# Quantifying Creativity in LLMs

`#creativity`, `#evaluation`, `#metric`, `#LLMs`

Abeen Bhattacharya

## Abstract

The quantification of creativity in large language models (LLMs) is essential given their increasing influence across domains such as poetry, technical writing, and storytelling. In this project, we introduce a novel Creativity Index (CI) that integrates measures of statistical novelty, semantic coherence, and contextual appropriateness to assess the creativity of LLM-generated text. In addition, we develop four neural models that combine these metrics with raw textual input and train them on a dataset of 2,002 contrastive pairs (1,063 literary and 939 technical) judged by GPT-4. This evaluation framework examines whether the metrics we use in constructing the CI align with LLMs' own conceptualization of creativity.

---

## What This Project Is About

Large language models (LLMs) generate outputs ranging from sonnets to technical abstracts and narratives steeped in magical realism. Yet, a standardized metric for assessing creativity has been lacking. We address this gap by pursuing two primary objectives:

1. **Developing a Quantitative Metric for Creativity:**
   We create a composite Creativity Index (CI) by integrating multiple metrics (novelty, coherence, contextual fit, syntactic complexity, lexical diversity, lexical novelty, surprise, emotional expressiveness, and human-likeness).
2. **Understanding Creativity in LLMs:**
   We investigate whether these metrics correspond to the way LLMs “perceive” creativity by training neural models on contrastive pairs (judged by GPT-4) and comparing their scores with our fixed CI.

*Text is segmented using SpaCy, and our models are trained contrastively using margin ranking loss.*

---

## Progress Made So Far

- **Dataset Creation:**
    - **Contrastive Pairs:** A total of **2,002** contrastive pairs were created: **1,063** pairs from the literary domain and **939** from the technical domain.
    - **Dataset Creation Process:**
        - **Prompt Generation:** Using a GPT-2–based prompt generator, we sampled 10 creative prompts per domain from baseline corpora (Project Gutenberg for literary and arXiv abstracts for technical).
        - **Text Generation:** For each prompt, two texts were generated using GPT-4/OpenAI.
        - **Contrastive Judgment:** GPT-4 was then used to determine which text was "more creative."
        - **Metric Computation:** All nine creativity metrics were computed for each segment of both texts.
        - **Data Storage:** The contrastive pairs (text segments and their metric vectors) were stored for training.
- **Model Training:**
  Four neural models were developed:

    i. **CompositeRegressor:** Combines DistilBERT embeddings with an LSTM on metric features.
    ii. **SimpleCreativityPredictor:** Uses DistilBERT embeddings processed by an LSTM (text-only).
    iii. **TransformerCreativityAggregator:** Uses a transformer encoder to combine metric interactions with text embeddings.
    iv. **TextBasedCreativityPredictor:** Uses DistilBERT to generate weights that re-calculate the CI formula.

  Training used a contrastive learning approach with a Margin Ranking Loss.
- **Evaluation:**
  Automated testing was conducted using fixed test prompts per domain. Generated texts were evaluated with our Fixed CI as well as the scores from each neural model. Additional graphical analyses (scatter plots, histograms, and heatmaps) were produced.

## Approach

### Main Approach

Creativity is quantified using nine normalized metrics:

1. **Novelty (N)**
   KL divergence between token distributions:

   ```
   D_KL(P || Q) = Σ P(t) * log( P(t) / Q(t) )
   ```

2. **Coherence (C)**
   Inverse GPT-2 perplexity:

   ```
   C = 1 / ( e^(loss) + 10^-6 )
   ```

3. **Contextual Fit (CT)**
   Sentence-BERT cosine similarity:

   ```
   CT = (A - B) / (||A|| * ||B||)
   ```

4. **Syntactic Complexity (SC)**
   Mean of scaled sub-metrics (e.g., sentence length, clause count, parse tree depth, POS entropy).

5. **Lexical Diversity (LD)**
   Scaled weighted mean of n-gram diversity and inverted Self-BLEU.

6. **Lexical Novelty (LN)**
   Proportion of unique n-grams ($n \geq 5$) not in a baseline corpus, computed via DJ Search.

7. **Surprise (S)**
   Mean cosine distance between consecutive sentence embeddings.

8. **Emotional Expressiveness (EE)**
   Variance of VADER sentiment scores per sentence.

9. **Human-Likeness (HL)**
   Exponentiated GPT-2 perplexity or a MAUVE score capturing overall text plausibility.

---

These metrics are then aggregated in two ways:

1. **Fixed CI**
   A simple weighted sum of normalized metrics:

   ```
   CI = Σ [w_i * norm(m_i)]    where Σ w_i = 1   (default w_i = 0.111)
   ```

2. **Neural Models**

    - **CompositeRegressor**
      Uses DistilBERT + LSTM + MLP for a hybrid text-and-metrics input.
    - **SimpleCreativityPredictor**
      A text-only baseline with DistilBERT + LSTM.
    - **TransformerCreativityAggregator**
      DistilBERT + a small Transformer (4 layers, 4 heads) to learn interactions among the metrics.
    - **TextBasedCreativityPredictor**
      DistilBERT + MLP that dynamically learns metric weights.

Text is segmented with SpaCy, and the four neural models are trained contrastively using margin ranking loss on GPT-4O judgments.

### Baselines

The fixed CI (equal weights) serves as the baseline, compared against neural model scores.

### Dataset Creation

The training dataset of 2,002 contrastive pairs was created as follows:

1. **Prompt Generation**: 10 imaginative prompts per domain (literary: e.g., "Quantum sonnet"; technical: e.g., "Encryption abstract") were crafted using GPT-2 and random sampling from baseline corpora. One prompt was selected per iteration.
2. **Response Generation**: For each prompt, GPT-4o generated two responses.
3. **Judgment**: GPT-4o judged which response was more creative based on its internal understanding.
4. **Metric Computation**: All nine metrics were computed for each response against Project Gutenberg (literary) or arXiv abstracts (technical).
5. **Storage**: Judgments, text pairs, and metrics were stored in .pkl files (1,063 literary, 939 technical pairs loaded from existing files).

# Experiments

### Data

- **Datasets**:
    - **Project Gutenberg**: ~5.5M words of literary texts for poetic/narrative creativity.
    - **arXiv Abstracts**: ~1M words of technical papers for scientific innovation.

- **Task**: Assess creativity in GPT-4 and Together AI outputs across literary and technical prompts.

### Evaluation Method

- **Metrics:** Fixed CI, neural model scores, Pearson/Spearman correlations between model predictions and CI/GPT-4o judgments.
- **Analysis**: Compared fixed CI to model scores and evaluated Together AI model-specific strengths.

### Experimental Details

- **Setup**: Python 3.11, Google Colab, PyTorch 2.x, Tesla T4 GPU (16 GB).
- **Hyperparameters**:
  - **CompositeRegressor**: LSTM (hidden=64) + MLP, lr=1e-3, batch size=8, AdamW (weight decay=1e-5), 50 epochs.
  - **SimpleCreativityPredictor**: LSTM (hidden=64) + MLP, lr=1e-3, batch size=8, AdamW, 50 epochs.
  - **TransformerCreativityAggregator**: Transformer (4 layers, 4 heads, hidden=64), lr=1e-3, batch size=8, AdamW, 50 epochs.
  - **TextBasedCreativityPredictor**: MLP (3 layers, hidden=64), lr=2e-5, batch size=8, AdamW, 100 epochs.
- **Generation**: Temperature=0.8, top-p=0.95, max_tokens=200.
- **Pairs**: 2,002 pairs (1,063 literary, 939 technical), split 80% train, 20% validation.

## Results

## Training Loss Trends

When training with only 500 pairs the models achieved significantly lower (i.e. “better”) best validation losses in fewer epochs, but when we increased the dataset size the best validation losses became worse and the convergence was more challenging. This suggests that even LLM-based evaluators may not be completely objective when judging creativity across larger, more diverse contrastive pair sets.

## Trends (2002 Pairs)

| Model | Best Val Loss | Epoch | Train Loss |
|---|---|---|---|
| **SimpleCreativityPredictor** | 0.8390 | 19 | 0.7725 |
| **CompositeRegressor** | 0.8332 | 19 | 0.7725 |

| Model | Best Val Loss | Epoch | Train Loss |
|---|---|---|---|
| **TransformerCreativityAggregator** | 0.7906 | 28 | 0.6571 |
| **TextBasedCreativityPredictor** | 0.8340 | 14 | 0.7362 |

> Note:
>
> - *SimpleCreativityPredictor* and *TextBasedCreativityPredictor* each eventually stopped early around those epochs, with the listed Val Loss being the best they achieved (even though some logs showed epochs continuing, the final "best" was reached by the epoch noted).
> - The *CompositeRegressor* and *SimpleCreativityPredictor* both happened to converge best at epoch 19 (though on different runs/logs), showing the same final training loss (0.7725) but slightly different best Val Losses (0.8332 vs. 0.8390).
> - *TransformerCreativityAggregator* improved more gradually up through epoch 28, achieving the lowest Val Loss of the four on this larger (2002) dataset.
> - The above training trends was taken such that the least validation loss was reached, across 5 seeds of training

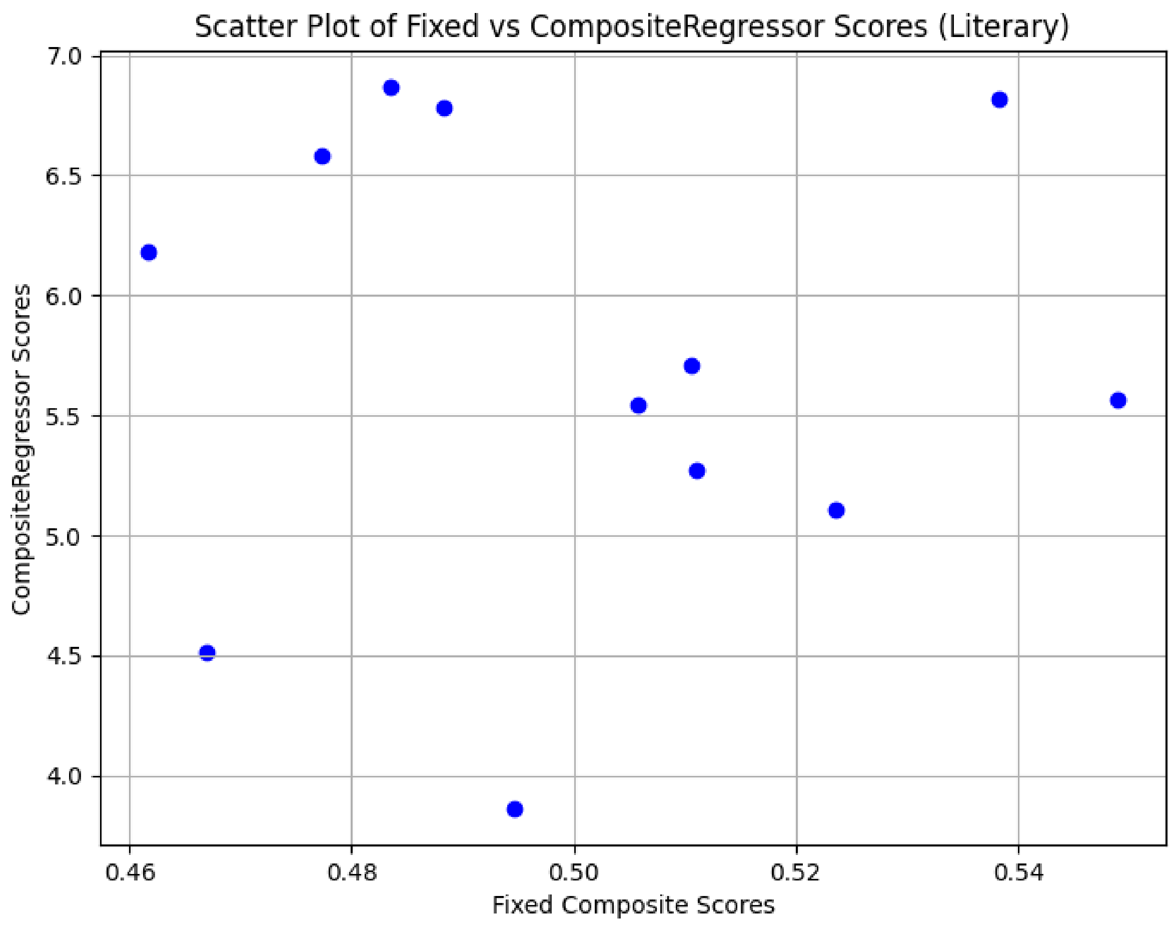
## Trends (500 Pairs)

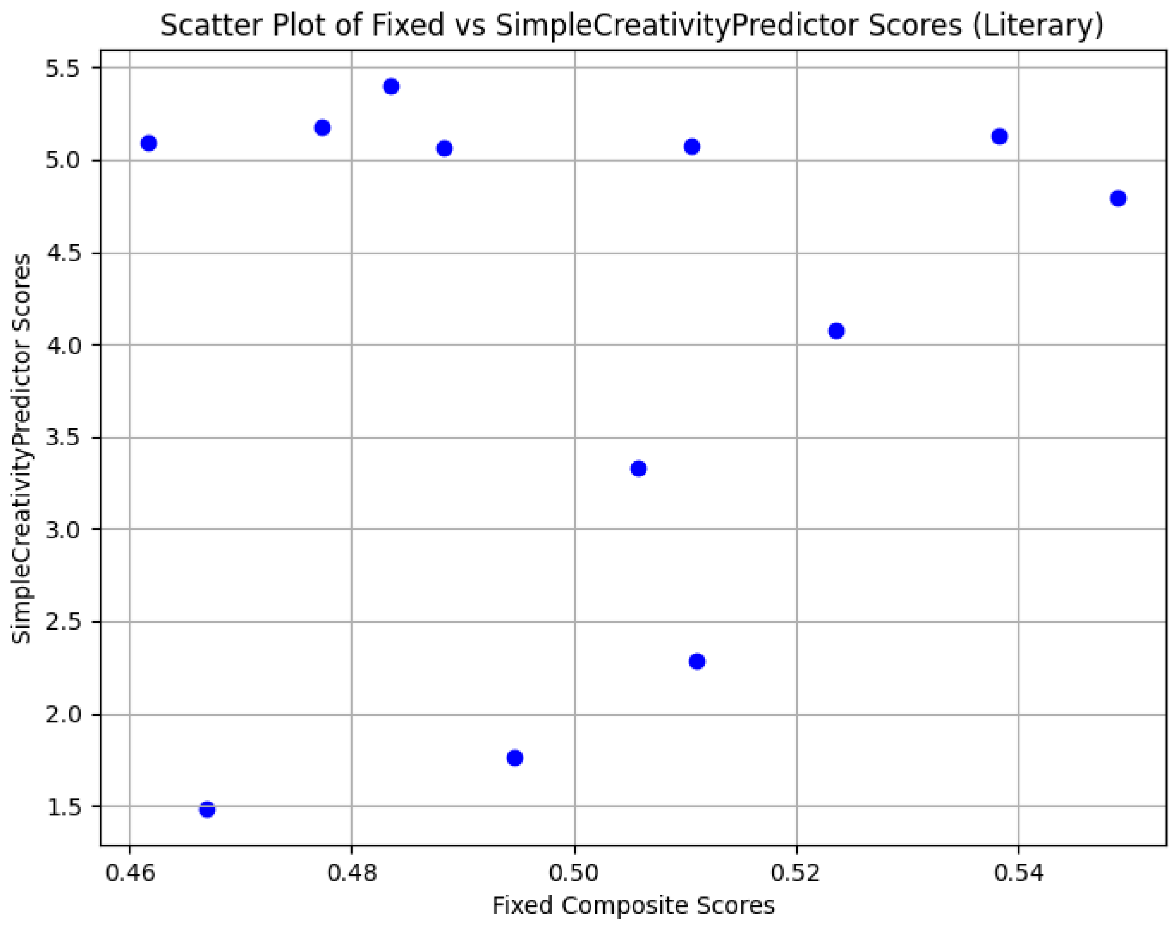
| Model | Best Val Loss | Epoch | Train Loss |
|---|---|---|---|
| **SimpleCreativityPredictor** | 0.7423 | 17 | 0.7222 |
| **CompositeRegressor** | 0.6259 | 14 | 0.6174 |
| **TransformerCreativityAggregator** | 0.5835 | 24 | 0.5956 |
| **TextBasedCreativityPredictor** | 0.6183 | 39 | 0.6139 |

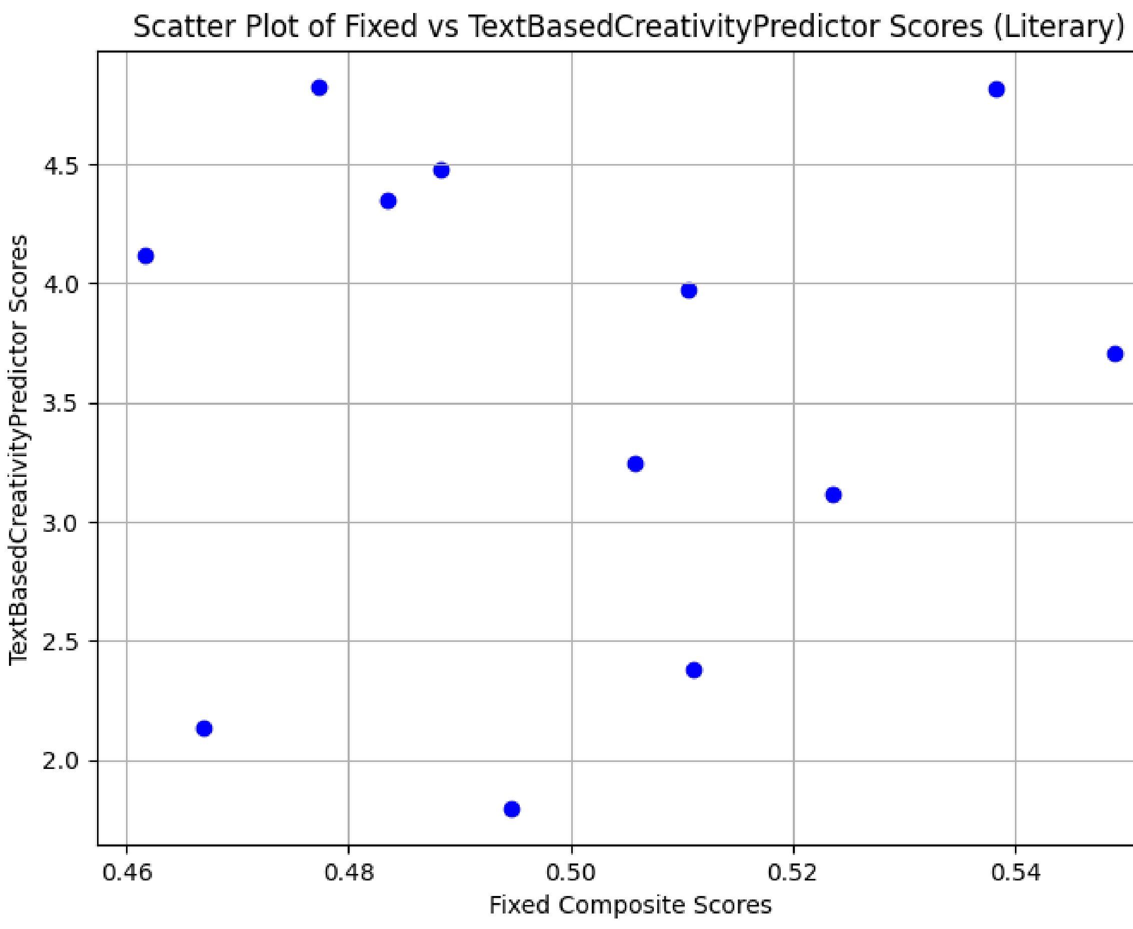
---

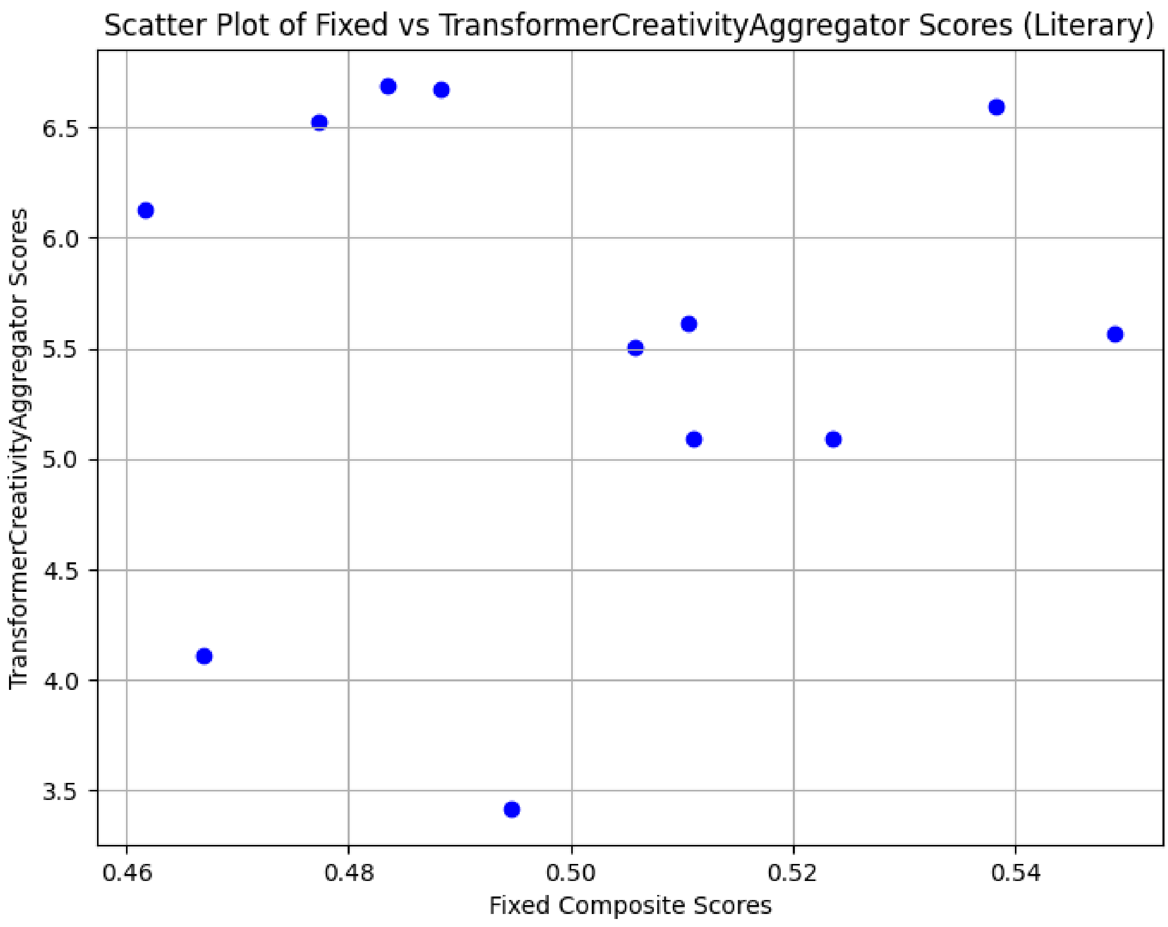
### Testing Results

#### Literary Domain


Scatter Plot of Fixed vs CompositeRegressor Scores (Literary)
CompositeRegressor Scores
Fixed Composite Scores
7.0
6.5
6.0
5.5
5.0
4.5
4.0
0.46
0.48
0.50
0.52
0.54


Scatter Plot of Fixed vs SimpleCreativityPredictor Scores (Literary)
SimpleCreativityPredictor Scores
Fixed Composite Scores
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0.46
0.48
0.50
0.52
0.54


Scatter Plot of Fixed vs TextBasedCreativityPredictor Scores (Literary)
TextBasedCreativityPredictor Scores
Fixed Composite Scores
4.5
4.0
3.5
3.0
2.5
2.0
0.46
0.48
0.50
0.52
0.54


Scatter Plot of Fixed vs TransformerCreativityAggregator Scores (Literary)
TransformerCreativityAggregator Scores
Fixed Composite Scores
6.5
6.0
5.5
5.0
4.5
4.0
3.5
0.46
0.48
0.50
0.52
0.54

The table below presents the top performances of different model types in the literary domain, focusing on prompts such as "Quantum Sonnet," "Short Story," and "Magical Realism." Each row reflects the best score achieved by a specific model type for a given prompt, offering a clear view of their effectiveness in literary creativity.

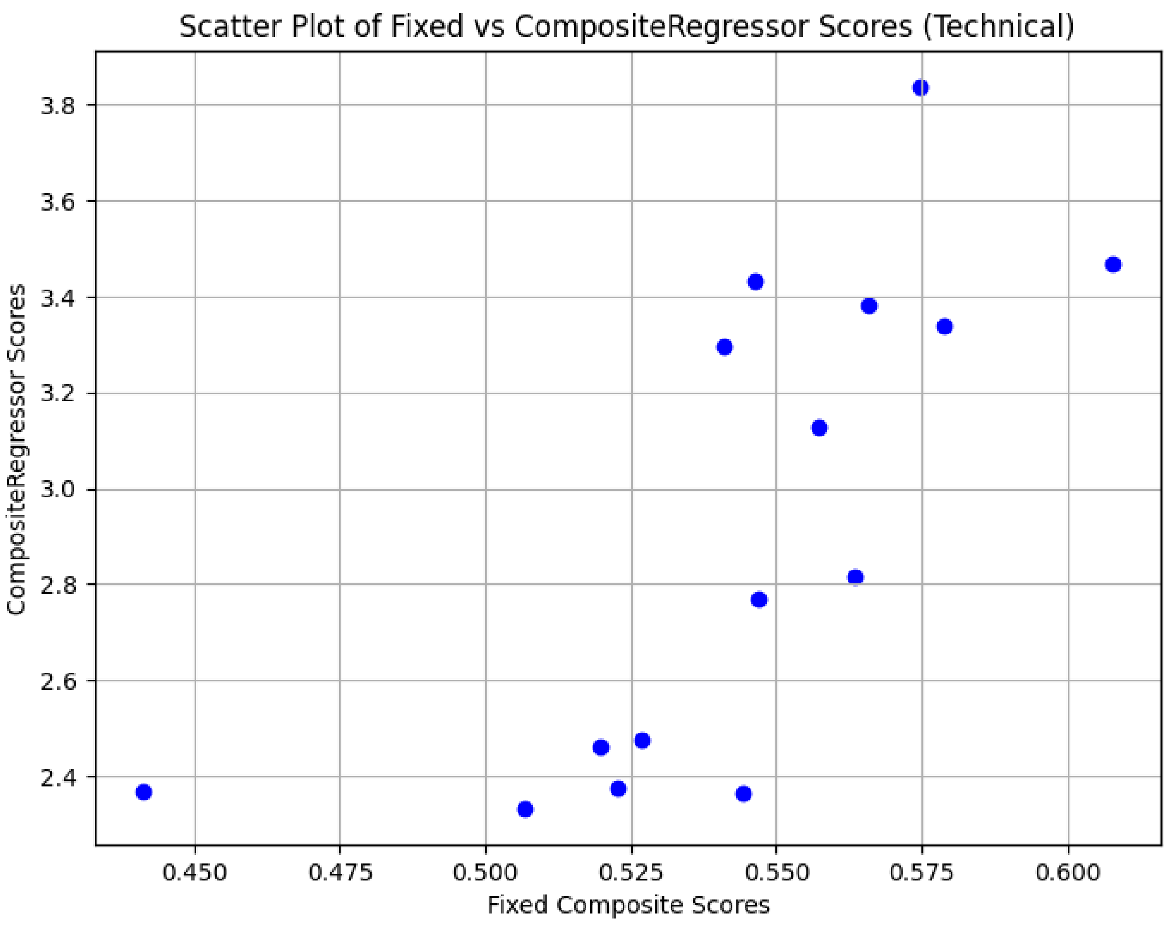
| Prompt | Fixed CI | Best Model Score | Model Type |
|---|---|---|---|
| **Quantum Sonnet** | 0.5152 | 9.92 | CompositeRegressor |
| | 0.5103 | 9.67 | TransformerCreativityAggregator |
| | 0.4596 | 5.22 | SimpleCreativityPredictor |
| | 0.4269 | 0.71 | TextBasedCreativityPredictor |
| **Short Story** | 0.4345 | 5.33 | CompositeRegressor |
| | 0.4164 | 4.16 | TransformerCreativityAggregator |
| | 0.3808 | 0.99 | SimpleCreativityPredictor |
| | 0.3299 | 0.40 | TextBasedCreativityPredictor |
| **Magical Realism** | 0.4203 | 6.65 | CompositeRegressor |
| | 0.4087 | 6.10 | TransformerCreativityAggregator |
| | 0.3927 | 4.15 | SimpleCreativityPredictor |
| | 0.3705 | 0.50 | TextBasedCreativityPredictor |

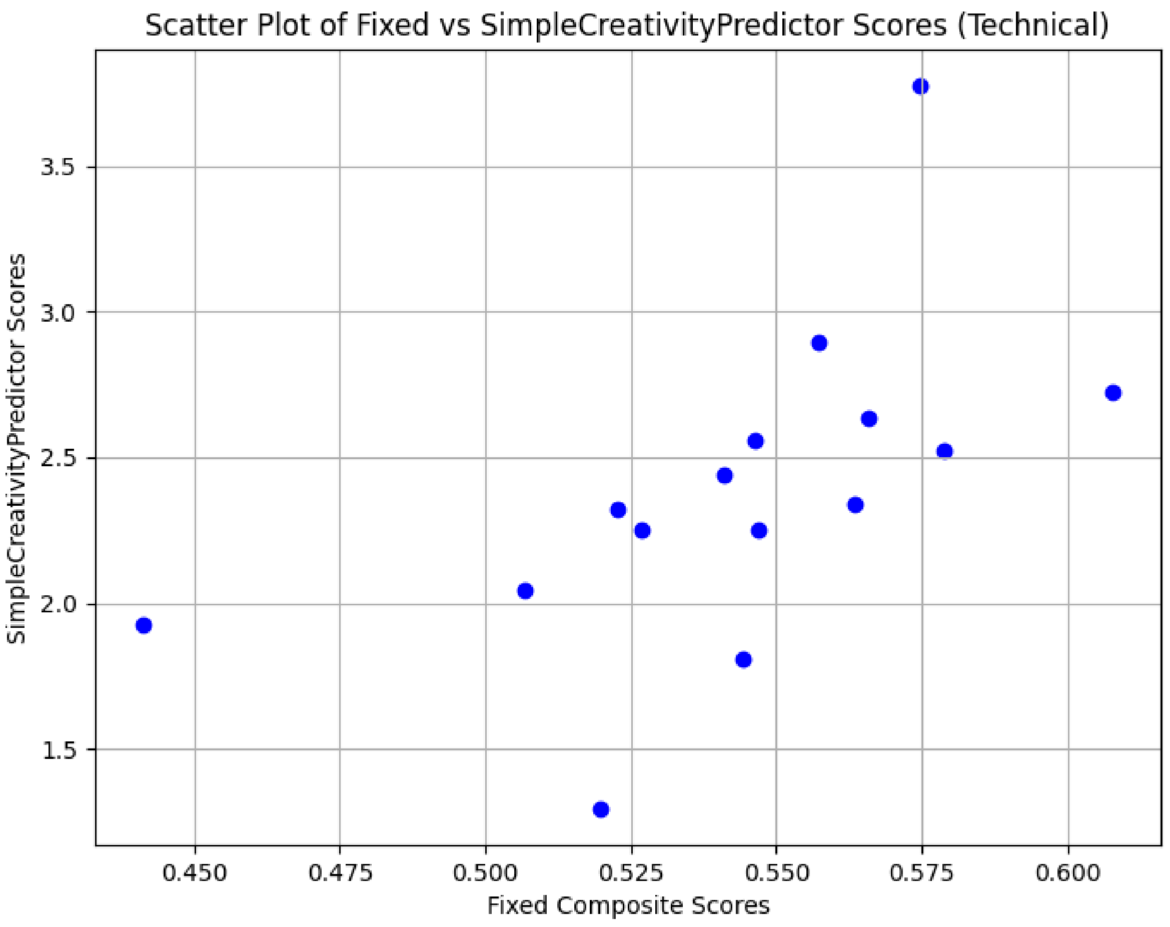
**Observations (Literary):**

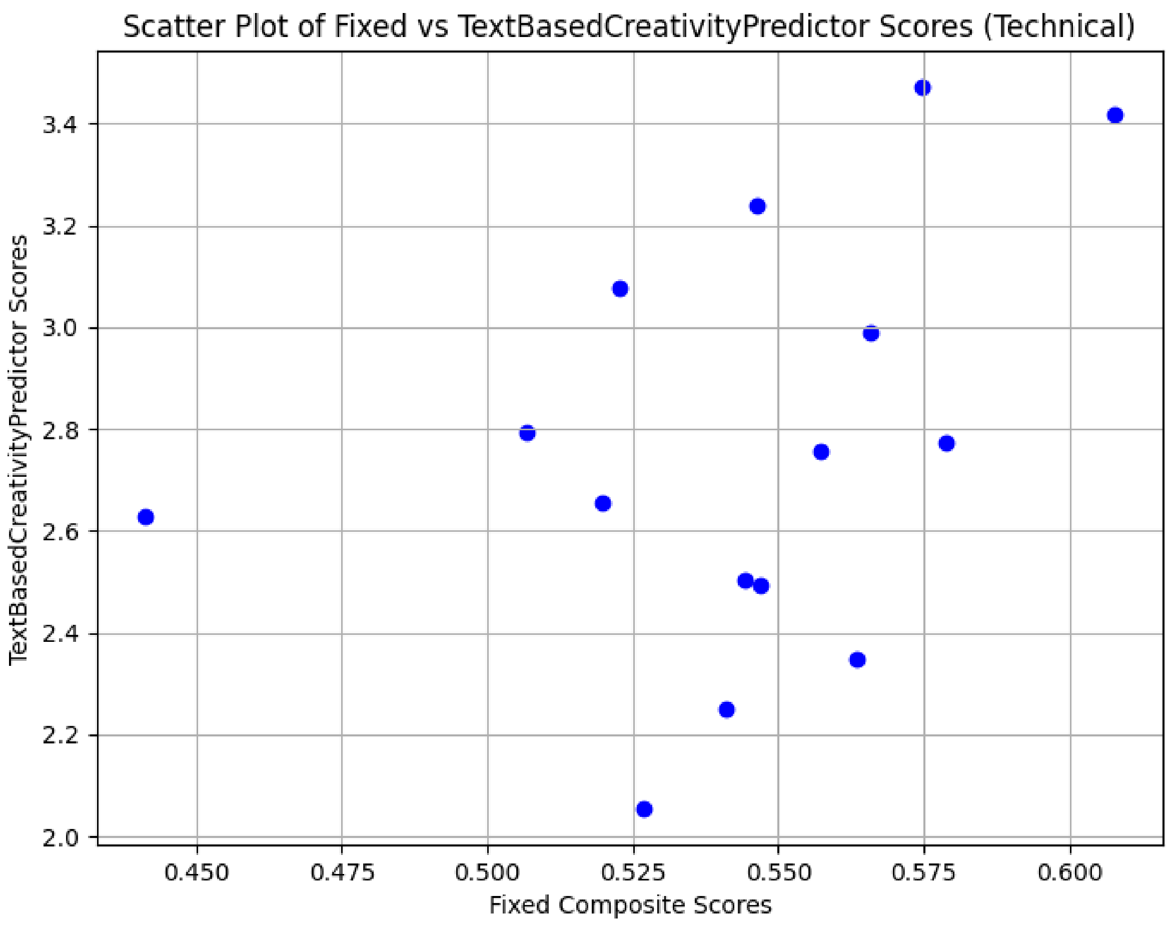
- **CompositeRegressor**:
  Achieved the highest score for “Quantum Sonnet” (9.92) and delivered strong performance across all prompts.
- **TransformerCreativityAggregator**:
  Performed competitively with high scores, especially on imaginative prompts like “Magical Realism.”
- **SimpleCreativityPredictor** and **TextBasedCreativityPredictor**:
  Produced lower and less consistent scores, indicating their limitations in capturing the full spectrum of literary creativity.

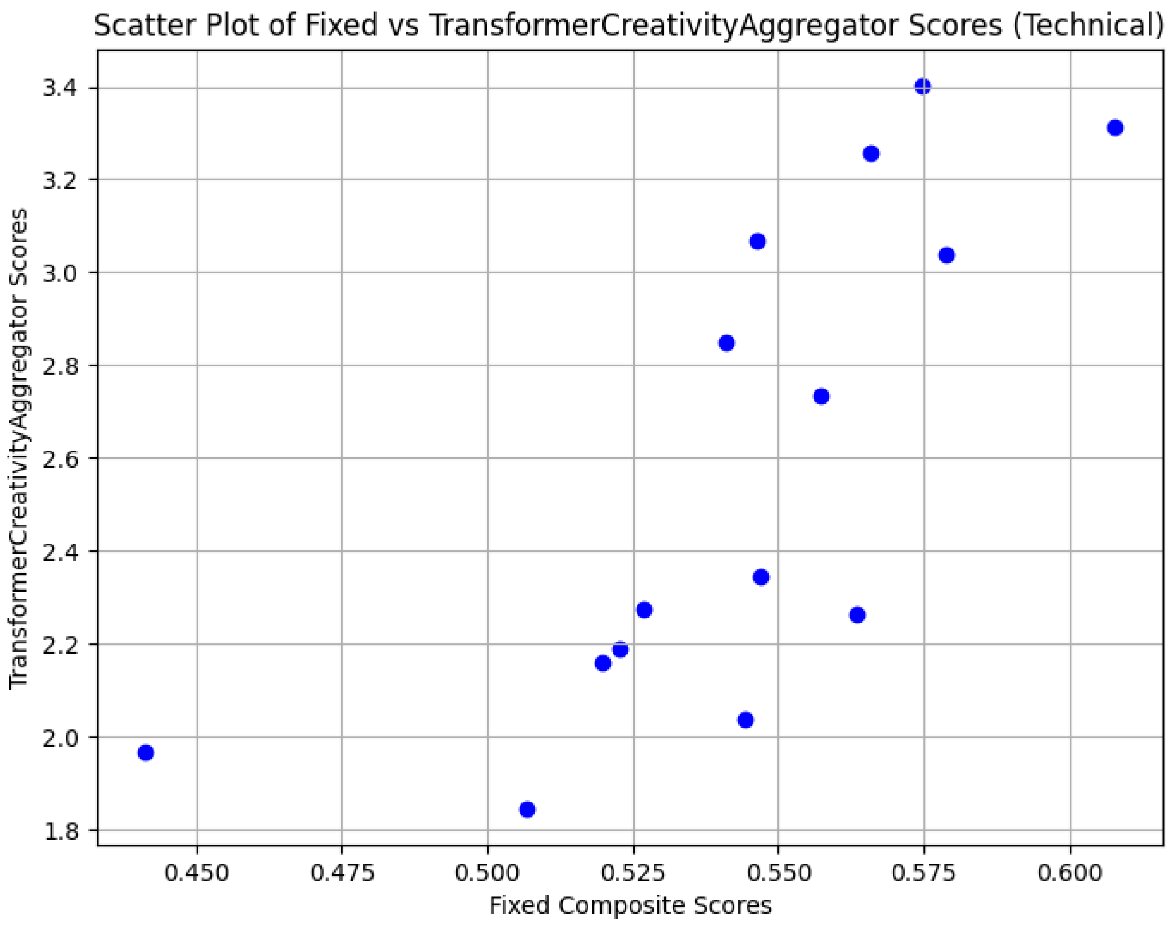
---

**Technical Domain**


Scatter Plot of Fixed vs CompositeRegressor Scores (Technical)
CompositeRegressor Scores
Fixed Composite Scores
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
0.450
0.475
0.500
0.525
0.550
0.575
0.600


Scatter Plot of Fixed vs SimpleCreativityPredictor Scores (Technical)
SimpleCreativityPredictor Scores
Fixed Composite Scores
3.5
3.0
2.5
2.0
1.5
0.450
0.475
0.500
0.525
0.550
0.575
0.600


Scatter Plot of Fixed vs TextBasedCreativityPredictor Scores (Technical)
TextBasedCreativityPredictor Scores
Fixed Composite Scores
3.4
3.2
3.0
2.8
2.6
2.4
2.2
2.0
0.450
0.475
0.500
0.525
0.550
0.575
0.600


Scatter Plot of Fixed vs TransformerCreativityAggregator Scores (Technical)
TransformerCreativityAggregator Scores
Fixed Composite Scores
3.4
3.2
3.0
2.8
2.6
2.4
2.2
2.0
1.8
0.450
0.475
0.500
0.525
0.550
0.575
0.600

The table below highlights the top performances of each model type in the technical domain, focusing on prompts like "AI Abstract," "Encryption Abstract," and "Cloud Architecture." Each row shows the best score for a specific model type per prompt, providing insights into their technical creativity strengths.

| Prompt | Fixed CI | Best Model Score | Model Type |
|---|---|---|---|
| **AI Abstract** | 0.4574 | 6.40 | TransformerCreativityAggregator |
| | 0.3483 | 5.31 | CompositeRegressor |
| | 0.3465 | 2.99 | SimpleCreativityPredictor |
| | 0.2988 | 0.49 | TextBasedCreativityPredictor |
| **Encryption Abstract** | 0.4917 | 8.33 | TransformerCreativityAggregator |
| | 0.4761 | 7.43 | CompositeRegressor |
| | 0.4099 | 5.17 | SimpleCreativityPredictor |
| | 0.3678 | 0.58 | TextBasedCreativityPredictor |
| **Cloud Architecture** | 0.4538 | 6.40 | TransformerCreativityAggregator |
| | 0.4328 | 5.12 | CompositeRegressor |
| | 0.3601 | 0.96 | SimpleCreativityPredictor |
| | 0.3216 | 0.50 | TextBasedCreativityPredictor |

**Observations (Technical):**

- **TransformerCreativityAggregator**:
  Consistently delivered high scores, reflecting its ability to capture nuanced technical details.
- **CompositeRegressor**:
  Showed balanced performance, though slightly behind the Transformer model on innovative tasks.
- **SimpleCreativityPredictor** and **TextBasedCreativityPredictor**:
  Performed at a lower level, highlighting challenges in assessing technical creativity.

## Together AI Model Comparison

We tested multiple prompts spanning both **literary** and **technical** domains against several Together AI models---namely:

- **meta-llama/Llama-3.3-70B-Instruct-Turbo-Free**
- **mistralai/Mixtral-8x7B-Instruct-v0.1**
- **Qwen/Qwen2-72B-Instruct**

and measured each output's **Fixed CI** (our fixed creativity index) alongside the scores assigned by our trained creativity models (e.g., **CompositeRegressor**, **SimpleCreativityPredictor**, **TransformerCreativityAggregator**, **TextBasedCreativityPredictor**).

Below is a broad synthesis of the observed trends:

1. **Literary Prompts (e.g., "Write a sonnet about quantum physics" or "Generate a creative narrative in the style of magical realism"):**

    - **Fixed CI** values typically ranged between **0.45--0.58** across the different Together AI models.
    - **CompositeRegressor** scores, which often spiked for well-structured, thematically rich texts, hovered around **5.0--6.7** in some cases.
    - **Qwen** models often produced slightly higher CompositeRegressor scores on sonnets (e.g., **~6.47** or more), whereas **mistralai** sometimes had a higher *fixed* CI (e.g., **0.54--0.57** range).
    - On average, we see that the **TransformerCreativityAggregator** and **CompositeRegressor** give higher numeric scores for outputs that exhibit strong coherence, thematic adherence to the prompt (contextual fit), and lexical diversity (important for literary tasks).

2. **Technical Prompts (e.g., "Generate a technical description of a cloud computing architecture," "Write a research paper abstract on a new AI algorithm"):**

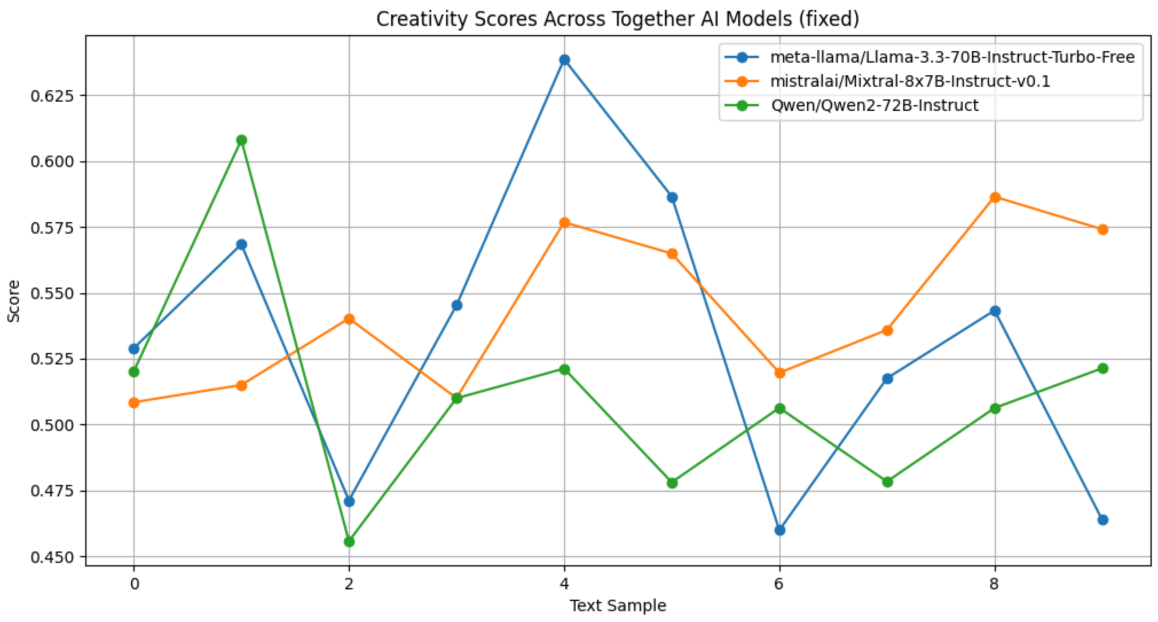
    - **Fixed CI** values here commonly landed in the **0.47--0.61** ballpark.
    - Interestingly, certain outputs with well-structured, precise technical language (e.g., from Qwen or Llama) scored relatively high on the fixed scale but had more moderate CompositeRegressor scores (in the **2--3.5** region). This is because the text is less "creative" in a literary sense yet still satisfies many of the metrics (like coherence and domain fit).
    - For some prompts, **Qwen** or **mistralai** gave expansions with quite thorough details on cloud computing layers or novel AI methods, achieving above-average **Fixed CI** (above **0.50** in many runs) and moderate to high model scores, especially if the text included novelty in describing the architecture or a distinctive approach to encryption.

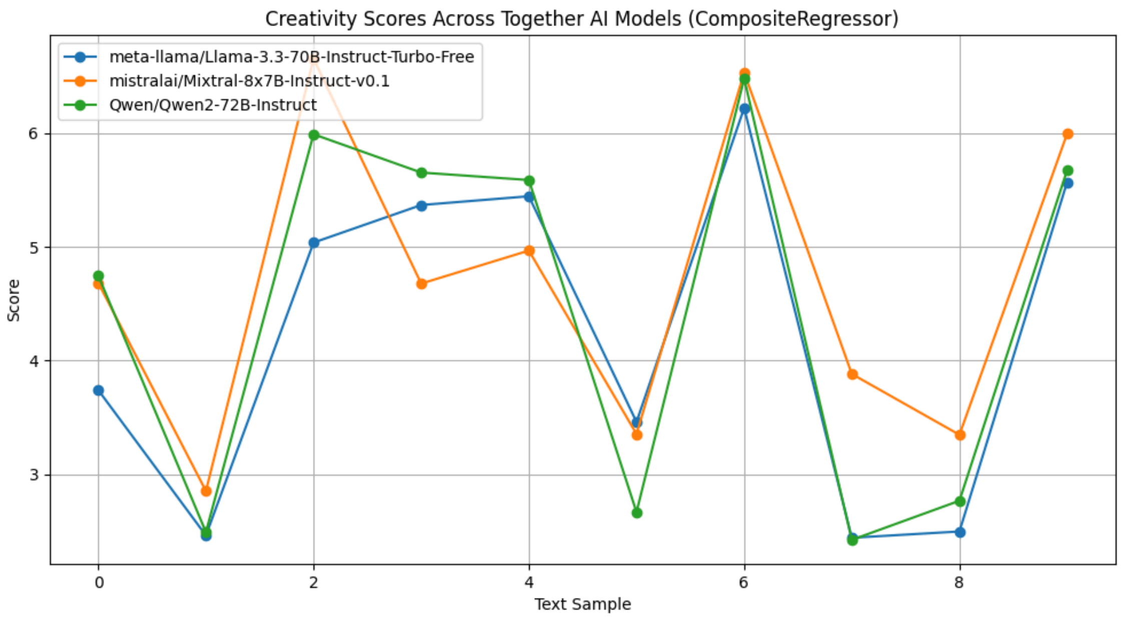
3. **Individual Max Scores & Overall Trends:**

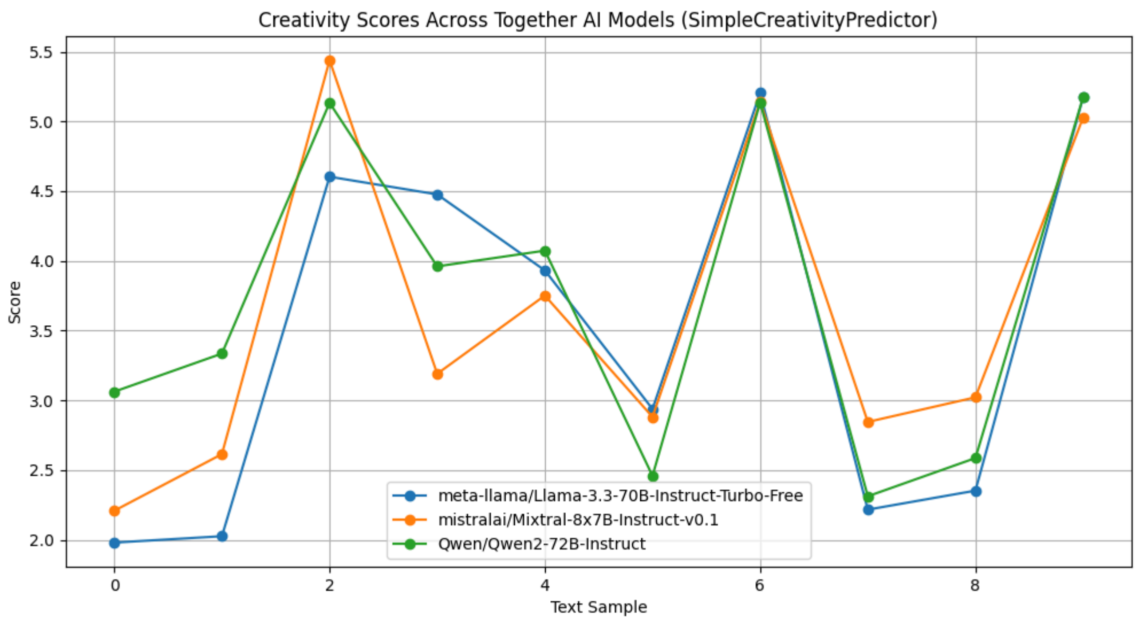
    - **Highest CompositeRegressor** spikes in literary tasks often appeared in Qwen or mistralai outputs that used vivid, imaginative language (particularly in the "sonnet" or "magical realism" prompts).

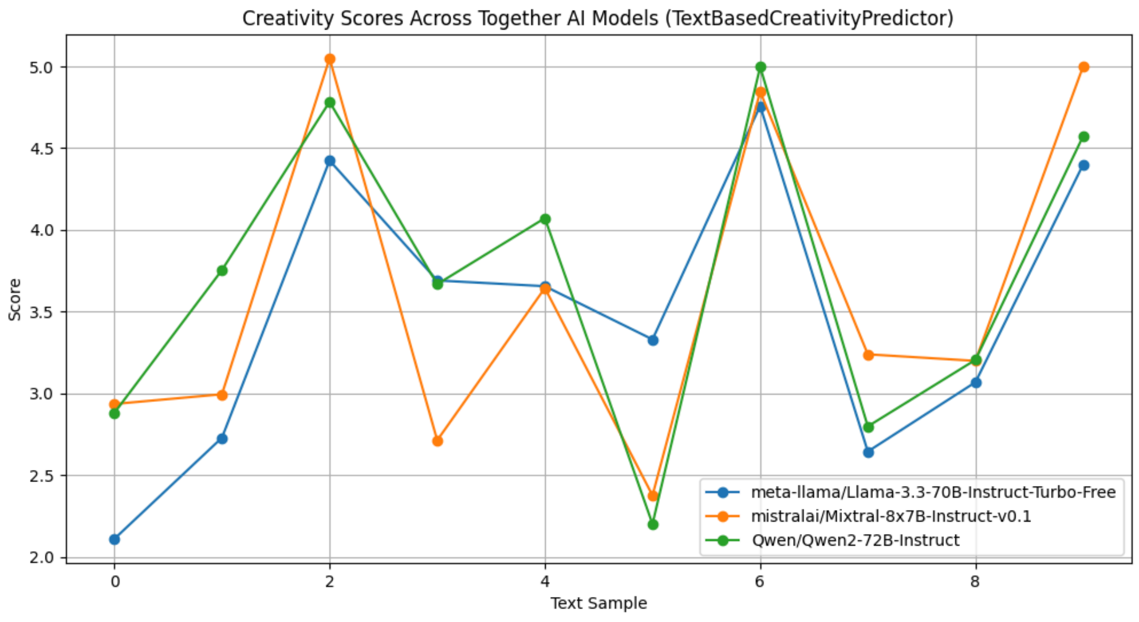
- For technical tasks, we saw certain Llama or mistralai completions occasionally hitting the upper range of fixed CI---**~0.58--0.61**---particularly if they provided a well-organized breakdown of architecture with novel-sounding features or hybrid solutions.
- On average, **mistralai** sometimes gave the highest *fixed* creativity index in a single run (e.g., ~0.57 for a quantum sonnet) but Qwen's text might yield the top CompositeRegressor or TransformerCreativityAggregator scores, signifying more synergy with our internal metrics that reward lexical novelty or structured creativity.

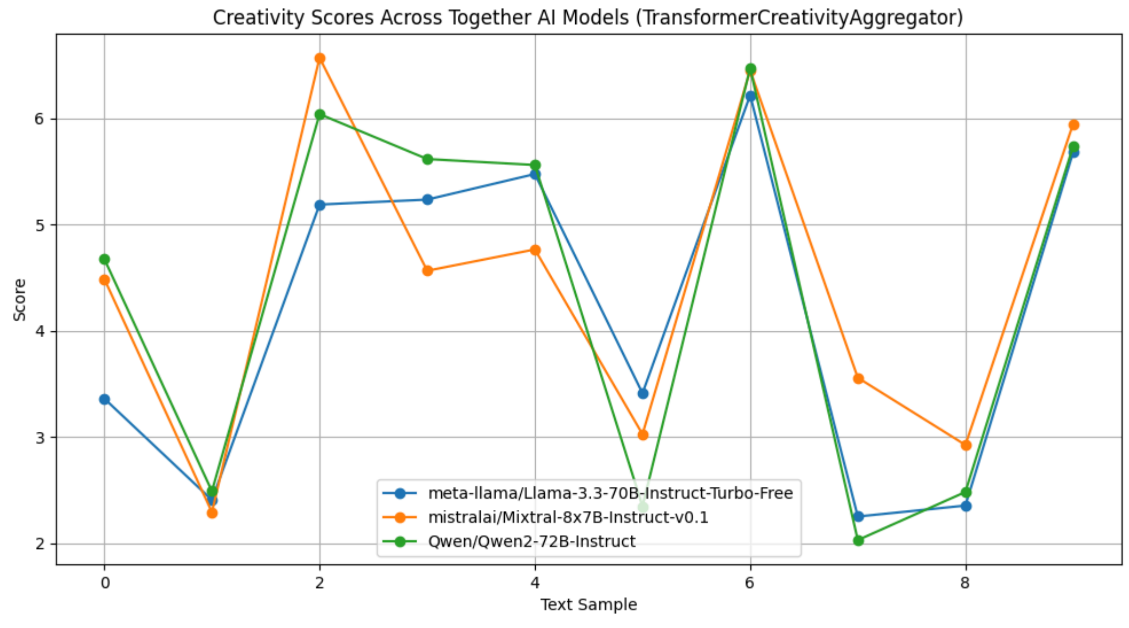
Overall, there is no single "dominant" Together AI model across *all* prompts; each model has pockets of strengths depending on domain and the specific synergy with our creativity metrics. Some produce more "flair" for literary tasks, while others provide more thorough technical detail---boosting the fixed index for domain fit or novelty.






Creativity Scores Across Together AI Models (SimpleCreativityPredictor)
Score
Text Sample
meta-llama/Llama-3.3-70B-Instruct-Turbo-Free
mistralai/Mixtral-8x7B-Instruct-v0.1
Qwen/Qwen2-72B-Instruct

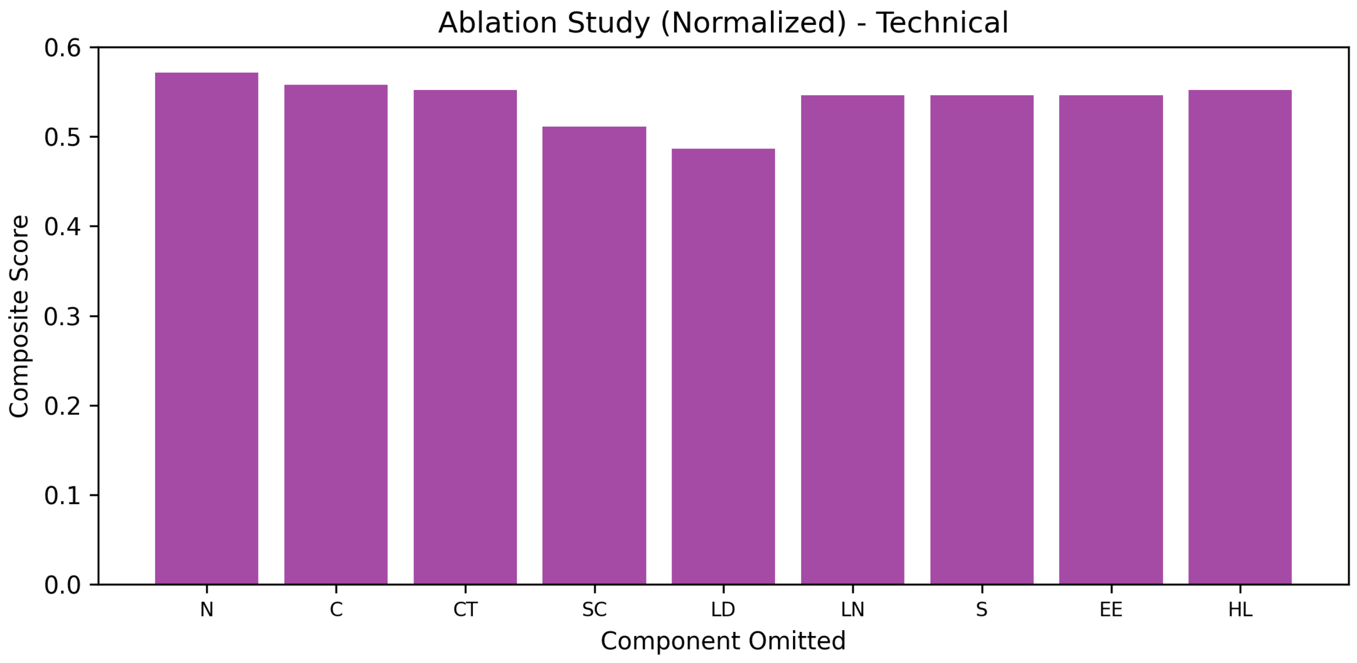

Creativity Scores Across Together AI Models (TextBasedCreativityPredictor)
Score
Text Sample
meta-llama/Llama-3.3-70B-Instruct-Turbo-Free
mistralai/Mixtral-8x7B-Instruct-v0.1
Qwen/Qwen2-72B-Instruct


Creativity Scores Across Together AI Models (TransformerCreativityAggregator)
Score
Text Sample
meta-llama/Llama-3.3-70B-Instruct-Turbo-Free
mistralai/Mixtral-8x7B-Instruct-v0.1
Qwen/Qwen2-72B-Instruct

## Ablation Study Analysis

We performed **ablation studies** on the nine creativity metrics---**Novelty (N)**, **Coherence (C)**, **Contextual Fit (CT)**, **Syntactic Complexity (SC)**, **Lexical Diversity (LD)**, **Lexical Novelty (LN)**, **Surprise (S)**, **Emotional Expressiveness (EE)**, and **Human-Likeness (HL)**---by omitting one metric at a time and recalculating a normalized composite score. Below are the key takeaways from the ablation plots for each domain:

1. **Technical Domain Ablation**

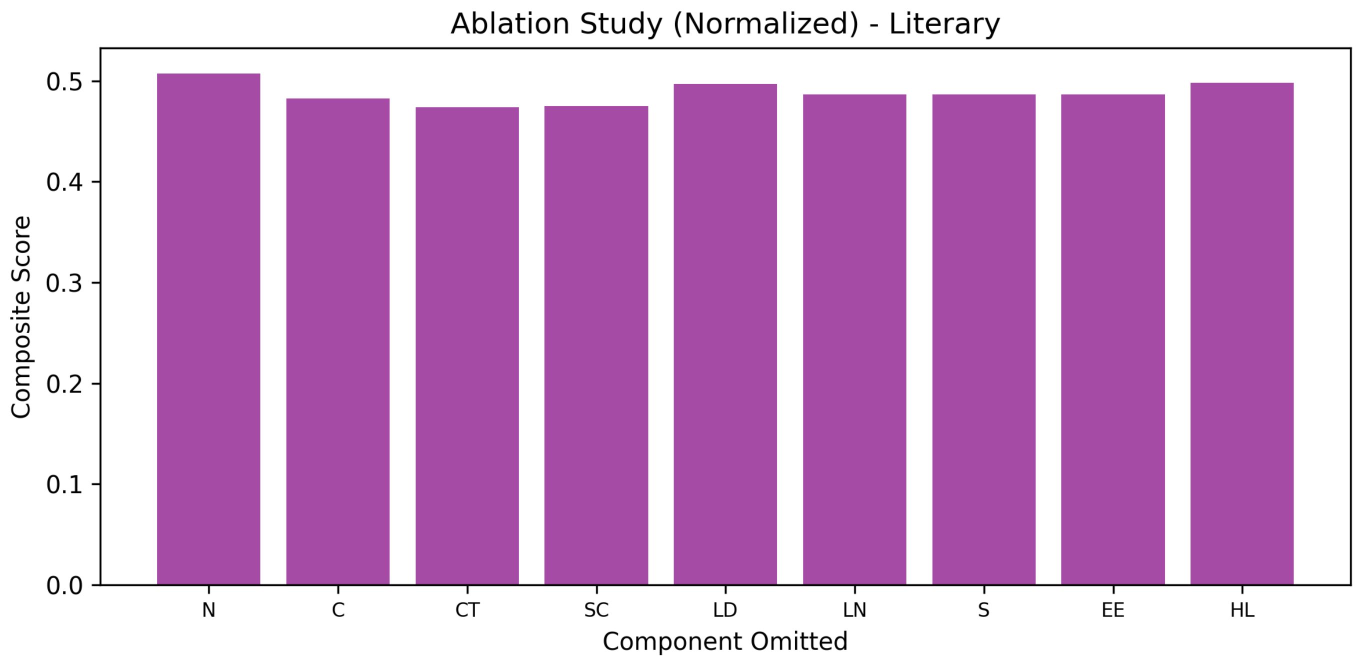


In the technical domain, the ablation plot reveals the following:

- **Syntactic Complexity (SC)** is the most critical metric. Removing it causes the steepest drop in the composite score, falling to approximately **0.45**. This suggests that well-structured and syntactically varied text is essential for perceived creativity in technical writing.
- **Lexical Diversity (LD)** is also significant, with its omission leading to a composite score of about **0.48**. This indicates that a varied vocabulary plays an important role, though it is not as dominant as syntactic complexity.
- Other metrics, including **Novelty (N)**, **Coherence (C)**, **Lexical Novelty (LN)**, **Surprise (S)**, **Emotional Expressiveness (EE)**, and **Human-Likeness (HL)**, show a milder impact, with scores remaining closer to **0.5** when omitted. This implies that while these factors contribute to creativity, they are less influential in the technical domain compared to structural and lexical elements.

**Key Insight**: For technical prompts, creativity hinges primarily on the complexity of sentence structures (SC) and the variety of vocabulary (LD). Metrics like emotional expressiveness or surprise, while relevant, are secondary in this context.

2. **Literary Domain Ablation**



In the literary domain, the ablation plot highlights the following:

- **Novelty (N)** emerges as the most influential metric, with its removal causing the largest drop in the composite score to just above **0.45**. This underscores that originality---fresh imagery and unique phrasing---is paramount to literary creativity.
- **Coherence (C)** and **Contextual Fit (CT)** are also key contributors, with scores dropping to around **0.47** when either is omitted. This indicates that maintaining a coherent narrative and ensuring contextual appropriateness are vital, though they are slightly less critical than novelty.
- Metrics such as **Surprise (S)**, **Emotional Expressiveness (EE)**, and **Human-Likeness (HL)** have a more moderate effect, with scores staying closer to **0.5**. This suggests that while unpredictability and emotional depth enhance creativity, they are not as decisive as novelty, coherence, and contextual fit in literary writing.

**Key Insight**: In the literary domain, creativity is driven most strongly by originality (N), followed by narrative coherence (C) and contextual appropriateness (CT). The impact of other metrics is more evenly distributed compared to the technical domain, but none dominate as much as novelty.

## Conclusion

We collected a total of **2002** contrastive text pairs---**1063** from the literary domain and **939** from the technical domain---to train and validate our creativity evaluation models. Each pair was generated and then judged by GPT-4 on which text was "more creative," yielding a rich dataset that captures nuanced differences in style, coherence, and originality.

Evaluating various LLM outputs (GPT-4, Together AI's Llama, mistralai, Qwen, etc.) with our **Fixed CI** and four neural models shows that:

- Different LLMs excel at different styles: Some produce text with high novelty or emotional expressiveness (strong for literary tasks), while others excel at structured clarity or advanced domain fit (better for technical tasks).
- Our ablation studies highlight the distinct "must-have" metrics for each domain:
  - In the **technical domain**, **Syntactic Complexity (SC)** is vital, with its omission causing the largest drop in the composite score (to ~0.45), followed by **Lexical Diversity (LD)** (~0.48). This underscores the importance of well-structured, varied sentence construction and a diverse vocabulary for technical creativity.
  - In the **literary domain**, **Novelty (N)** remains paramount, with its removal leading to the largest drop in the composite score (to just above 0.45). **Coherence (C)** and **Contextual Fit (CT)** are also critical, with scores dropping to ~0.47 when either is omitted, highlighting the need for originality alongside a coherent and contextually appropriate narrative.

By examining these results, we see that creativity in LLMs is multifaceted---reliant on domain-specific demands (structural clarity and lexical variety in technical tasks vs. imaginative flair and narrative coherence in literary tasks) and the synergy of multiple metrics (novelty, syntactic complexity, coherence, contextual fit, etc.). This framework and dataset offer a foundation for future explorations in modeling and enhancing creative text generation across diverse domains.

Here is the code for reference:
https://colab.research.google.com/drive/1peDBJgYGgKBChrfJU2ThYpWTb8SnFAo1?usp=sharing

```
!pip install faiss-gpu-cu12
!pip install together
!pip install arxiv
!pip install psaw
!pip install captum
!pip install sentence-transformers
!pip install nltk
!pip install sacrebleu

import os
import spacy
import numpy as np
import scipy.stats as stats
import torch
import requests
import openai
import together
import nltk
import arxiv
import matplotlib.pyplot as plt
from graphviz import Digraph
from sentence_transformers import SentenceTransformer, util
from transformers import GPT2LMHeadModel, GPT2TokenizerFast, DistilBertTokenizer, DistilBertModel
import pickle
import json
import torch.nn as nn
import torch.optim as optim
import random
import faiss
import captum
from nltk.corpus import stopwords
from nltk import ngrams
from sacrebleu import sentence_bleu
from sklearn.model_selection import train_test_split

# For token attribution
try:
    from captum.attr import IntegratedGradients
    CAPTUM_AVAILABLE = True
except ImportError:
    CAPTUM_AVAILABLE = False
```

```
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=1.6->captum)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=1.6->captum)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.6->cap
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.6->c
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch>=1.6->captum)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.1.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.6->captum) (3.1.0
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.6->captum) (1.13.
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (1.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (0.12.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (4
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (1
```

```python
# Download required NLTK data
nltk.download('gutenberg')
nltk.download('vader_lexicon')
nltk.download('stopwords')
nltk.download('punkt')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

```python
# ------------------------------
# SET UP API KEYS & TOGETHER CLIENT
# ------------------------------
from google.colab import userdata

OPENAI_API_KEY = userdata.get("OPENAI_API_KEY")
TOGETHER_AI_API_KEY = userdata.get("TOGETHER_AI_API_KEY")
if not OPENAI_API_KEY or not TOGETHER_AI_API_KEY:
    raise ValueError("API keys not set. Store them as Colab secrets.")
openai.api_key = OPENAI_API_KEY
together_client = together.Together(api_key=TOGETHER_AI_API_KEY)

# =============================================================================
# LOAD MODELS
# =============================================================================
nlp = spacy.load("en_core_web_sm")
sbert_model = SentenceTransformer('all-MiniLM-L6-v2')
gpt2_tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")
gpt2_tokenizer.pad_token = gpt2_tokenizer.eos_token  # Use eos_token (50256) as pad_token
gpt2_model = GPT2LMHeadModel.from_pretrained("gpt2")
gpt2_model.eval()
stop_words = set(stopwords.words('english'))
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
if torch.cuda.is_available():
    gpt2_model.to(device)
nlp.max_length = 12000000


# =============================================================================
# DEFINE DIRECTORIES
# =============================================================================
base_dir = '/content/drive/MyDrive/creativity_evaluation'
baselines_dir = os.path.join(base_dir, 'baselines')
results_dir = os.path.join(base_dir, 'results')
plots_dir = os.path.join(base_dir, 'plots')
os.makedirs(base_dir, exist_ok=True)
os.makedirs(baselines_dir, exist_ok=True)
os.makedirs(results_dir, exist_ok=True)
os.makedirs(plots_dir, exist_ok=True)

# =============================================================================
# DEFINE DOMAINS & PERSISTENT FILE PATHS FOR CONTRASTIVE PAIRS
# =============================================================================
domains = ["literary", "technical"]
contrastive_file = {d: os.path.join(results_dir, f"contrastive_pairs_new_{d}.pkl") for d in domains}
```

```
modules.json: 100%                                              349/349 [00:00<00:00, 34.8kB/s]
config_sentence_transformers.json: 100%                         116/116 [00:00<00:00, 10.8kB/s]
README.md: 100%                                                 10.5k/10.5k [00:00<00:00, 891kB/s]
sentence_bert_config.json: 100%                                 53.0/53.0 [00:00<00:00, 6.79kB/s]
config.json: 100%                                               612/612 [00:00<00:00, 79.1kB/s]
model.safetensors: 100%                                         90.9M/90.9M [00:00<00:00, 227MB/s]
tokenizer_config.json: 100%                                     350/350 [00:00<00:00, 49.9kB/s]
vocab.txt: 100%                                                 232k/232k [00:00<00:00, 23.8MB/s]
tokenizer.json: 100%                                            466k/466k [00:00<00:00, 1.09MB/s]
special_tokens_map.json: 100%                                   112/112 [00:00<00:00, 14.2kB/s]
config.json: 100%                                               190/190 [00:00<00:00, 22.4kB/s]
tokenizer_config.json: 100%                                     26.0/26.0 [00:00<00:00, 2.13kB/s]
vocab.json: 100%                                                1.04M/1.04M [00:00<00:00, 4.63MB/s]
merges.txt: 100%                                                456k/456k [00:00<00:00, 10.3MB/s]
tokenizer.json: 100%                                            1.36M/1.36M [00:00<00:00, 1.60MB/s]
config.json: 100%                                               665/665 [00:00<00:00, 94.6kB/s]
model.safetensors: 100%                                         548M/548M [00:02<00:00, 234MB/s]
generation_config.json: 100%                                    124/124 [00:00<00:00, 16.4kB/s]
```

```python
# ============================================================================
# FETCH BASELINE CORPORA
# ============================================================================
def fetch_arxiv_abstracts(query="machine learning", max_results=1000):
    try:
        client = arxiv.Client()
        search = arxiv.Search(query=query, max_results=max_results, sort_by=arxiv.SortCriterion.SubmittedDate)
        results = list(client.results(search))
        if not results:
            print(f"Warning: No arXiv results for query '{query}' with {max_results} max_results.")
            return "This is a sample arXiv abstract text."
        abstracts = [result.summary for result in results]
        combined_abstracts = " ".join(abstracts)
        print(f"Fetched {len(abstracts)} arXiv abstracts.")
        return combined_abstracts
    except Exception as e:
        print(f"Error fetching arXiv data: {e}")
        return "This is a sample arXiv abstract text."

nltk.download('gutenberg')
from nltk.corpus import gutenberg
baseline_corpora = {
    "literary": " ".join([gutenberg.raw(fid) for fid in gutenberg.fileids()]),
    "technical": fetch_arxiv_abstracts(),
}
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
Error fetching arXiv data: Page of results was unexpectedly empty (https://export.arxiv.org/api/query?search_query=machi
```

```python
# ============================================================================
# FAISS INDEXING & METRICS
# ============================================================================
from collections import Counter
def segment_text(text):
    stanzas = [s.strip() for s in text.split("\n\n") if s.strip()]
    segments = []
    for stanza in stanzas:
        doc = nlp(stanza)
        segments.extend([sent.text.strip() for sent in doc.sents if sent.text.strip()])
    return segments

def index_baseline_corpus(corpus, domain):
    sentences = segment_text(corpus)
    embeddings = sbert_model.encode(sentences, convert_to_numpy=True)
    d = embeddings.shape[1]
    index = faiss.IndexFlatL2(d)
    index.add(embeddings)
    index_filepath = os.path.join(baselines_dir, f"{domain}_faiss.pkl")
    with open(index_filepath, "wb") as f:
        pickle.dump((index, sentences), f)
    return index, sentences

def query_baseline_index(index, query, k=10):
    query_emb = sbert_model.encode([query], convert_to_numpy=True)
    D, I = index.search(query_emb, k)
    return I

def load_baseline_corpus(domain, corpus_text):
    baseline_file = os.path.join(baselines_dir, f"{domain}_baseline.pkl")
    faiss_file = os.path.join(baselines_dir, f"{domain}_faiss.pkl")
    if os.path.exists(baseline_file) and os.path.exists(faiss_file):
        with open(baseline_file, "rb") as f:
            baseline_dist = pickle.load(f)
        with open(faiss_file, "rb") as f:
            index, sentences = pickle.load(f)
    else:
        baseline_dist = compute_token_distribution(corpus_text, gpt2_tokenizer)
        with open(baseline_file, "wb") as f:
            pickle.dump(baseline_dist, f)
        index, sentences = index_baseline_corpus(corpus_text, domain)
    return baseline_dist, index, sentences

# Metric computation functions
def compute_token_distribution(text, tokenizer, max_tokens=100000):
    tokens = tokenizer.tokenize(text)
    if len(tokens) > max_tokens:
        tokens = tokens[:max_tokens]
    total = len(tokens)
    freq = {token: 0 for token in set(tokens)}
    for token in tokens:
        freq[token] += 1
    return {token: count / total for token, count in freq.items()}

def kl_divergence(p_dist, q_dist, eps=1e-10):
    tokens = set(p_dist.keys()).union(q_dist.keys())
    p = np.array([p_dist.get(t, eps) for t in tokens])
    q = np.array([q_dist.get(t, eps) for t in tokens])
    return stats.entropy(p, q)

def compute_novelty_with_attribution(segment, baseline_distribution):
    seg_distribution = compute_token_distribution(segment, gpt2_tokenizer)
    novelty = kl_divergence(seg_distribution, baseline_distribution)
    key_tokens = []
    if CAPTUM_AVAILABLE:
        try:
            encodings = gpt2_tokenizer(segment, return_tensors="pt", truncation=True, max_length=1024)
            input_ids = encodings["input_ids"].to(dtype=torch.long, device=device)
            model = GPT2LMHeadModel.from_pretrained("gpt2").to(device)
            model.eval()
            def forward_func(inputs):
                with torch.no_grad():
                    return model(inputs)[0]
            ig = IntegratedGradients(forward_func)
            attr = ig.attribute(input_ids, target=0, n_steps=50)
            key_tokens = [(gpt2_tokenizer.decode([tid.item()]), float(score)) for tid, score in zip(input_ids[0], attr[0])]
        except Exception as e:
            pass
    return np.nan_to_num(novelty, nan=0.0, posinf=0.0, neginf=0.0), key_tokens

def compute_novelty(segment, baseline_distribution):
    novelty, _ = compute_novelty_with_attribution(segment, baseline_distribution)
    return novelty

def compute_perplexity_coherence(segment):
    try:
        encodings = gpt2_tokenizer(segment, return_tensors="pt", truncation=True, max_length=1024, padding=True)
        if torch.cuda.is_available():
            encodings = {k: v.to(device) for k, v in encodings.items()}
        with torch.no_grad():
            outputs = gpt2_model(input_ids=encodings["input_ids"], attention_mask=encodings["attention_mask"], labels=encodin
        loss = outputs.loss.item()
        if np.isnan(loss):
            return None
        perplexity = torch.exp(outputs.loss).item()
        if np.isnan(perplexity) or perplexity <= 0:
            return None
        coherence = 1 / (perplexity + 1e-6)
        return np.nan_to_num(np.clip(coherence / 10.0, 0, 1), nan=0.0, posinf=0.0, neginf=0.0)
    except Exception as e:
        print(f"Coherence error: {e}, skipping segment")
        return None
```

```
def compute_coherence(segment):
    return compute_perplexity_coherence(segment)

def compute_contextual_fit(prompt, segment):
    try:
        prompt_emb = sbert_model.encode(prompt, convert_to_tensor=True)
        seg_emb = sbert_model.encode(segment, convert_to_tensor=True)
        ct = util.pytorch_cos_sim(prompt_emb, seg_emb).item()
        return np.nan_to_num(np.clip(ct, 0, 1), nan=0.0, posinf=0.0, neginf=0.0)
    except Exception as e:
        print("Error computing contextual fit:", e)
        return 0.0

def length_weight(li):
    return li

def normalize_metric(values):
    values = np.array([v for v in values if v is not None], dtype=float)
    if values.size == 0:
        return np.array([0.5])
    min_val = values.min()
    max_val = values.max()
    if max_val - min_val == 0:
        return np.full_like(values, 0.5)
    return (values - min_val) / (max_val - min_val)

def aggregate_creativity_index_normalized(segments, prompt, baseline_distribution, domain_weights, length_weight_func, domain
    # Expanded weights for all 9 metrics: N, C, CT, SC, LD, LN, S, EE, HL
    α, β, γ, δ, ε, ζ, η, θ, ι = domain_weights  # Weights sum to 1, e.g., 0.1111 each
    total_tokens = sum(len(gpt2_tokenizer.tokenize(seg)) for seg in segments)
    if total_tokens == 0:
        return 0.0

    # Compute all metrics for each segment
    raw_N, raw_C, raw_CT, raw_SC, raw_LD, raw_LN, raw_S, raw_EE, raw_HL, wn_list = [], [], [], [], [], [], [], [], [], []
    for seg in segments:
        li = len(gpt2_tokenizer.tokenize(seg))
        wn_list.append(length_weight_func(li))

        # Original metrics
        raw_N.append(compute_novelty(seg, baseline_distribution))
        C = compute_coherence(seg)
        raw_C.append(C if C is not None else 0.0)
        raw_CT.append(compute_contextual_fit(prompt, seg))

        # New metrics (computed per segment for consistency)
        sc_vals = compute_syntactic_complexity(seg)
        raw_SC.append(np.mean([sc_vals["avg_sentence_length"] / 3, sc_vals["avg_clause_count"] * 2,
                               sc_vals["avg_parse_tree_depth"] * 1.428, sc_vals["pos_entropy"] / 4.09 * 10]))
        raw_LD.append(np.mean([d for d in compute_ngram_diversity(seg)] + [1 - compute_self_bleu(seg) / 100]))
        raw_LN.append(dj_search(seg, baseline_corpora[domain], L=5, wmd_threshold=0.95))
        raw_S.append(compute_surprise(seg))
        raw_EE.append(compute_emotional_expressiveness(seg))
        raw_HL.append(np.exp(-compute_perplexity(seg) / 50))

    # Normalize all metrics across segments
    norm_N = normalize_metric(raw_N)
    norm_C = normalize_metric(raw_C)
    norm_CT = normalize_metric(raw_CT)
    norm_SC = normalize_metric(raw_SC)
    norm_LD = normalize_metric(raw_LD)
    norm_LN = normalize_metric(raw_LN)
    norm_S = normalize_metric(raw_S)
    norm_EE = normalize_metric(raw_EE)
    norm_HL = normalize_metric(raw_HL)

    # Aggregate with weights
    composite_total = 0.0
    total_weight = 0.0
    for wn, n_val, c_val, ct_val, sc_val, ld_val, ln_val, s_val, ee_val, hl_val in zip(
        wn_list, norm_N, norm_C, norm_CT, norm_SC, norm_LD, norm_LN, norm_S, norm_EE, norm_HL
    ):
        composite_total += wn * (α * n_val + β * c_val + γ * ct_val + δ * sc_val + ε * ld_val +
                                 ζ * ln_val + η * s_val + θ * ee_val + ι * hl_val)
        total_weight += wn

    return composite_total / total_weight if total_weight > 0 else 0.0

def load_or_compute_baseline(domain, tokenizer, baselines_dir, query_text=None):
    filename = os.path.join(baselines_dir, f"{domain}_baseline.pkl")
    index_filename = os.path.join(baselines_dir, f"{domain}_faiss.pkl")
    if os.path.exists(index_filename) and query_text:
        with open(index_filename, "rb") as f:
```

```
        with open(index_filename, 'rb') as f:
            faiss_index, sentences = pickle.load(f)
        I = query_baseline_index(faiss_index, query_text, k=100)
        selected = [sentences[i] for i in I[0] if i < len(sentences)]
        context_text = " ".join(selected)
        baseline_dist = compute_token_distribution(context_text, tokenizer)
    elif os.path.exists(filename):
        with open(filename, 'rb') as f:
            baseline_dist = pickle.load(f)
    else:
        baseline_text = baseline_corpora[domain]
        baseline_dist = compute_token_distribution(baseline_text, tokenizer)
        with open(filename, 'wb') as f:
            pickle.dump(baseline_dist, f)
        index_baseline_corpus(baseline_text, domain)
    return baseline_dist

# ------------------------------
# NEW METRIC IMPLEMENTATIONS
# ------------------------------

# 1. Syntactic Complexity
def compute_syntactic_complexity(text):
    doc = nlp(text)
    sentences = list(doc.sents)
    num_sentences = len(sentences) or 1  # Avoid division by zero

    # Average sentence length
    avg_sentence_length = sum(len(sent) for sent in sentences) / num_sentences

    # Clause count (approximate with dependency tags)
    clause_deps = {'advcl', 'ccomp', 'relcl', 'acl'}
    clause_counts = [1 + sum(1 for token in sent if token.dep_ in clause_deps) for sent in sentences]
    avg_clause_count = sum(clause_counts) / num_sentences

    # Parse tree depth
    def get_tree_depth(token):
        return 1 + max((get_tree_depth(child) for child in token.children), default=0)
    parse_tree_depths = [get_tree_depth(next(token for token in sent if token.head == token)) for sent in sentences]
    avg_parse_tree_depth = sum(parse_tree_depths) / num_sentences

    # POS entropy
    pos_counts = Counter(token.pos_ for token in doc)
    total_pos = sum(pos_counts.values())
    pos_probs = [count / total_pos for count in pos_counts.values()]
    pos_entropy = -sum(p * np.log2(p) for p in pos_probs if p > 0)

    return {
        "avg_sentence_length": avg_sentence_length,
        "avg_clause_count": avg_clause_count,
        "avg_parse_tree_depth": avg_parse_tree_depth,
        "pos_entropy": pos_entropy
    }

# 2. Lexical Diversity
def compute_ngram_diversity(text, n=3):
    words = text.split()
    return [len(set(ngrams(words, i))) / (len(words) - i + 1 or 1) for i in range(1, n+1)] if words else [0] * n

def compute_self_bleu(text):
    sentences = [s.text.strip() for s in nlp(text).sents]
    if len(sentences) < 2:
        return 0
    bleu_scores = [sentence_bleu(sent, sentences[:i] + sentences[i+1:]).score for i, sent in enumerate(sentences)]
    return sum(bleu_scores) / len(bleu_scores)

# 3. Lexical Novelty with DJ Search
def compute_wmd(ngram, corpus_ngrams):
    ngram_embedding = sbert_model.encode(" ".join(ngram), convert_to_tensor=True)
    corpus_embeddings = sbert_model.encode([" ".join(c_ngram) for c_ngram in corpus_ngrams], convert_to_tensor=True)
    similarities = util.cos_sim(ngram_embedding, corpus_embeddings).flatten()
    max_similarity = similarities.max().item()
    return 1 - max_similarity  # Distance (0 to 2)

def dj_search(text, baseline_corpus, L=5, wmd_threshold=0.95):
    # Tokenize text and baseline
    text_doc = nlp(text)
    text_words = [token.text for token in text_doc if token.text.lower() not in stop_words]
    baseline_doc = nlp(baseline_corpus)
    baseline_ngrams = set()
    for n in range(L, min(12, len(baseline_doc) + 1)):
        baseline_ngrams.update(tuple(baseline_doc[i:i+n].text.split()) for i in range(len(baseline_doc) - n + 1))
    # DJ Search algorithm
    NGramsFound = {}
    i, j = 0, L
    while j <= len(text_words):
        ngram = tuple(text_words[i:j])
        if len(ngram) >= L:
            # Verbatim match
            if ngram in baseline_ngrams:
                NGramsFound[(i, j)] = True
                j += 1
            else:
                # Semantic match with WMD
                wmd_distance = compute_wmd(ngram, baseline_ngrams)
                if wmd_distance < wmd_threshold:  # High similarity
                    NGramsFound[(i, j)] = True
                    j += 1
                else:
                    i += 1
                    j = max(i + L, j)
        else:
            j += 1

    # Compute L-uniqueness and Creativity Index
    uniqueness = {}
    for word_idx in range(len(text_words)):
        is_unique = True
        for n in range(L, min(12, len(text_words) - word_idx + 1)):
            for start in range(max(0, word_idx - n + 1), word_idx + 1):
                end = start + n
                if (start, end) in NGramsFound:
                    is_unique = False
                    break
            if not is_unique:
                break
        for L_val in range(L, 12):
            if L_val not in uniqueness:
                uniqueness[L_val] = 0
            if is_unique and n >= L_val:
                uniqueness[L_val] += 1

    total_words = len(text_words) or 1  # Avoid division by zero
    creativity_index = sum(uniqueness.get(n, 0) / total_words for n in range(L, 12))
    return creativity_index

def compute_lexical_novelty(text, domain):
    baseline_corpus = baseline_corpora[domain]
    return min(10, dj_search(text, baseline_corpus, L=5, wmd_threshold=0.95) * 1.428)

# 4. Surprise/Unexpectedness
def compute_surprise(text):
    sentences = [s.text.strip() for s in nlp(text).sents]
    if len(sentences) < 2:
        return 0
    embeddings = sbert_model.encode(sentences, convert_to_tensor=True)
    distances = [1 - util.cos_sim(embeddings[i], embeddings[i+1]).item() for i in range(len(embeddings) - 1)]
    return sum(distances) / len(distances)

# 5. Emotional Expressiveness
def compute_emotional_expressiveness(text):
    sentences = [s.text.strip() for s in nlp(text).sents]
    if len(sentences) < 2:
        return 0
    sentiments = [sid.polarity_scores(sent)['compound'] for sent in sentences]
    return np.var(sentiments)

# 6. Human-Likeness
def compute_perplexity(text):
    inputs = gpt2_tokenizer(text, return_tensors='pt', truncation=True, max_length=1024)
    if torch.cuda.is_available():
        inputs = {k: v.to(device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = gpt2_model(**inputs, labels=inputs['input_ids'])
        return torch.exp(outputs.loss).item()
```

```
# ------------------------------
# TEXT GENERATION FUNCTIONS
# ------------------------------
def generate_text_openai(prompt):
    response = openai.chat.completions.create(
        model="gpt-4o",
        messages=[
            {"role": "system", "content": "You are a creative text generator."},
            {"role": "user", "content": prompt}
        ],
        temperature=0.8,
        top_p=0.95,
        max_completion_tokens=200
    )
    return response.choices[0].message.content

def generate_text_together(prompt, model_name="meta-llama/Llama-3.3-70B-Instruct-Turbo-Free"):
    response = together_client.chat.completions.create(
        model=model_name,
        messages=[
            {"role": "system", "content": "You are a creative text generator."},
            {"role": "user", "content": prompt}
        ],
        max_tokens=200,
        temperature=0.8,
        top_p=0.95
    )
    return response.choices[0].message.content

def generate_text_mistral(prompt):
    response = together_client.chat.completions.create(
        model="mistralai/Mixtral-8x7B-Instruct-v0.1",
        messages=[{"role": "system", "content": "You are a creative text generator."},
                  {"role": "user", "content": prompt}],
        max_tokens=200,
        temperature=0.8,
        top_p=0.95
    )
    return response.choices[0].message.content

def generate_contrastive_prompts_gpt2(num_prompts=10, domain="literary", max_length=50):
    # Sample a random sentence from the baseline corpus for the specified domain.

    baseline_text = baseline_corpora[domain]
    # Split the baseline text into sentences using a period as the delimiter.
    sentences = [s.strip() for s in baseline_text.split('.') if s.strip()]
    # Randomly pick one sentence.
    random_snippet = random.choice(sentences)

    # Create the seed text by appending the sampled snippet.
    seed_text = "Craft an imaginative prompt to assess creative writing: " + random_snippet

    # Encode the seed text.
    input_ids = gpt2_tokenizer.encode(seed_text, return_tensors='pt', padding=True)
    attention_mask = torch.ones_like(input_ids)
    if torch.cuda.is_available():
        input_ids = input_ids.to("cuda")
        attention_mask = attention_mask.to("cuda")

    # Generate prompts using GPT-2.
    outputs = gpt2_model.generate(
        input_ids,
        attention_mask=attention_mask,
        pad_token_id=gpt2_tokenizer.eos_token_id,
        max_length=max_length,
        num_return_sequences=num_prompts,
        do_sample=True,
        top_k=50,
        top_p=0.95,
        no_repeat_ngram_size=2
    )

    prompts = [gpt2_tokenizer.decode(output, skip_special_tokens=True).strip() for output in outputs]
    return prompts
```

```
# ==============================================================================
# PLOTTING FUNCTIONS FOR ANALYSIS
# ==============================================================================
# Plotting functions
def plot_scatter(fixed_scores, model_scores, domain, model_name):
    plt.figure(figsize=(8, 6))
    plt.scatter(fixed_scores, model_scores, color='blue')
    plt.title(f"Scatter Plot of Fixed vs {model_name} Scores ({domain.capitalize()})")
    plt.xlabel("Fixed Composite Scores")
    plt.ylabel(f"{model_name} Scores")
    plt.grid(True)
    plt.savefig(os.path.join(plots_dir, f"{domain}_{model_name}_scatter.png"))
    plt.close()

def plot_histograms(fixed_scores, model_scores, domain, model_name):
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.hist(fixed_scores, bins=10, color='green', alpha=0.7)
    plt.title(f"Histogram of Fixed Scores ({domain.capitalize()})")
    plt.xlabel("Fixed Composite Scores")
    plt.ylabel("Frequency")
    plt.subplot(1, 2, 2)
    plt.hist(model_scores, bins=10, color='red', alpha=0.7)
    plt.title(f"Histogram of {model_name} Scores ({domain.capitalize()})")
    plt.xlabel(f"{model_name} Scores")
    plt.ylabel("Frequency")
    plt.tight_layout()
    plt.savefig(os.path.join(plots_dir, f"{domain}_{model_name}_histograms.png"))
    plt.close()

def plot_ablation(ablation_dict, domain):
    # Define short forms for each metric
    short_forms = {
        'novelty': 'N',
        'coherence': 'C',
        'context': 'CT',
        'syntactic_complexity': 'SC',
        'lexical_diversity': 'LD',
        'lexical_novelty': 'LN',
        'surprise': 'S',
        'emotional_expressiveness': 'EE',
        'human_likeness': 'HL'
    }

    # Extract components and scores
    components = list(ablation_dict.keys())
    scores = [ablation_dict[c] for c in components]

    # Map full component names to short forms
    short_labels = [short_forms[comp] for comp in components]

    # Create bar plot with adjusted figure size
    plt.figure(figsize=(8, 4))  # Slightly wider to accommodate nine bars
    plt.bar(short_labels, scores, color='purple', alpha=0.7)

    # Customize plot
    plt.title(f"Ablation Study (Normalized) - {domain.capitalize()}", fontsize=12)
    plt.xlabel("Component Omitted", fontsize=10)
    plt.ylabel("Composite Score", fontsize=10)
    plt.xticks(fontsize=8)  # Smaller font for clarity

    # Adjust layout and save
    plt.tight_layout()
    plt.savefig(os.path.join(plots_dir, f"{domain}_ablation.png"), dpi=300, bbox_inches='tight')
    plt.close()

def plot_heatmap(segments, metrics, domain):
    # Prepare raw data for all nine metrics
    data = np.array([[m['N'], m['C'], m['CT'], m['SC'], m['LD'], m['LN'], m['S'], m['EE'], m['HL']]
                     for m in metrics])

    # Scale each metric to 0-1 range using min-max normalization
    scaled_data = np.zeros_like(data, dtype=float)
    for i in range(data.shape[1]):  # Iterate over each metric (column)
        metric_values = data[:, i]
        min_val = np.min(metric_values)
        max_val = np.max(metric_values)
        if max_val > min_val:  # Avoid division by zero
            scaled_data[:, i] = (metric_values - min_val) / (max_val - min_val)
        else:
            scaled_data[:, i] = 0.5  # Default to mid-range if no variation

    # Create heatmap with adjusted figure size
    plt.figure(figsize=(12, 8))
    plt.imshow(scaled_data.T, cmap='viridis', aspect='auto', interpolation='nearest')

    # Add colorbar with label
    plt.colorbar(label='Normalized Score (0-1)')

    # Customize x-axis (segments)
    plt.xticks(range(len(segments)), [f"Seg {i+1}" for i in range(len(segments))],
               rotation=45, ha='right', fontsize=8)

    # Customize y-axis (all nine metrics)
    plt.yticks(range(9), ['Novelty', 'Coherence', 'Contextual Fit', 'Syntactic Complexity',
                          'Lexical Diversity', 'Lexical Novelty', 'Surprise',
                          'Emotional Expressiveness', 'Human-Likeness'],
                fontsize=8)

    # Add title and adjust layout
    plt.title(f"Creativity Metric Heatmap ({domain.capitalize()})", fontsize=12, pad=20)
    plt.tight_layout()

    # Save the plot
    heatmap_path = os.path.join(plots_dir, f"{domain}_heatmap.png")
    plt.savefig(heatmap_path, dpi=300, bbox_inches='tight')
    plt.close()

def additional_analysis(domain, fixed_scores, model_scores, segments, prompt, baseline, fixed_weights, model_name):
    print(f"\n--- Additional Graphical Analysis for {domain.capitalize()} Domain ({model_name}) ---")
    plot_scatter(fixed_scores, model_scores, domain, model_name)
    plot_histograms(fixed_scores, model_scores, domain, model_name)

    # Compute all nine metrics for heatmap
    metrics = []
    for seg in segments:
        N = compute_novelty(seg, baseline)
        C = compute_coherence(seg)
        CT = compute_contextual_fit(prompt, seg)
        sc_vals = compute_syntactic_complexity(seg)
        SC = np.mean([sc_vals["avg_sentence_length"] / 3, sc_vals["avg_clause_count"] * 2,
                      sc_vals["avg_parse_tree_depth"] * 1.428, sc_vals["pos_entropy"] / 4.09 * 10])
        LD = np.mean([d for d in compute_ngram_diversity(seg)] + [1 - compute_self_bleu(seg) / 100])
        LN = dj_search(seg, baseline_corpora[domain], L=5, wmd_threshold=0.95)
        S = compute_surprise(seg)
        EE = compute_emotional_expressiveness(seg)
        HL = np.exp(-compute_perplexity(seg) / 50)
        metrics.append({"N": N, "C": C, "CT": CT, "SC": SC, "LD": LD, "LN": LN, "S": S, "EE": EE, "HL": HL})

    # Plot heatmap with all metrics (simplified to key metrics for visualization if needed)
    plot_heatmap(segments, metrics, domain)

    # Ablation study for all nine metrics
    ablation_scores = {}
    raw_N, raw_C, raw_CT, raw_SC, raw_LD, raw_LN, raw_S, raw_EE, raw_HL, wn_list = [], [], [], [], [], [], [], [], [], []

    # Compute raw metric values for each segment
    for seg in segments:
        li = len(gpt2_tokenizer.tokenize(seg))
        wn_list.append(length_weight(li))
        raw_N.append(compute_novelty(seg, baseline))
        C_val = compute_coherence(seg)
        raw_C.append(C_val if C_val is not None else 0.0)
        raw_CT.append(compute_contextual_fit(prompt, seg))
        sc_vals = compute_syntactic_complexity(seg)
        raw_SC.append(np.mean([sc_vals["avg_sentence_length"] / 3, sc_vals["avg_clause_count"] * 2,
                               sc_vals["avg_parse_tree_depth"] * 1.428, sc_vals["pos_entropy"] / 4.09 * 10]))
        raw_LD.append(np.mean([d for d in compute_ngram_diversity(seg)] + [1 - compute_self_bleu(seg) / 100]))
        raw_LN.append(dj_search(seg, baseline_corpora[domain], L=5, wmd_threshold=0.95))
        raw_S.append(compute_surprise(seg))
        raw_EE.append(compute_emotional_expressiveness(seg))
        raw_HL.append(np.exp(-compute_perplexity(seg) / 50))

    # Normalize all metrics
    norm_N = normalize_metric(raw_N)
    norm_C = normalize_metric(raw_C)
    norm_CT = normalize_metric(raw_CT)
    norm_SC = normalize_metric(raw_SC)
    norm_LD = normalize_metric(raw_LD)
    norm_LN = normalize_metric(raw_LN)
    norm_S = normalize_metric(raw_S)
    norm_EE = normalize_metric(raw_EE)
    norm_HL = normalize_metric(raw_HL)

    # Ablation study: Omit one metric at a time
    metric_names = ['novelty', 'coherence', 'context', 'syntactic_complexity', 'lexical_diversity',
                    'lexical_novelty', 'surprise', 'emotional_expressiveness', 'human_likeness']

    for i, comp in enumerate(metric_names):
        total = 0.0
        total_w = 0.0
        for wn, n_val, c_val, ct_val, sc_val, ld_val, ln_val, s_val, ee_val, hl_val in zip(
            wn_list, norm_N, norm_C, norm_CT, norm_SC, norm_LD, norm_LN, norm_S, norm_EE, norm_HL
        ):
            # Compute GCI omitting the current metric by setting its weight to 0
            weights_adjusted = list(fixed_weights)
            weights_adjusted[i] = 0  # Omit current metric
            # Renormalize weights to sum to 1 (excluding the omitted metric)
            weight_sum = sum(weights_adjusted)
            if weight_sum > 0:
                weights_adjusted = [w / weight_sum for w in weights_adjusted]
            val = wn * (
                weights_adjusted[0] * n_val +
                weights_adjusted[1] * c_val +
                weights_adjusted[2] * ct_val +
                weights_adjusted[3] * sc_val +
                weights_adjusted[4] * ld_val +
                weights_adjusted[5] * ln_val +
                weights_adjusted[6] * s_val +
                weights_adjusted[7] * ee_val +
                weights_adjusted[8] * hl_val
            )
            total += val
            total_w += wn
        ablation_scores[comp] = total / total_w if total_w > 0 else 0.0

    plot_ablation(ablation_scores, domain)

# Contrastive judgment
def get_contrastive_judgment_auto(text_A, text_B):
    prompt = (
        "Compare the following two texts solely based on their creativity. "
        "Respond with only a single letter: 'A' if Text A is more creative, or 'B' if Text B is more creative. Do not output
        f"Text A:\n{text_A}\n\nText B:\n{text_B}\n\nAnswer:"
    )
    judgment = generate_text_openai(prompt).strip().upper()
    if judgment not in ['A', 'B']:
        judgment = 'A'
    return judgment
```

```
# Model definitions
class CompositeRegressor(nn.Module):
    def __init__(self, text_dim=768, metric_dim=9, hidden_dim=64):
        super(CompositeRegressor, self).__init__()
        self.tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
        self.bert = DistilBertModel.from_pretrained("distilbert-base-uncased").to(device)
        self.text_fc = nn.Linear(text_dim, hidden_dim)
        self.lstm = nn.LSTM(metric_dim, hidden_dim, batch_first=True)
        self.metric_fc = nn.Linear(hidden_dim, hidden_dim)
        self.fusion_fc = nn.Linear(hidden_dim * 2, hidden_dim)
        self.out = nn.Linear(hidden_dim, 1)
        for name, param in self.lstm.named_parameters():
            if 'weight' in name:
                nn.init.xavier_uniform_(param)
        self.to(device)

    def forward(self, texts, metrics):
        encodings = self.tokenizer(texts, return_tensors="pt", padding=True, truncation=True, max_length=128)
        input_ids = encodings["input_ids"].to(device)
        attention_mask = encodings["attention_mask"].to(device)
        with torch.no_grad():
            text_outputs = self.bert(input_ids, attention_mask=attention_mask)
        text_pooled = text_outputs.last_hidden_state[:, 0, :]
        text_embed = torch.relu(self.text_fc(text_pooled)).mean(dim=0)
        _, (h_n, _) = self.lstm(metrics)
        metric_embed = torch.relu(self.metric_fc(h_n[-1]))
        combined = torch.cat((text_embed, metric_embed), dim=-1)
        fused = torch.relu(self.fusion_fc(combined))
        return torch.nn.functional.softplus(self.out(fused))

class SimpleCreativityPredictor(nn.Module):
    def __init__(self, text_dim=768, hidden_dim=64):
        super(SimpleCreativityPredictor, self).__init__()
        self.tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
        self.bert = DistilBertModel.from_pretrained("distilbert-base-uncased").to(device)
        for param in self.bert.parameters():
            param.requires_grad = False
        self.text_fc = nn.Linear(text_dim, hidden_dim)
        self.lstm = nn.LSTM(hidden_dim, hidden_dim, batch_first=True)
        self.text_final_fc = nn.Linear(hidden_dim, hidden_dim)
        self.fusion_fc = nn.Linear(hidden_dim * 2, hidden_dim)
        self.out = nn.Linear(hidden_dim, 1)
        for name, param in self.lstm.named_parameters():
            if 'weight' in name:
                nn.init.xavier_uniform_(param)
        self.to(device)

    def forward(self, texts, metrics=None):
        full_text = " ".join(texts)
        encodings = self.tokenizer(full_text, return_tensors="pt", padding=True, truncation=True, max_length=128)
        input_ids = encodings["input_ids"].to(device)
        attention_mask = encodings["attention_mask"].to(device)
        bert_output = self.bert(input_ids, attention_mask=attention_mask)
        text_pooled = bert_output.last_hidden_state
        text_embed = torch.relu(self.text_fc(text_pooled))
        text_embed = text_embed.mean(dim=1).unsqueeze(1)
        lstm_out, (h_n, _) = self.lstm(text_embed)
        text_final = torch.relu(self.text_final_fc(lstm_out.squeeze(1)))
        dummy_embed = torch.zeros_like(text_final, requires_grad=False)
        combined = torch.cat((text_final, dummy_embed), dim=-1)
        fused = torch.relu(self.fusion_fc(combined))
        score = torch.nn.functional.softplus(self.out(fused))
        return score.squeeze()

class TransformerCreativityAggregator(nn.Module):
    def __init__(self, text_dim=768, metric_dim=9, hidden_dim=64, num_layers=4, num_heads=4):
        super(TransformerCreativityAggregator, self).__init__()
        self.tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
        self.bert = DistilBertModel.from_pretrained("distilbert-base-uncased").to(device)
        self.text_fc = nn.Linear(text_dim, hidden_dim)
        self.embedding = nn.Linear(metric_dim, hidden_dim)
        self.transformer = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(
                d_model=hidden_dim,
                nhead=num_heads,
                dim_feedforward=hidden_dim * 4,
                dropout=0.1,
                batch_first=True
            ),
            num_layers=num_layers
        )
        self.norm = nn.LayerNorm(hidden_dim)
        self.fusion_fc = nn.Linear(hidden_dim * 2, hidden_dim)
        self.out = nn.Linear(hidden_dim, 1)
        self.to(device)

    def forward(self, texts, metrics):
        encodings = self.tokenizer(texts, return_tensors="pt", padding=True, truncation=True, max_length=128)
        input_ids = encodings["input_ids"].to(device)
        attention_mask = encodings["attention_mask"].to(device)
        with torch.no_grad():
            text_outputs = self.bert(input_ids, attention_mask=attention_mask)
        text_pooled = text_outputs.last_hidden_state[:, 0, :]
        text_embed = torch.relu(self.text_fc(text_pooled)).mean(dim=0)
        x = self.embedding(metrics)
        x = self.transformer(x)
        metric_embed = self.norm(x.mean(dim=0))
        combined = torch.cat((text_embed, metric_embed), dim=-1)
        fused = torch.relu(self.fusion_fc(combined))
        return torch.nn.functional.softplus(self.out(fused))

class TextBasedCreativityPredictor(nn.Module):
    def __init__(self, pretrained_model="distilbert-base-uncased", hidden_dim=64):
        super(TextBasedCreativityPredictor, self).__init__()
        self.tokenizer = DistilBertTokenizer.from_pretrained(pretrained_model)
        self.bert = DistilBertModel.from_pretrained(pretrained_model).to(device)
        self.fc1 = nn.Linear(768, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.weight_out = nn.Linear(hidden_dim, 9)
        self.to(device)

    def forward(self, texts, metrics):
        full_text = " ".join(texts)
        encodings = self.tokenizer(full_text, return_tensors="pt", padding=True, truncation=True, max_length=128)
        input_ids = encodings["input_ids"].to(device)
        attention_mask = encodings["attention_mask"].to(device)
        with torch.no_grad():
            outputs = self.bert(input_ids, attention_mask=attention_mask)
        pooled_output = outputs.last_hidden_state[:, 0, :]
        x = torch.relu(self.fc1(pooled_output))
        x = torch.relu(self.fc2(x))
        weights = torch.softmax(self.weight_out(x), dim=-1)
        metrics_tensor = metrics.clone().detach().to(device)
        scores = torch.matmul(metrics_tensor, weights.T)
        return scores.mean()

# Contrastive data storage
def load_existing_contrastive_data(file_path):
    if os.path.exists(file_path):
        try:
            with open(file_path, 'rb') as f:
                data = pickle.load(f)
            print(f"Loaded {len(data)} existing contrastive pairs from {file_path}.")
            if data and isinstance(data[0], tuple):
                data = [{"metrics": pair, "text": ([], [])} for pair in data]
            return data
        except Exception as e:
            print(f"Error loading {file_path}: {e}. Starting fresh.")
            return []
    print(f"No existing contrastive pairs found at {file_path}. Starting fresh.")
    return []

# Contrastive data storage
def load_existing_contrastive_data(file_path):
    if os.path.exists(file_path):
        try:
            with open(file_path, 'rb') as f:
                data = pickle.load(f)
            print(f"Loaded {len(data)} existing contrastive pairs from {file_path}.")
            if data and isinstance(data[0], tuple):
                data = [{"metrics": pair, "text": ([], [])} for pair in data]
            return data
        except Exception as e:
            print(f"Error loading {file_path}: {e}. Starting fresh.")
            return []
    print(f"No existing contrastive pairs found at {file_path}. Starting fresh.")
    return []

def save_contrastive_data(data, file_path):
    try:
        with open(file_path, 'wb') as f:
            pickle.dump(data, f)
        print(f"Saved {len(data)} contrastive pairs to {file_path}.")
    except Exception as e:
        print(f"Error saving to {file_path}: {e}")


# Domain-specific contrastive data collection
def collect_contrastive_pair_domain(domain):
    generated_prompts = generate_contrastive_prompts_gpt2(num_prompts=10, domain=domain, max_length=100)
    prompt = random.choice(generated_prompts)
    print(f"\n[{domain} domain] Using generated prompt: {prompt}")
    text_A = generate_text_openai(prompt)
    text_B = generate_text_openai(prompt)
    judgment = get_contrastive_judgment_auto(text_A, text_B)
    print(f"GPT-4 judged that Text {judgment} is more creative.")
    preferred_text, other_text = (text_A, text_B) if judgment == "A" else (text_B, text_A)
    baseline = load_or_compute_baseline(domain, gpt2_tokenizer, baselines_dir)
    segments_preferred = segment_text(preferred_text)
    segments_other = segment_text(other_text)
    if not segments_preferred or not segments_other:
        print("Empty segmentation encountered, reattempting.")
        return collect_contrastive_pair_domain(domain)

    metrics_preferred = []
    metrics_other = []

    # Compute metrics for preferred segments
    for seg in segments_preferred:
        N, _ = compute_novelty_with_attribution(seg, baseline)
        C = compute_coherence(seg)
        CT = compute_contextual_fit(prompt, seg)
        sc_vals = compute_syntactic_complexity(seg)
        SC = np.mean([sc_vals["avg_sentence_length"] / 3, sc_vals["avg_clause_count"] * 2,
                      sc_vals["avg_parse_tree_depth"] * 1.428, sc_vals["pos_entropy"] / 4.09 * 10])
        LD = np.mean([d for d in compute_ngram_diversity(seg)] + [1 - compute_self_bleu(seg) / 100])
        LN = dj_search(seg, baseline_corpora[domain], L=5, wmd_threshold=0.95)
        S = compute_surprise(seg)
        EE = compute_emotional_expressiveness(seg)
        HL = np.exp(-compute_perplexity(seg) / 50)
        if None not in [N, C, CT, SC, LD, LN, S, EE, HL]:
            metrics_preferred.append([N, C, CT, SC, LD, LN, S, EE, HL])

    # Compute metrics for other segments
    for seg in segments_other:
        N, _ = compute_novelty_with_attribution(seg, baseline)
        C = compute_coherence(seg)
        CT = compute_contextual_fit(prompt, seg)
        sc_vals = compute_syntactic_complexity(seg)
        SC = np.mean([sc_vals["avg_sentence_length"] / 3, sc_vals["avg_clause_count"] * 2,
                      sc_vals["avg_parse_tree_depth"] * 1.428, sc_vals["pos_entropy"] / 4.09 * 10])
        LD = np.mean([d for d in compute_ngram_diversity(seg)] + [1 - compute_self_bleu(seg) / 100])
        LN = dj_search(seg, baseline_corpora[domain], L=5, wmd_threshold=0.95)
        S = compute_surprise(seg)
```

```python
        EE = compute_emotional_expressiveness(seg)
        HL = np.exp(-compute_perplexity(seg) / 50)
        if None not in [N, C, CT, SC, LD, LN, S, EE, HL]:
            metrics_other.append([N, C, CT, SC, LD, LN, S, EE, HL])

    # Check if metrics are valid
    if not metrics_preferred or not metrics_other:
        print("No valid metrics, reattempting.")
        return collect_contrastive_pair_domain(domain)

    # Convert to tensors
    features_preferred = torch.tensor(metrics_preferred, dtype=torch.float32)
    features_other = torch.tensor(metrics_other, dtype=torch.float32)
    return {
        "metrics": (features_preferred, features_other),
        "text": (segments_preferred, segments_other)
    }

def collect_contrastive_training_examples_domain(domain, num_pairs=20):
    file_path = contrastive_file[domain]
    examples = load_existing_contrastive_data(file_path)  # Load pre-existing data

    pairs_to_collect = max(0, num_pairs)
    for i in range(pairs_to_collect):
        print(f"\n[{domain} domain] Collecting contrastive pair {i+1}/{num_pairs}")
        pair = collect_contrastive_pair_domain(domain)
        examples.append(pair)
        save_contrastive_data(examples, file_path)  # Save incrementally to avoid data loss

    return examples

def collect_all_contrastive_examples(num_pairs_per_domain=20):
    all_pairs = {"metrics": [], "text": []}
    for d in domains:
        print(f"\nCollecting/loading contrastive pairs for {d} domain:")
        domain_pairs = collect_contrastive_training_examples_domain(d, num_pairs=num_pairs_per_domain)
        all_pairs["metrics"].extend([pair["metrics"] for pair in domain_pairs])
        all_pairs["text"].extend([pair["text"] for pair in domain_pairs])
    total_pairs = len(all_pairs["metrics"])
    print(f"Total contrastive pairs (loaded + collected): {total_pairs}")
    return all_pairs

# Training function
def train_composite_model(model, training_pairs, num_epochs=50, lr=5e-4, margin=1.0, patience=5):
    model = model.to(device)
    optimizer = optim.AdamW(model.parameters(), lr=lr if not isinstance(model, TextBasedCreativityPredictor) else 2e-5, weig
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=5)
    warmup_scheduler = torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda=lambda epoch: min(epoch / 5, 1) if epoch < 5 e
    criterion = nn.MarginRankingLoss(margin=margin)
    train_pairs, val_pairs = train_test_split(training_pairs, test_size=0.2, random_state=42)
    best_val_loss = float('inf')
    patience_counter = 0
    for epoch in range(num_epochs if not isinstance(model, TextBasedCreativityPredictor) else 100):
        model.train()
        train_loss = 0.0
        train_pairs_count = 0
        for pair in train_pairs:
            if isinstance(pair, dict):
                metrics_A, metrics_B = pair["metrics"]
                text_A, text_B = pair["text"]
                metrics_A, metrics_B = metrics_A.to(device), metrics_B.to(device)
                if isinstance(model, SimpleCreativityPredictor):
                    score_A = model(text_A)
                    score_B = model(text_B)
                else:
                    score_A = model(text_A, metrics_A)
                    score_B = model(text_B, metrics_B)
            else:
                text_A, text_B = pair
                score_A = model(text_A)
                score_B = model(text_B)
            score_A = torch.tensor([score_A], device=device) if not torch.is_tensor(score_A) else score_A.view(1)
            score_B = torch.tensor([score_B], device=device) if not torch.is_tensor(score_B) else score_B.view(1)
            target = torch.tensor([1.0], device=device)
            loss = criterion(score_A, score_B, target)
            if not torch.isnan(loss):
                optimizer.zero_grad()
                loss.backward()
                torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
                optimizer.step()
                train_loss += loss.item()
                train_pairs_count += 1
                if train_pairs_count % 100 == 0:
                    print(f"Epoch {epoch+1}, Pair {train_pairs_count}, Scores - A: {score_A.item():.4f}, B: {score_B.item():
        if epoch < 5:
            warmup_scheduler.step()
        train_loss = train_loss / train_pairs_count if train_pairs_count > 0 else float('inf')
        model.eval()
        val_loss = 0.0
        val_pairs_count = 0
        with torch.no_grad():
            for pair in val_pairs:
                if isinstance(pair, dict):
                    metrics_A, metrics_B = pair["metrics"]
                    text_A, text_B = pair["text"]
                    metrics_A, metrics_B = metrics_A.to(device), metrics_B.to(device)
                    if isinstance(model, SimpleCreativityPredictor):
                        score_A = model(text_A)
                        score_B = model(text_B)
                    else:
                        score_A = model(text_A, metrics_A)
                        score_B = model(text_B, metrics_B)
                else:
                    text_A, text_B = pair
                    score_A = model(text_A)
                    score_B = model(text_B)
                score_A = torch.tensor([score_A], device=device) if not torch.is_tensor(score_A) else score_A.view(1)
                score_B = torch.tensor([score_B], device=device) if not torch.is_tensor(score_B) else score_B.view(1)
                target = torch.tensor([1.0], device=device)
                loss = criterion(score_A, score_B, target)
                if not torch.isnan(loss):
                    val_loss += loss.item()
                    val_pairs_count += 1
        val_loss = val_loss / val_pairs_count if val_pairs_count > 0 else float('inf')
        print(f"Epoch {epoch+1}/{num_epochs if not isinstance(model, TextBasedCreativityPredictor) else 100}, Train Loss: {t
        scheduler.step(val_loss)
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print(f"Early stopping at epoch {epoch+1}")
                break
    return model
```

```
# ==============================================================================
# FIXED TEST PROMPTS PER DOMAIN
# ==============================================================================
test_prompts_by_domain = {
    "literary": [
        "Write a sonnet about quantum physics.",
        "Create a short story with an unexpected twist.",
        "Generate a creative narrative in the style of magical realism."
    ],
     "technical": [
         "Write a research paper abstract on a new AI algorithm.",
         "Generate a technical description of a cloud computing architecture.",
         "Describe a novel method for data encryption in a research abstract."
    ],
}

# ==============================================================================
# TESTING FUNCTION FOR A DOMAIN (With Additional Analysis)
# ==============================================================================
# Testing function
def evaluate_output(output, domain, models):
    segments = segment_text(output)
    baseline = load_or_compute_baseline(domain, gpt2_tokenizer, baselines_dir)

    # Compute all nine metric features for models requiring them
    metrics = []
    for seg in segments:
        N = compute_novelty(seg, baseline)
        C = compute_coherence(seg)
        CT = compute_contextual_fit(output, seg)  # Using full output as prompt context
        sc_vals = compute_syntactic_complexity(seg)
        SC = np.mean([sc_vals["avg_sentence_length"] / 3, sc_vals["avg_clause_count"] * 2,
                      sc_vals["avg_parse_tree_depth"] * 1.428, sc_vals["pos_entropy"] / 4.09 * 10])
        LD = np.mean([d for d in compute_ngram_diversity(seg)] + [1 - compute_self_bleu(seg) / 100])
        LN = dj_search(seg, baseline_corpora[domain], L=5, wmd_threshold=0.95)
        S = compute_surprise(seg)
        EE = compute_emotional_expressiveness(seg)
        HL = np.exp(-compute_perplexity(seg) / 50)
        # Only include if all metrics are valid
        if None not in [N, C, CT, SC, LD, LN, S, EE, HL]:
            metrics.append([N, C, CT, SC, LD, LN, S, EE, HL])

    features = torch.tensor(metrics, dtype=torch.float32).to(device) if metrics else None

    # Model predictions
    scores = {}
    for name, model in models.items():
        model.eval()
        with torch.no_grad():
            if features is None or not segments:
                scores[name] = 0.0
            elif isinstance(model, SimpleCreativityPredictor):
                scores[name] = model(segments).item()
            else:
                scores[name] = model(segments, features).item()

    # Compute all nine metrics for composite score (averaged across segments)
    scores.update({
        "Novelty": np.mean([compute_novelty(seg, baseline) for seg in segments]) if segments else 0.0,
        "Coherence": np.mean([compute_coherence(seg) or 0.0 for seg in segments]) if segments else 0.0,
        "ContextualFit": np.mean([compute_contextual_fit(output, seg) for seg in segments]) if segments else 0.0,
        "SyntacticComplexity": np.mean([min(10, v) for v in [
            compute_syntactic_complexity(output)["avg_sentence_length"] / 3,
            compute_syntactic_complexity(output)["avg_clause_count"] * 2,
            compute_syntactic_complexity(output)["avg_parse_tree_depth"] * 1.428,
            compute_syntactic_complexity(output)["pos_entropy"] / 4.09 * 10
        ]]),
        "LexicalDiversity": np.mean([10 * d for d in compute_ngram_diversity(output)] +
                                    [10 * (1 - compute_self_bleu(output) / 100)]),
        "LexicalNovelty": compute_lexical_novelty(output, domain),
        "Surprise": compute_surprise(output) * 5,
        "EmotionalExpressiveness": min(10, compute_emotional_expressiveness(output) * 10),
        "HumanLikeness": 10 * np.exp(-compute_perplexity(output) / 50)
    })

    composite_score = sum(scores.values()) / len(scores)
    return scores, composite_score

# Testing function
def test_model_for_domain(domain, models, fixed_weights):
    baseline = load_or_compute_baseline(domain, gpt2_tokenizer, baselines_dir)
    results = {name: {"fixed_scores": [], "model_scores": []} for name in models}
    outputs_all = []
    segments_for_analysis = None
    prompt_for_analysis = test_prompts_by_domain[domain][0]
    for prompt in test_prompts_by_domain[domain]:
        print(f"\n[Domain: {domain.capitalize()} | Test Prompt: {prompt}]")
        for i in range(5):
            text = generate_text_openai(prompt)
            outputs_all.append(text)
            segments = segment_text(text)
            if segments and segments_for_analysis is None:
                segments_for_analysis = segments
            if not segments:
                fixed_score = 0.0
                model_scores = {name: 0.0 for name in models}
            else:
                fixed_score = aggregate_creativity_index_normalized(segments, prompt, baseline, fixed_weights, length_weight,
                model_scores, _ = evaluate_output(text, domain, models)
            for name in models:
                results[name]["fixed_scores"].append(fixed_score)
                results[name]["model_scores"].append(model_scores[name])
            print(f"\nOutput {i+1}:\n{text}")
            print(f"Normalized Fixed Composite Score: {fixed_score:.4f}")
            print(f"Model Scores: {', '.join(f'{name}: {score:.4f}' for name, score in model_scores.items())}")
    return outputs_all, results, segments_for_analysis, prompt_for_analysis, baseline

# Compare Together AI models
def compare_together_ai_models(trained_models, fixed_weights_by_domain, num_texts=10):
    together_models = [
        "meta-llama/Llama-3.3-70B-Instruct-Turbo-Free",
        "mistralai/Mixtral-8x7B-Instruct-v0.1",
        "Qwen/Qwen2-72B-Instruct"
    ]
    all_prompts = [p for prompts in test_prompts_by_domain.values() for p in prompts]
    results = {model: {"fixed": [], **{name: [] for name in trained_models}} for model in together_models}
    for _ in range(num_texts):
        prompt = random.choice(all_prompts)
        domain = "technical" if prompt in test_prompts_by_domain["technical"] else "literary"
        baseline = load_or_compute_baseline(domain, gpt2_tokenizer, baselines_dir)
        for t_model in together_models:
            print(f"\nTesting Together AI model: {t_model} with prompt: {prompt}")
            text = generate_text_together(prompt, model_name=t_model)
            segments = segment_text(text)
            if not segments:
                fixed_score = 0.0
                model_scores = {name: 0.0 for name in trained_models}
            else:
                fixed_score = aggregate_creativity_index_normalized(segments, prompt, baseline, fixed_weights_by_domain[domai
                model_scores, _ = evaluate_output(text, domain, trained_models)
            results[t_model]["fixed"].append(fixed_score)
            for name in trained_models:
                results[t_model][name].append(model_scores[name])
            print(f"{t_model} - Fixed: {fixed_score:.4f}, Model Scores: {model_scores}")
    for metric in ["fixed"] + list(trained_models.keys()):
        plt.figure(figsize=(12, 6))
        for t_model in together_models:
            scores = results[t_model][metric]
            plt.plot(range(len(scores)), scores, label=t_model, marker='o')
        plt.title(f"Creativity Scores Across Together AI Models ({metric})")
        plt.xlabel("Text Sample")
        plt.ylabel("Score")
        plt.legend()
        plt.grid(True)
        plt.savefig(os.path.join(plots_dir, f"together_ai_comparison_{metric}.png"))
        plt.close()
        print(f"Saved comparison plot for {metric} to: {os.path.join(plots_dir, f'together_ai_comparison_{metric}.png')}")
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Move GPT-2 model to device
gpt2_model.to(device)
```

```
Using device: cuda
GPT2LMHeadModel(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-11): 12 x GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D(nf=2304, nx=768)
          (c_proj): Conv1D(nf=768, nx=768)
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.1, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D(nf=3072, nx=768)
          (c_proj): Conv1D(nf=768, nx=3072)
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=768, out_features=50257, bias=False)
)
```

Double-click (or enter) to edit

```
trained_models = {}
def main_demo():
    fixed_weights_by_domain = {
        "literary": (1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9),  # N, C, CT, SC, LD, LN, S, EE, HL
        "technical": (1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9),
    }

    print("\nStandard Evaluation (Fixed Weights - Normalized):")
    for d in domains:
        print(f"\n{'='*44}\nDomain: {d.capitalize()}")
        baseline = load_or_compute_baseline(d, gpt2_tokenizer, baselines_dir)
        fixed_prompts = {
            "constrained": "Write a sonnet about quantum physics.",
            "semi_constrained": "Create a robot story.",
            "unconstrained": "Generate creative output.",
        }
        for pt, prompt in fixed_prompts.items():
            text = generate_text_openai(prompt)
            segments = segment_text(text)
            composite_index = aggregate_creativity_index_normalized(segments, prompt, baseline, fixed_weights_by_domain[d], l
            print(f"[{d} | {pt}] Normalized Composite Creativity Index: {composite_index:.4f}")

    print("\nCollecting contrastive training pairs from all domains...")
    all_training_pairs = collect_all_contrastive_examples(num_pairs_per_domain=0)

    print("\nTraining the domain-agnostic composite models...")
    combined_pairs = [{"metrics": m, "text": t} for m, t in zip(all_training_pairs["metrics"], all_training_pairs["text"])]
    models = {
        "CompositeRegressor": CompositeRegressor(),
        "SimpleCreativityPredictor": SimpleCreativityPredictor(),
        "TransformerCreativityAggregator": TransformerCreativityAggregator(),
        "TextBasedCreativityPredictor": TextBasedCreativityPredictor()
    }
    for model_name, model in models.items():
        print(f"\nTraining {model_name}...")
        if model_name == "TextBasedCreativityPredictor":
            trained_models[model_name] = train_composite_model(model, combined_pairs, num_epochs=50, lr=1e-4, margin=1.0, pat
        else:
            trained_models[model_name] = train_composite_model(model, combined_pairs, num_epochs=50, lr=1e-5, margin=1.0, pat

    print("\nAutomated Testing & Analysis:")
    domain_results = {}
    for d in domains:
        print(f"\n--- Testing and Analysis for Domain: {d.capitalize()} ---")
        outputs, results, segments, prompt, baseline = test_model_for_domain(d, trained_models, fixed_weights_by_domain[d])
        for model_name in trained_models:
          additional_analysis(d, results[model_name]["fixed_scores"], results[model_name]["model_scores"], segments, prompt,
          domain_results[f"{d}_{model_name}"] = (results[model_name]["fixed_scores"], results[model_name]["model_scores"])

    print("\nComparing creativity across Together AI models...")
    compare_together_ai_models(trained_models, fixed_weights_by_domain, num_texts=10)

main_demo()
```

```
Epoch 11, Pair 700, Scores - A: 4.2872, B: 5.7290, Loss: 2.4418
Epoch 11, Pair 800, Scores - A: 2.8742, B: 3.7047, Loss: 1.8305
Epoch 11, Pair 900, Scores - A: 3.5922, B: 2.6445, Loss: 0.0523
Epoch 11, Pair 1000, Scores - A: 6.2472, B: 3.7913, Loss: 0.0000
Epoch 11, Pair 1100, Scores - A: 4.2362, B: 4.0081, Loss: 0.7719
Epoch 11, Pair 1200, Scores - A: 3.7645, B: 2.4726, Loss: 0.0000
Epoch 11, Pair 1300, Scores - A: 4.1091, B: 3.7102, Loss: 0.6011
Epoch 11, Pair 1400, Scores - A: 3.0731, B: 2.8503, Loss: 0.7773
```

```
# import os
# import random
# import string
# import torch
# import matplotlib.pyplot as plt
# from sacrebleu import sentence_bleu
# from nltk.corpus import stopwords

# # Fixed weights for each domain (already defined)
# fixed_weights_by_domain = {
#     "literary": (1/9,)*9,
#     "technical": (1/9,)*9,
# }

# # ==========================================================
# # New Relevant Corpus Segmentation Function
# # ==========================================================
# def segment_relevant_corpus(prompt, corpus, stopwords_set=None):
#     """
#     Instead of segmenting the entire corpus, return only the paragraphs
#     from the corpus that contain at least one keyword from the prompt.
#     The prompt is cleaned (punctuation removed and lower-cased) and compared
#     against each paragraph.
#     """
#     if stopwords_set is None:
#         stopwords_set = set(stopwords.words('english'))
#     # Remove punctuation and lowercase the prompt.
#     translator = str.maketrans('', '', string.punctuation)
#     prompt_clean = prompt.translate(translator).lower()
#     prompt_tokens = set(prompt_clean.split()) - stopwords_set

#     paragraphs = [para.strip() for para in corpus.split("\n\n") if para.strip()]
#     relevant_paras = []
#     for para in paragraphs:
#         para_lower = para.lower()
#         if any(keyword in para_lower for keyword in prompt_tokens):
#             relevant_paras.append(para)
#     if relevant_paras:
#         return "\n\n".join(relevant_paras)
#     else:
#         # Fallback: return the entire corpus if no paragraphs are found.
#         return corpus

# # ==========================================================
# # Updated Reference Retrieval: Use Relevant Segments
# # ==========================================================
# def get_reference_text(domain, prompt, baseline_corpora):
#     """
#     For the given domain, filter the baseline corpus to only those segments
#     that are relevant to the prompt. The resulting text is used as the reference.
#     """
#     corpus = baseline_corpora[domain]
#     relevant_text = segment_relevant_corpus(prompt, corpus)
#     return relevant_text

# # ==========================================================
# # Utility functions for n-gram based uniqueness (unchanged)
# # ==========================================================
# def get_ngrams(text, n, tokenizer):
#     tokens = tokenizer.tokenize(text)
#     return {tuple(tokens[i:i+n]) for i in range(len(tokens) - n + 1)}

# def compute_ngram_uniqueness(text, n, reference_text, tokenizer):
#     text_ngrams = get_ngrams(text, n, tokenizer)
#     ref_ngrams = get_ngrams(reference_text, n, tokenizer)
#     if not text_ngrams:
#         return 0.0
#     unique_count = sum(1 for ng in text_ngrams if ng not in ref_ngrams)
#     return unique_count / len(text_ngrams)

# def compute_creativity_index(text, reference_text, tokenizer, n_min=5, n_max=12):
#     total_uniqueness = 0.0
#     for n in range(n_min, n_max+1):
#         total_uniqueness += compute_ngram_uniqueness(text, n, reference_text, tokenizer)
#     return total_uniqueness
# def compute_bleu_uniqueness(text, reference_text):
#     try:
#         bleu = sentence_bleu(text, [reference_text]).score
#     except Exception:
#         bleu = 0.0
#     uniqueness = 1 - (bleu / 100)
#     return bleu, uniqueness

# # ===========================================================
# # External Text Generation Function (unchanged)
# # ===========================================================
# def generate_text_openai(prompt):
#     response = openai.chat.completions.create(
#         model="gpt-4o",
#         messages=[
#             {"role": "system", "content": "You are a creative text generator."},
#             {"role": "user", "content": prompt}
#         ],
#         temperature=0.8,
#         top_p=0.95,
#         max_completion_tokens=200
#     )
#     return response.choices[0].message.content

# # ===========================================================
# # Additional Analysis Pipeline: Compare Outputs from All Models
# # ===========================================================
# def additional_analysis_pipeline_all_models(trained_models, fixed_weights_by_domain, baseline_corpora):
#     results = {domain: {} for domain in domains}

#     for domain in domains:
#         prompt = test_prompts_by_domain[domain][0]
#         print(f"\n[Additional Analysis] Domain: {domain.capitalize()}, Prompt: {prompt}")

#         # Use the prompt to filter the baseline corpus for relevant text.
#         reference_text = get_reference_text(domain, prompt, baseline_corpora)

#         # Generate 5 texts using the external generator.
#         generated_texts = []
#         for i in range(5):
#             text = generate_text_openai(prompt)
#             generated_texts.append(text)
#             print(f"Generated Text {i+1} (first 100 chars): {text[:100]}...")

#         for model_name, model in trained_models.items():
#             model_results = []
#             print(f"\nEvaluating with model: {model_name}")
#             for text in generated_texts:
#                 # Evaluate the text.
#                 scores, composite_score = evaluate_output(text, domain, {model_name: model})
#                 creat_index = compute_creativity_index(text, reference_text, gpt2_tokenizer, n_min=5, n_max=12)
#                 bleu_score, bleu_uniqueness = compute_bleu_uniqueness(text, reference_text)
#                 model_results.append({
#                     "text": text,
#                     "composite_score": composite_score,
#                     "creativity_index": creat_index,
#                     "bleu_score": bleu_score,
#                     "bleu_uniqueness": bleu_uniqueness
#                 })
#                 print(f"  Text: Composite={composite_score:.4f}, CreativityIndex={creat_index:.4f}, "
#                       f"BLEU={bleu_score:.2f}, BLEU-Uniqueness={bleu_uniqueness:.4f}")
#             results[domain][model_name] = model_results

#             # Plot: Composite vs. Creativity Index.
#             comp_scores = [res["composite_score"] for res in model_results]
#             creat_indices = [res["creativity_index"] for res in model_results]
#             plt.figure(figsize=(8, 6))
#             plt.scatter(comp_scores, creat_indices, color='blue', marker='o')
#             plt.title(f"{domain.capitalize()} - {model_name}: Composite vs. Creativity Index")
#             plt.xlabel("Composite Creativity Index")
#             plt.ylabel("Creativity Index (Sum of n-uniqueness, n=5..12)")
#             plt.grid(True)
#             plt.savefig(os.path.join(plots_dir, f"{domain}_{model_name}_composite_vs_creativity_index.png"))
#             plt.close()

#             # Plot: Composite vs. BLEU Uniqueness.
#             bleu_uniqueness_scores = [res["bleu_uniqueness"] for res in model_results]
#             plt.figure(figsize=(8, 6))
#             plt.scatter(comp_scores, bleu_uniqueness_scores, color='green', marker='o')
#             plt.title(f"{domain.capitalize()} - {model_name}: Composite vs. BLEU Uniqueness")
#             plt.xlabel("Composite Creativity Index")
#             plt.ylabel("BLEU Uniqueness Score (1 - BLEU/100)")
#             plt.grid(True)
#             plt.savefig(os.path.join(plots_dir, f"{domain}_{model_name}_composite_vs_bleu_uniqueness.png"))
#             plt.close()

#     return results

# # ============================================================
# # Run the pipeline for all models using relevant segmentation
# # ============================================================
# all_models_results = additional_analysis_pipeline_all_models(trained_models, fixed_weights_by_domain, baseline_corpora)
```

```
[Additional Analysis] Domain: Literary, Prompt: Write a sonnet about quantum physics.
Generated Text 1 (first 100 chars): In realms unseen, where particles reside,
A dance of chance, the universe unfolds.
Within the qu...
Generated Text 2 (first 100 chars): In realms where certainty dissolves to haze,
A dance of particles unseen, they weave,
A tapestry...
Generated Text 3 (first 100 chars): In realms unseen by naked, mortal eyes,
A dance of particles both wild and small,
Where certaint...
Generated Text 4 (first 100 chars): In realms where certainty dissolves to haze,
The dance of atoms weaves a mystic thread.
Each par...
Generated Text 5 (first 100 chars): In realms unseen, where mysteries unfold,
The dance of particles defies our sight;
In quantum fi...

Evaluating with model: CompositeRegressor
  Text: Composite=3.5333, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.5819, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.5870, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.4516, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.5934, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000

Evaluating with model: SimpleCreativityPredictor
  Text: Composite=3.3994, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.3997, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.4277, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.3329, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.4246, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000

Evaluating with model: TransformerCreativityAggregator
  Text: Composite=3.5155, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.5512, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.5687, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.4442, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.5701, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000

Evaluating with model: TextBasedCreativityPredictor
  Text: Composite=3.2541, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.3143, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.3673, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.2019, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000
  Text: Composite=3.3685, CreativityIndex=8.0000, BLEU=0.00, BLEU-Uniqueness=1.0000

[Additional Analysis] Domain: Technical, Prompt: Write a research paper abstract on a new AI algorithm.
Generated Text 1 (first 100 chars): Title: Enhancing Autonomous Decision-Making: The Synergy Algorithm for Multi-Agent S

Abstract...
Generated Text 2 (first 100 chars): Title: "Neural Adaptive Framework for Enhanced Contextual Understanding in Conversat

Abstr...
Generated Text 3 (first 100 chars): **Abstract:**

In this paper, we introduce Neural Adaptive Meta-Learning (NAML), a novel algorithm d...
Generated Text 4 (first 100 chars): Title: "EfficientNet-GAN: A Novel Algorithm for High-Fidelity Image Synthesis"
```