### 3.3 Computing Word Counts for Filtering and Analysis

The total number of words in each story was calculated. This metric is important because it:

Enables filtering out stories that are too short or too long
Supports statistical analysis of story lengths
Helps understand the general distribution of narrative sizes in the dataset
Word count became a key feature for downstream filtering decisions.

### 3.4 Normalizing Text for Model Compatibility

To ensure compatibility with NLP models and tokenizers, the story text was normalized by removing:

Excess whitespace
Line breaks
Tabs
Irregular spacing patterns
The purpose is to create clean, standardized text representations that prevent tokenization errors and inconsistencies during model training.

### 3.5 Removing Irrelevant Metadata Fields and Validating Data Quality

Certain fields such as summary, source, and instruction were removed because they were not required for the story generation task and would only add noise to the dataset. Additionally, null values were checked to confirm dataset completeness and identify potential issues.

This validation step ensured that the cleaned dataset contained only fully usable samples.

### 3.6 Visualizing Story Length Distributions and Applying Filtering

Histograms of word counts were plotted to visualize the distribution of story lengths before and after filtering. The initial distribution revealed many extremely short or excessively long stories.

To maintain modeling consistency, only stories with 50 to 400 words were retained.

Rationale for filtering:

Stories < 50 words: Too short to provide meaningful narrative structure
Stories > 400 words: May exceed model context limits and introduce training instability
The filtered dataset resulted in a more uniform, high-quality collection of stories appropriate for training multiple generative models.

## 4. Evaluation Metrics

This project employs a combination of lexical, semantic, structural, and constraint-based evaluation metrics to comprehensively assess story-generation models. The following subsections describe each metric, its mathematical formulation, and the intuition behind its use.

### 4.1 BLEU

BLEU (Bilingual Evaluation Understudy) is a precision-based metric that evaluates how similar a generated story is to a reference story by measuring overlap in n-grams (continuous word sequences). BLEU rewards longer

matching sequences and penalizes unnaturally short generations. BLEU is primarily a lexical similarity metric, making it useful for analyzing surface-level overlap but less reliable for creative, semantically flexible tasks such as story generation.

BLEU = 1 (100%) → perfect n-gram overlap
BLEU = 0 → no similarity
Typical BLEU for generative tasks: 0.1–0.5
BLEU is calculated by combining n-gram precisions ($p_1$, $p_2$, $p_3$, $p_4$) using the geometric mean. If any precision is 0, the BLEU score drops sharply.

Brevity Penalty (BP): To prevent artificially short outputs from achieving high scores, BLEU applies a brevity penalty. Let c = length of candidate (generated story) and r = length of reference. If $c < r$, $BP < 1$; otherwise $BP = 1$.

## 4.2 Perplexity

Perplexity quantifies how well a language model predicts a sequence of text and serves as an indicator of fluency and naturalness in generated stories. In this project, perplexity is computed on each generated story using a pretrained GPT-2 model as the evaluation reference, effectively treating GPT-2 as a "judge" that assesses how predictable and linguistically coherent the generated output is.

Low perplexity = the model is confident and predicts text well
High perplexity = the model is confused or surprised by the text
You can think of perplexity as "how many choices the model feels it has at each step." A model with perplexity = 10 behaves like it has ~10 equally likely word choices when generating text. Perplexity is calculated as the exponential of cross-entropy loss, where cross-entropy measures how well the model predicts the next token.

## 4.3 BERTScore

BERTScore assesses the semantic similarity between generated and reference stories by comparing their contextual representations rather than relying on exact word overlap, as in BLEU or ROUGE. Each token is transformed into a contextual embedding using the RoBERTa-Large model, and semantic closeness is measured using cosine similarity.

For every token, the metric identifies its strongest semantic match in the opposing sentence and computes precision, recall, and an overall F1 score; the latter being the primary measure used in this project. RoBERTa-Large, a 355-million-parameter transformer encoder with 24 layers, 1024-dimensional embeddings, and 16 attention heads per layer, trained on 160 GB of text, provides deep contextual representations that effectively capture meaning. This makes BERTScore particularly well suited for evaluating creative story generation, where semantic fidelity is more important than exact lexical overlap.

## 4.4 ROUGE Scores

ROUGE evaluates how much of the reference content appears in the generated story, focusing on recall rather than precision. This makes ROUGE suitable for generative tasks where the model may paraphrase but still maintain content relevance. These metrics together assess lexical and structural alignment of generated stories with reference narratives.

ROUGE Variants:

ROUGE-1 (Unigram Recall): Measures overlap of individual words.
ROUGE-2 (Bigram Recall): Measures overlap of word pairs; captures phrase-level similarity.

ROUGE-L (Longest Common Subsequence): Measures the longest sequence of words appearing in the same order in both texts.

### 4.5 Keyword-based Metrics

**keyword_strict_accuracy** evaluates whether the generated story includes all required keywords. It is a binary metric: if every required term is present, the score is 1; if even a single keyword is missing, the score is 0. The final strict accuracy value is obtained by averaging these binary outcomes across all samples. Because it provides no partial credit, this metric serves as a stringent measure of a model's ability to fully satisfy keyword constraints.

**keyword_avg_percentage** measures the proportion of required keywords that appear in the generated story. Unlike strict accuracy, this metric provides partial credit, reflecting how many of the required terms were successfully included. It offers a more flexible assessment by quantifying the percentage of matched keywords relative to the total required set.

## 5. Model Architectures

### 5.1 Baseline Model: GPT-2 Causal Language Model

As a baseline model, we employ a fine-tuned GPT-2 Transformer configured as a causal language model for conditional story generation on the TinyStories dataset. GPT-2 is a unidirectional (decoder-only) architecture that generates text by predicting each next token conditioned on all preceding tokens, making it well suited for story continuation tasks.

Each training sample is constructed as a single concatenated text sequence containing:

An instruction header (e.g., "You are a children's story writer…")
A list of required keywords
The initial story prompt
The target continuation

#### 5.1.1 Architecture Description

**Tokenization and Embedding Layer:** GPT-2 uses a Byte-Pair Encoding (BPE) tokenizer that breaks text into subword units. Each token ID is mapped into two learned embeddings:

Token embedding — captures semantic meaning of the token
Positional embedding — encodes the token's position in the sequence
The sum of these two vectors forms the input to the Transformer stack. Unlike recurrent models, GPT-2 does not process words sequentially through time — it processes all positions in parallel, but with masking to ensure causality.

**Transformer Decoder Blocks (Core of GPT-2):** GPT-2 consists of a stack of identical decoder-only Transformer layers, each containing:

**a. Masked Multi-Head Self-Attention:** This is the key mechanism that replaces recurrence. For each token, GPT-2 computes attention with all previous tokens, but not future ones (causal mask). This allows the model to learn long-range dependencies such as remembering a character introduced earlier, connecting events across the story, and maintaining consistent style. The "multi-head" part means GPT-2 uses multiple parallel attention operations, each learning a different aspect of context (e.g., syntax, dialogue structure, keyword relevance).

**b. Feed-Forward Network (FFN):** Each position passes through a 2-layer MLP that transforms and enriches its representation.

**c. Residual Connections + LayerNorm:** These stabilize deep networks and help preserve information across layers. Together, these blocks transform raw token embeddings into rich contextual representations that understand relationships within the prompt.

**Causal Language Modeling Head:** At the top of GPT-2, a linear projection layer maps each final hidden vector into a vocabulary-sized logits vector. Softmax gives a probability distribution over the next token. This is what enables fluent story continuation, one token at a time.

### 5.1.2 Key Training Settings

Seed: 42 (for reproducibility)
Max sequence length: 192 tokens
Epochs: 5
Batch size: 2 (train and validation)
Optimizer: Adam with learning rate $3\times10^{-5}$
Device: GPU if available, otherwise CPU
This sequence is tokenized using the GPT-2 tokenizer (≈50k vocabulary; maximum length 192 tokens in our setup) and processed by a 12-layer decoder stack with masked multi-head self-attention and position-wise feed-forward sublayers. The model predicts the next token at every position, with the final linear language-model head mapping hidden states to vocabulary logits. Training is performed end-to-end using Adam with a learning rate of $3\times10^{-5}$, optimizing the causal language modeling objective. Cross-entropy loss is computed on all non-padded target tokens and averaged across tokens. The checkpoint achieving the lowest validation loss is retained and subsequently used to generate children's stories conditioned on prompts and keyword lists.

### 5.1.3 Results

**Validation Evaluation Summary:**

corpus_bleu: 0.0829
avg_perplexity: 13.5553
bert_precision: 0.5843
bert_recall: 0.6464
bert_f1: 0.6132
rouge1: 0.4313
rouge2: 0.1386
rougeL: 0.2395
keyword_strict_accuracy: 0.5300
keyword_avg_percentage: 80.9038
The evaluation results show that GPT-2 performs reasonably well in capturing the semantic meaning of stories (BERTScore F1 ≈ 0.61) and maintains moderate lexical overlap with reference stories (ROUGE-1 ≈ 0.43). The model successfully includes the required keywords in most cases, achieving 80.9% keyword coverage and 53% strict accuracy. However, the low BLEU score (0.0829) and relatively high perplexity (13.55) indicate that GPT-2 struggles with generating fluent, precise, and structurally aligned continuations. Overall, GPT-2 provides a meaningful but imperfect baseline, demonstrating clear potential while leaving significant room for improvement through more advanced architectures.

## 5.2 LSTM–Attention Seq2Seq Model

As a second architecture, we implement a sequence-to-sequence LSTM model with Bahdanau attention for conditional story generation on the Tiny Stories dataset. In contrast to the decoder-only GPT-2 baseline, this model follows an encoder–decoder design:

The encoder reads the story prompt (including keywords)
The decoder generates the full story one token at a time
The attention module lets the decoder focus on the most relevant parts of the encoded prompt at each step
The GPT-2 tokenizer is used only to map text to subword token IDs; all sequence modeling is done by the custom LSTM network.

Each training example is split into two text fields:

**1. Source sequence (src_text):** Template: "You are a children's story writer. Use these words: <keywords>. Story prompt: <beginning>. Write a complete story:" This serves as the encoder input and encodes both the required words and the initial story context.

**2. Target sequence (tgt_text):** The gold story continuation, stripped of extra whitespace. This is fed to the decoder during training using teacher forcing.

**5.2.1 Architecture Description**
The proposed model follows a Sequence-to-Sequence (Seq2Seq) architecture augmented with Bahdanau attention to generate story continuations from a given prompt and set of required keywords. The system consists of three main components: a GPT-2 tokenizer, an LSTM-based encoder, and an LSTM-based decoder with attention, all implemented using PyTorch.

**1. Tokenization and Embedding Layer:** Text inputs, including the constructed prompts and reference stories, are first processed using the GPT-2 Byte-Pair Encoding (BPE) tokenizer. Although GPT-2's tokenizer is used for token-to-ID mapping, the model employs a trainable embedding layer (nn.Embedding) that learns 256-dimensional dense vector representations for all vocabulary tokens. These embeddings are learned jointly with the Seq2Seq model during training.

**2. Encoder: LSTM Representation of the Prompt:** The encoder is a unidirectional LSTM that reads the tokenized prompt sequence one word at a time. For each input token, the encoder produces a hidden-state vector capturing contextual information. The encoder outputs a sequence of hidden states representing every input token and the final hidden and cell states (h and c), which initialize the decoder. The encoder thus serves as a compressed representation of the story prompt and the required keywords.

**3. Bahdanau Attention Mechanism:** To overcome the limitations of fixed-size context vectors, the model integrates a Bahdanau (additive) attention mechanism. At each decoding step, the attention module: (1) Compares the current decoder hidden state with all encoder hidden states, (2) Computes alignment scores for each input position, (3) Produces a normalized attention distribution using softmax, (4) Generates a context vector, which is a weighted sum of encoder states. This enables the decoder to selectively "attend to" the most relevant parts of the input, such as specific keywords, when generating each word in the story.

**4. Decoder: LSTM with Attention:** The decoder is another LSTM network that generates the story one token at a time. At each time step, the decoder receives the embedding of the previous output token (teacher-forcing during training) and the attention-derived context vector representing relevant parts of the prompt. These two vectors are concatenated and fed into the decoder LSTM. The decoder then updates its hidden state and produces a vocabulary-sized logits vector via a linear output layer. During training, the model learns to predict the next token in the reference story. During inference, the decoder begins with a special BOS token and generates tokens greedily until the EOS token is reached.

**5. Training Objective:** The model is trained using cross-entropy loss, comparing predicted output tokens with ground-truth target tokens. Padding tokens are ignored during loss computation. Gradient clipping is applied to

prevent exploding gradients, and the optimizer used is Adam with a learning rate of 3e-4. The best-performing model on the validation set (measured by token-level loss) is saved for downstream evaluation.

**6. Story Generation Process:** During inference, the encoder processes the input prompt, and the decoder generates a continuation word-by-word using greedy decoding. At each step, attention guides the decoder to focus on the most relevant tokens in the input, enabling more coherent and keyword-aligned storytelling.

### 5.2.2 Key Training Settings

Training follows the same overall procedure as the GPT-2 baseline, with hyperparameters adapted for the LSTM model:

Seed: 42 (for reproducibility)
Tokenizer: GPT-2 tokenizer used for both source and target sequences
Embedding dimension: 256
Hidden state dimension: 512
Encoder / decoder layers: 1 LSTM layer each
Dropout: 0.1 on LSTM layers (disabled when there is only one layer internally)
Max source length: 128 tokens
Max target length: 192 tokens (including BOS/EOS)
Batch size: 16
Epochs: 5
Optimizer: Adam with learning rate $3 \times 10^{-4}$
Loss function: Cross-entropy loss over decoder outputs, with the pad token ID ignored
Gradient clipping: global norm clipped at 1.0 to stabilize training
Device: GPU when available, else CPU
At each epoch, the model is trained on all mini batches from the training loader and evaluated on the validation loader. Validation loss is computed per non-padded target token, analogous to the GPT-2 baseline. The checkpoint with the lowest validation loss is saved to lstm_attention_best.pt and used for subsequent generation and evaluation.

### 5.2.3 Results

**Validation Evaluation Summary:**

corpus_bleu: 0.1219
avg_perplexity: 9.5241
bert_precision: 0.6721
bert_recall: 0.6222
bert_f1: 0.6456
rouge1: 0.4509
rouge2: 0.1952
rougeL: 0.3054
keyword_strict_accuracy: 0.0672
keyword_avg_percentage: 42.9269
The LSTM-Attention model demonstrates noticeable improvements over the GPT-2 baseline in several key areas. It achieves higher lexical alignment and phrase-level similarity, reflected in improved BLEU (0.1219) and ROUGE scores (ROUGE-2 = 0.1952, ROUGE-L = 0.3054). The model also produces more semantically coherent stories, with a stronger BERTScore F1 (0.6456) and significantly lower perplexity (9.52), indicating more fluent and confident generation. However, the architecture struggles with enforcing keyword usage, achieving only 6.7% strict keyword accuracy and 42.9% keyword coverage, suggesting that the model often ignores required

constraints. Overall, LSTM-Attention generates more fluent and structurally aligned stories than GPT-2 but performs poorly in controlled generation tasks where keyword adherence is essential.

## 5.3 Flan-T5 + LoRA

Flan-T5 is a sequence-to-sequence Transformer model trained using large-scale instruction-tuning. Its encoder–decoder architecture makes it well suited for prompt-conditioned generation tasks such as story completion. In this project, Flan-T5 was fine-tuned using the LoRA (Low-Rank Adaptation) method to reduce computational cost while allowing efficient adaptation to the Tiny Stories dataset.

### 5.3.1 Architecture Description

Flan-T5 follows the standard T5 design:

Encoder: Processes the combined prompt containing keywords and the story beginning.
Decoder: Generates the continuation autoregressively, attending to encoder output through cross-attention.
Text-to-text paradigm: Every input and output is treated as text, making formatting consistent across tasks.
LoRA modules were injected into the attention layers, enabling training of only low-rank update matrices while keeping the original model weights frozen. This dramatically reduces trainable parameters and training time.

### 5.3.2 Key Training Settings

Model: Flan-T5-base
Fine-tuning method: LoRA
LoRA rank (r): 16
LoRA α: 32
Batch size: Train: 6, Validation: 4
Epochs: 3
Optimizer: AdamW
Learning rate: $2 \times 10^{-4}$
Input format: Keywords + story prompt + target story as a combined sequence
Output: Full story continuation

### 5.3.3 Results

Evaluation logs from the inference pipeline show the following:

BLEU: Very low
Perplexity: ~17.4
BERTScore F1: Moderate semantic similarity (≈ 0.62–0.67 range)
ROUGE: Weak lexical overlap
Keyword strict accuracy: ~0.72
Keyword partial percentage: ~0.89
Although Flan-T5 generated coherent and grammatically well-formed narratives, it struggled with strong keyword adherence and failed to show large metric gains relative to simpler models.

## 5.4 GNN + SLM Story Generator

This hybrid architecture augments a Small Language Model (GPT-2-based) with structured graph embeddings derived from story keywords. The intuition is that keyword relationships—when encoded as a graph—could provide additional semantic structure to guide story generation.

### 5.4.1 Architecture Description

The model consists of four primary components:

1. Graph Construction Module: Builds a graph from story keywords and extracted entities. Node features encode keyword properties.
2. Graph Neural Network (GNN) Encoder: 3-layer GNN with 4 attention heads. Computes graph embeddings capturing relational structure.
3. Fusion Layer: Projects GNN embedding into the GPT-2 hidden space. Converts the representation into 16 soft-prompt tokens prepended to the input sequence.
4. SLM Decoder (GPT-2): Takes prefix tokens + prompt + retrieved examples. Generates the full story autoregressively.

**5.4.2 Retrieval Module**

A k-nearest-neighbor retrieval system selects k = 3 similar stories from the training set to provide contextual examples. These examples are concatenated with the prompt to give the decoder more narrative guidance.

**5.4.3 Key Training Settings**

Base LM: GPT-2 small
Node feature dimension: 390
GNN hidden dimension: 256
Prefix length: 16 tokens
Batch size: 4 graphs per batch
Epochs: up to 10 with early stopping (patience = 5)
Optimizer: AdamW
LR: $5 \times 10^{-5}$ with warmup + cosine decay
Gradient clipping: 0.5

**5.4.4 Results**

Despite architectural sophistication, results showed:

BLEU: Low, similar to baseline
Perplexity: High
ROUGE: No measurable improvement
BERTScore: Moderate
Keyword metrics: Weak

Adding graph structure and retrieval did not translate into better metric performance, indicating that simple SLM-only models were equally or more effective.

## 5.5 Large Language Model (LLM): Mistral-7B with LoRA Fine-Tuning

The final architecture evaluated in this study is a large language model (LLM), specifically Mistral-7B, fine-tuned using Low-Rank Adaptation (LoRA). Unlike GPT-2 and other small models, Mistral-7B is an instruction-tuned transformer that incorporates extensive pretraining on diverse text corpora, enabling stronger reasoning, fluency, and narrative coherence.

**5.5.1 Architecture Description**

Mistral-7B is a decoder-only transformer based on the next-token prediction objective. Key architectural features include:

Multi-head self-attention with grouped-query attention, improving inference efficiency.
Sliding window attention, enabling handling of long contexts.
Instruction tuning, allowing the model to better follow structured prompts.

For this project, the model was adapted for story generation using LoRA, which inserts low-rank trainable matrices into attention layers while keeping the original pretrained weights frozen. This approach significantly reduces computational cost and memory usage, making it feasible to fine-tune a 7B model on moderate hardware.

### 5.5.2 Input Formatting and Training Design

Mistral was fine-tuned in a causal-language-modeling setup. Each training sample concatenated:

A structured instruction (e.g., "You are a children's story writer…")
The list of required keywords
The story-beginning prompt
The full gold story (as training target)
The combined text was tokenized with the Mistral tokenizer, using the EOS token for padding.

Training configuration:

LoRA parameters: r = 16, α = 32
Optimizer: AdamW
Learning rate: $2 \times 10^{-4}$
Batch size: 6 (train), 4 (validation)
Epochs: 3
Loss: Causal language modeling loss

### 5.5.3 Results

Mistral-7B produced the most fluent and human-like stories among all approaches. Its narratives displayed strong coherence, logical progression, and natural phrasing. However, evaluation metrics revealed several limitations:

Keyword strict accuracy: 72%
Keyword partial accuracy: 89%
Perplexity: ~17.4 (higher than smaller models)
BLEU and ROUGE: Low
BERTScore: Moderate, but not superior to LSTM-Attention
Overall, the LLM excelled in qualitative story quality, but did not outperform smaller models on quantitative evaluation metrics. The results indicate a trade-off: strong expressive ability but weaker structural alignment and constraint satisfaction when evaluated against reference stories.

# 6. Training Procedure

All models were trained individually using the cleaned Tiny Stories dataset described earlier. The training process followed these steps:

## 6.1 Dataset Processing

Used normalized stories, extracted keywords, and a 10-word story beginning prompt.
Split into train/validation/test (80/10/10).
Keywords and prompts were inserted into template instructions for conditional generation.

## 6.2 Model-Specific Training Pipelines

**GPT-2 Baseline:**

Sequence length: 192
Epochs: 5
Batch size: 2
Adam optimizer (3e-5)
Causal Language Modeling loss

**LSTM–Attention Seq2Seq:**

Encoder length: 128, decoder length: 192
Epochs: 5
Batch size: 16
Adam (3e-4) with gradient clipping
Teacher forcing during training

**Flan-T5 LoRA:**

LoRA adapters applied to attention layers
Epochs: 3
Batch size: 6
Learning rate: 2e-4
Full text-to-text sequence training

**GNN-Retrieval + SLM:**

Graph construction per sample
Forward pass through GNN → prefix tokens → GPT-2
Retrieval module active during inference
AdamW, LR 5e-5 with warmup

**Mistral-7B LoRA:**

Full causal LM training with LoRA (r=16, α=32)
Batch size: 6
Epochs: 3
Optimized with AdamW

### 6.3 Checkpointing

Each model saved the checkpoint with lowest validation loss for inference and evaluation.

## 7. Results

A summary of each model's performance across the evaluation metrics:

### 7.1 GPT-2 Baseline

BLEU: 0.0829
Perplexity: 13.55
BERT F1: 0.6132
ROUGE-1: 0.4313
Keyword strict: 53%
Keyword partial: 81%

### 7.2 LSTM–Attention

BLEU: 0.1219 (highest)
Perplexity: 9.52 (best fluency)
BERT F1: 0.6456
ROUGE-2: 0.1952
Keyword strict: 6.7% (worst)
Keyword partial: 42.9%

### 7.3 Flan-T5 LoRA

Perplexity: ~17.4 (highest → least confident)
Keyword strict: 72%

Keyword partial: 89%
ROUGE & BLEU: Low
Semantic similarity: moderate

### 7.4 GNN-Retrieval + SLM

No metric showed improvement vs. baseline
Perplexity remained high
BLEU and ROUGE stayed low
Keyword adherence did not improve

### 7.5 Mistral-7B LoRA

Keyword strict: 72%
Keyword partial: 89%
Semantic coherence: high
Perplexity: ~17.4
ROUGE: weak
Better narrative quality than GPT-2 but not better metric alignment

## 8. Challenges & Limitations

### 8.1 Keyword Constraint Enforcement

Even powerful models struggled to incorporate all required keywords consistently. Seq2Seq models especially showed poor adherence.

### 8.2 Low BLEU and ROUGE Scores

All models showed weak lexical overlap with reference stories due to high variability in narrative rephrasing.

### 8.3 High Perplexity for Larger Models

Flan-T5 and Mistral-7B exhibited high perplexity, indicating difficulty predicting next tokens despite strong language capabilities.

### 8.4 GNN + Retrieval Did Not Improve Results

Despite added structure, the hybrid approach showed no meaningful gains. Possible causes:

Graphs too small to add semantic value
Prefix tokens not strongly influencing generation

### 8.5 Dataset Variability

Stories vary significantly in style and structure, making strict metric alignment difficult.

## 9. Key Findings

LSTM-Attention produced the best fluency (lowest perplexity) and highest BLEU/ROUGE among all models.
Mistral-7B LoRA and Flan-T5 LoRA generated the most coherent stories, but did not perform well on lexical metrics.
GPT-2 provided a reasonable baseline with balanced performance across metrics.
GNN + SLM did not outperform simpler architectures, showing that added structure does not always improve generation.
Keyword adherence remains the hardest challenge, even for large LLMs.

## 10. Conclusion

This project compared multiple neural architectures for story generation and found that no single model excelled across all dimensions. While LSTM-Attention achieved the strongest BLEU, ROUGE, and perplexity scores, it failed at consistent keyword integration. Large models such as Mistral-7B and Flan-T5 produced fluent and coherent stories but did not achieve strong metric alignment or constraint satisfaction.

Structured approaches like GNN-augmented generation did not produce the expected improvements, suggesting that story generation may require different forms of conditioning beyond simple keyword graphs.

Overall, the findings indicate that controllable story generation remains an open research problem, especially when balancing fluency, narrative structure, and hard constraints. Future work should explore reinforcement learning, stronger keyword-penalty mechanisms, and more expressive retrieval strategies.