02_story_generation

June 14, 2025

1 Story Generation Model Analysis

1.1 Academic Analysis of Large Language Models for Children's Bedtime Story Generation

Author: University of London Student
Project: Smart Visual Storyteller for Children
Analysis Date: June 2025

1.1.1 Abstract

This notebook presents a comprehensive analysis of ten state-of-the-art large language models for generating children's bedtime stories. We evaluate OpenAI GPT-4o/GPT-4o-mini, Anthropic Claude-3.5-sonnet/Claude-3.5-haiku, Google Gemini-2.0-flash/Gemini-1.5-pro, and DeepSeek deepseek-chat/deepseek-v3 models across multiple dimensions including cost efficiency, execution speed, content quality, and age-appropriateness.

1.1.2 Research Objectives

1. **Performance Evaluation**: Assess computational efficiency and cost-effectiveness
2. **Quality Analysis**: Evaluate story quality, structure, and appropriateness
3. **Comparative Assessment**: Rank models for optimal story generation
4. **Implementation Guidance**: Provide evidence-based model selection recommendations

1.2 0. Requirements Installation

Before running this analysis, ensure all required packages are installed. Run the following cell if you haven't installed the requirements yet:

[66]:
```
# Install required packages for analysis
# Uncomment and run if packages are missing:

# %pip install pandas matplotlib seaborn numpy scipy scikit-learn jupyter

# Or install from requirements file:
# %pip install -r ../requirements.txt

# Check if all packages are available
try:
```

```python
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from scipy import stats
    from sklearn.preprocessing import StandardScaler
    print("  All required packages are installed and available")
except ImportError as e:
    print(f"  Missing package: {e}")
    print("Please uncomment and run the pip install commands above")
```

```
 All required packages are installed and available
```

1.3 1. Environment Setup and Data Loading

[67]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import json
from scipy import stats
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')

# Configure plotting parameters for academic presentation
plt.style.use('default')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 12
plt.rcParams['axes.titlesize'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['xtick.labelsize'] = 10
plt.rcParams['ytick.labelsize'] = 10
plt.rcParams['legend.fontsize'] = 10

print("Analysis environment initialized successfully")
print(f"pandas version: {pd.__version__}")
print(f"numpy version: {np.__version__}")
```

```
Analysis environment initialized successfully
pandas version: 2.3.0
numpy version: 2.2.6
```

[68]:
```python
# Load the most recent story generation results
results_dir = Path('../../results/story_generation')
```

```
csv_files = list(results_dir.glob('story_generation_results_*.csv'))

if not csv_files:
    raise FileNotFoundError("No story generation results found. Please run␣
    ↪02_story_generation_collect.py first.")

# Use the most recent results file
latest_file = max(csv_files, key=lambda x: x.stat().st_mtime)
print(f"Loading data from: {latest_file.name}")

# Load the dataset
df = pd.read_csv(latest_file)

print(f"Dataset loaded successfully")
print(f"Total records: {len(df)}")
print(f"Unique models: {df['story_model'].nunique()}")
print(f"Unique images: {df['image_file'].nunique()}")
print(f"Date range: {latest_file.stat().st_mtime}")
```

```
Loading data from: story_generation_results_20250613_231958.csv
Dataset loaded successfully
Total records: 240
Unique models: 15
Unique images: 16
Date range: 1749847628.1840665
```

[69]:
```
# Data exploration and basic information
print("=== DATASET CHARACTERIZATION ===\n")

print("Story generation models evaluated:")
for model in df['story_model'].unique():
    print(f"  - {model}")

print(f"\nTest images:")
for image in df['image_file'].unique():
    print(f"  - {image}")

print(f"\nData completeness assessment:")
expected_combinations = len(df['story_model'].unique()) * len(df['image_file'].
    ↪unique())
print(f"  - Expected combinations: {len(df['story_model'].unique())} models ×␣
    ↪{len(df['image_file'].unique())} images = {expected_combinations}")
print(f"  - Actual records: {len(df)}")
print(f"  - Missing records: {expected_combinations - len(df)}")

# Data quality assessment
print(f"\nData quality assessment:")
```

```python
print(df.isnull().sum())

# Display dataset structure
print(f"\nDataset columns:")
print(list(df.columns))
print(f"\nSample data:")
df.head()
```

```
=== DATASET CHARACTERIZATION ===

Story generation models evaluated:
  - gpt-4o
  - gpt-4o-mini
  - claude-opus-4
  - claude-sonnet-4
  - claude-3.7-sonnet
  - claude-3.5-sonnet
  - claude-3.5-haiku
  - gemini-2.0-flash
  - gemini-2.0-flash-lite
  - gemini-1.5-pro
  - gemini-1.5-flash
  - mistral-large-latest
  - mistral-medium-latest
  - mistral-small-latest
  - deepseek-chat

Test images:
  - toy_01.jpeg
  - toy_02.jpeg
  - toy_03.jpeg
  - toy_04.jpeg
  - toy_05.jpeg
  - toy_06.jpeg
  - toy_07.jpeg
  - toy_08.jpeg
  - toy_09.jpeg
  - toy_10.jpeg
  - drawing_01.jpeg
  - drawing_02.jpeg
  - drawing_03.jpeg
  - drawing_04.jpeg
  - drawing_05.jpeg
  - drawing_06.jpeg

Data completeness assessment:
  - Expected combinations: 15 models × 16 images = 240
  - Actual records: 240
```

```
  - Missing records: 0

Data quality assessment:
image_file           0
image_type           0
image_caption        0
story_model          0
generated_story      0
execution_time       0
cost                 0
word_count           0
quality_score        0
meets_length_req     0
has_title            0
contains_dialogue    0
positive_tone        0
story_structure      0
age_appropriate      0
bedtime_suitable     0
dtype: int64

Dataset columns:
['image_file', 'image_type', 'image_caption', 'story_model', 'generated_story',
'execution_time', 'cost', 'word_count', 'quality_score', 'meets_length_req',
'has_title', 'contains_dialogue', 'positive_tone', 'story_structure',
'age_appropriate', 'bedtime_suitable']

Sample data:
```

[69]:

```
    image_file image_type                                      image_caption  \
0  toy_01.jpeg        toy  A doll with curly blonde hair and a red jumpsu…
1  toy_01.jpeg        toy  A doll with curly blonde hair and a red jumpsu…
2  toy_01.jpeg        toy  A doll with curly blonde hair and a red jumpsu…
3  toy_01.jpeg        toy  A doll with curly blonde hair and a red jumpsu…
4  toy_01.jpeg        toy  A doll with curly blonde hair and a red jumpsu…

         story_model                                    generated_story  \
0             gpt-4o  Title: Lily's Cozy Adventure\n\nOnce upon a ti…
1        gpt-4o-mini  Title: Daisy's Cozy Adventure\n\nOnce upon a t…
2      claude-opus-4  Title: Rosie's Couch Adventure\n\nRosie the do…
3    claude-sonnet-4  **Title: Rosie's Special Day**\n\nRosie the do…
4  claude-3.7-sonnet  # The Couch Adventure\n\nMolly the doll with c…

   execution_time      cost  word_count  quality_score  meets_length_req  \
0            4.52  0.004670         201          26.00              True
1            4.70  0.000172         185          24.00              True
2           11.63  0.018480         179          23.25              True
```

```
3           6.86  0.003681         169          22.00                True
4           6.14  0.003981         191          24.62                True

   has_title  contains_dialogue  positive_tone  story_structure  \
0       True               True           True             True
1       True               True           True             True
2       True               True           True             True
3       True               True           True             True
4      False               True           True             True

   age_appropriate  bedtime_suitable
0             True              True
1             True              True
2             True              True
3             True              True
4             True              True
```

1.4 2. Performance Metrics Analysis

[71]:

```
# Performance statistics by model
print("=== PERFORMANCE METRICS ANALYSIS ===\n")

# Aggregate performance statistics by model
performance_stats = df.groupby('story_model').agg({
    'execution_time': ['mean', 'std', 'min', 'max'],
    'cost': ['mean', 'std', 'min', 'max']
}).round(4)

performance_stats.columns = ['_'.join(col) for col in performance_stats.columns]
print("Performance Statistics by Model:")
print(performance_stats)

# Calculate efficiency metrics
print("\n=== EFFICIENCY METRICS ===")
efficiency_metrics = df.groupby('story_model').agg({
    'execution_time': 'mean',
    'cost': 'mean'
}).round(4)

# Add cost per second metric
efficiency_metrics['cost_per_second'] = (efficiency_metrics['cost'] /␣
↪efficiency_metrics['execution_time']).round(6)
efficiency_metrics['speed_rank'] = efficiency_metrics['execution_time'].rank()
efficiency_metrics['cost_rank'] = efficiency_metrics['cost'].rank()

print("Efficiency Rankings:")
print(efficiency_metrics.sort_values('execution_time'))
```

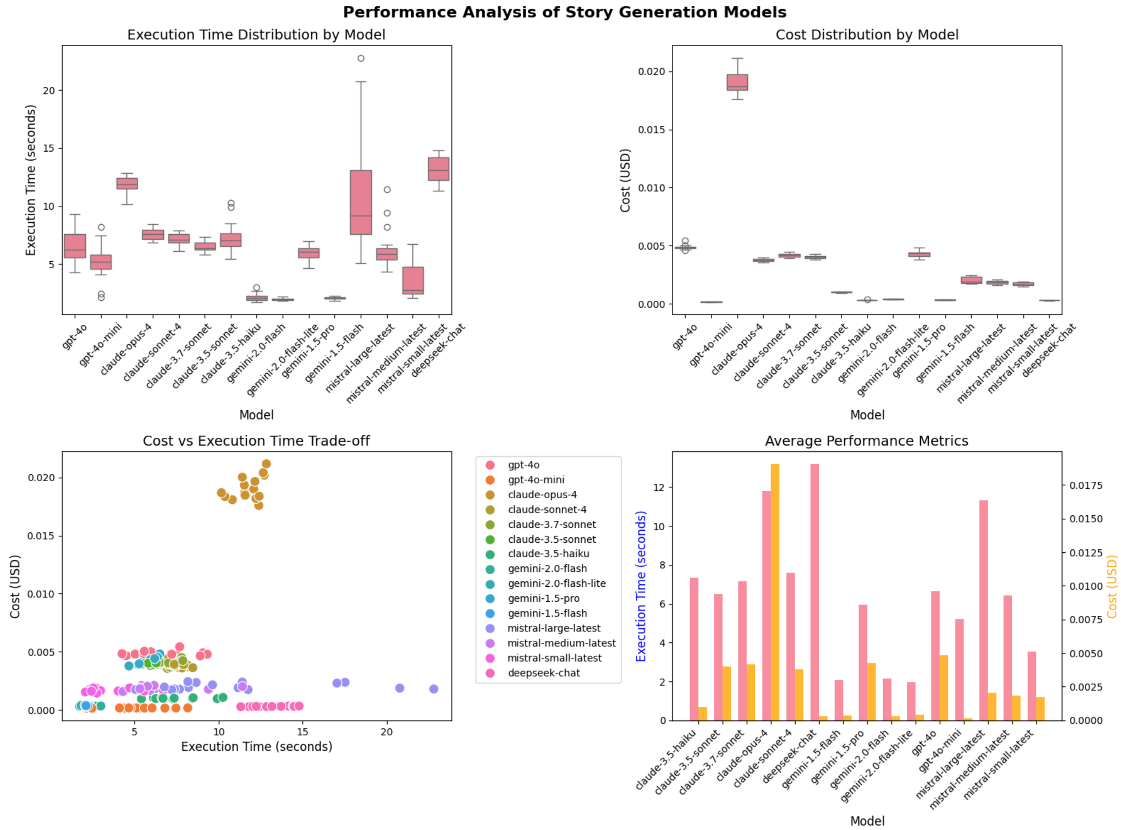
```
=== PERFORMANCE METRICS ANALYSIS ===

Performance Statistics by Model:
                       execution_time_mean  execution_time_std  \
story_model
claude-3.5-haiku                    7.3294              1.3386
claude-3.5-sonnet                   6.5094              0.4776
claude-3.7-sonnet                   7.1412              0.5581
claude-opus-4                      11.7831              0.8059
claude-sonnet-4                     7.5962              0.4994
deepseek-chat                      13.1644              1.1027
gemini-1.5-flash                    2.0706              0.1248
gemini-1.5-pro                      5.9519              0.5986
gemini-2.0-flash                    2.1431              0.3390
gemini-2.0-flash-lite               1.9762              0.1060
gpt-4o                              6.6425              1.5734
gpt-4o-mini                         5.2044              1.5751
mistral-large-latest               11.3275              5.3297
mistral-medium-latest               6.4062              1.8114
mistral-small-latest                3.5500              1.4485

                       execution_time_min  execution_time_max  cost_mean  \
story_model
claude-3.5-haiku                     5.42               10.26     0.0010
claude-3.5-sonnet                    5.83                7.36     0.0040
claude-3.7-sonnet                    6.14                7.88     0.0042
claude-opus-4                       10.17               12.84     0.0190
claude-sonnet-4                      6.86                8.47     0.0038
deepseek-chat                       11.32               14.78     0.0003
gemini-1.5-flash                     1.84                2.26     0.0003
gemini-1.5-pro                       4.68                6.95     0.0043
gemini-2.0-flash                     1.74                3.01     0.0003
gemini-2.0-flash-lite                1.84                2.19     0.0004
gpt-4o                               4.27                9.33     0.0049
gpt-4o-mini                          2.15                8.17     0.0002
mistral-large-latest                 5.08               22.78     0.0020
mistral-medium-latest                4.32               11.42     0.0019
mistral-small-latest                 2.09                6.72     0.0017

                       cost_std  cost_min  cost_max
story_model
claude-3.5-haiku         0.0000    0.0010    0.0011
claude-3.5-sonnet        0.0001    0.0038    0.0043
claude-3.7-sonnet        0.0002    0.0039    0.0045
claude-opus-4            0.0010    0.0176    0.0212
claude-sonnet-4          0.0001    0.0036    0.0040
deepseek-chat            0.0000    0.0003    0.0003
gemini-1.5-flash         0.0000    0.0003    0.0004
```

```
gemini-1.5-pro          0.0003    0.0038    0.0048
gemini-2.0-flash        0.0000    0.0003    0.0004
gemini-2.0-flash-lite   0.0000    0.0004    0.0005
gpt-4o                  0.0002    0.0046    0.0054
gpt-4o-mini             0.0000    0.0002    0.0002
mistral-large-latest    0.0003    0.0017    0.0024
mistral-medium-latest   0.0002    0.0016    0.0021
mistral-small-latest    0.0001    0.0015    0.0019

=== EFFICIENCY METRICS ===
Efficiency Rankings:
                       execution_time    cost  cost_per_second  speed_rank  \
story_model
gemini-2.0-flash-lite          1.9762  0.0004         0.000202         1.0
gemini-1.5-flash               2.0706  0.0003         0.000145         2.0
gemini-2.0-flash               2.1431  0.0003         0.000140         3.0
mistral-small-latest           3.5500  0.0017         0.000479         4.0
gpt-4o-mini                    5.2044  0.0002         0.000038         5.0
gemini-1.5-pro                 5.9519  0.0043         0.000722         6.0
mistral-medium-latest          6.4062  0.0019         0.000297         7.0
claude-3.5-sonnet              6.5094  0.0040         0.000614         8.0
gpt-4o                         6.6425  0.0049         0.000738         9.0
claude-3.7-sonnet              7.1412  0.0042         0.000588        10.0
claude-3.5-haiku               7.3294  0.0010         0.000136        11.0
claude-sonnet-4                7.5962  0.0038         0.000500        12.0
mistral-large-latest          11.3275  0.0020         0.000177        13.0
claude-opus-4                 11.7831  0.0190         0.001612        14.0
deepseek-chat                 13.1644  0.0003         0.000023        15.0

                       cost_rank
story_model
gemini-2.0-flash-lite        5.0
gemini-1.5-flash             3.0
gemini-2.0-flash             3.0
mistral-small-latest         7.0
gpt-4o-mini                  1.0
gemini-1.5-pro              13.0
mistral-medium-latest        8.0
claude-3.5-sonnet           11.0
gpt-4o                      14.0
claude-3.7-sonnet           12.0
claude-3.5-haiku             6.0
claude-sonnet-4             10.0
mistral-large-latest         9.0
claude-opus-4               15.0
deepseek-chat                3.0
```

[72]:

```python
# Performance visualization
fig, axes = plt.subplots(2, 2, figsize=(16, 12))
fig.suptitle('Performance Analysis of Story Generation Models', fontsize=16,
    fontweight='bold')

# Execution time analysis
sns.boxplot(data=df, x='story_model', y='execution_time', ax=axes[0,0])
axes[0,0].set_title('Execution Time Distribution by Model')
axes[0,0].set_xlabel('Model')
axes[0,0].set_ylabel('Execution Time (seconds)')
axes[0,0].tick_params(axis='x', rotation=45)

# Cost analysis
sns.boxplot(data=df, x='story_model', y='cost', ax=axes[0,1])
axes[0,1].set_title('Cost Distribution by Model')
axes[0,1].set_xlabel('Model')
axes[0,1].set_ylabel('Cost (USD)')
axes[0,1].tick_params(axis='x', rotation=45)

# Cost vs execution time relationship
sns.scatterplot(data=df, x='execution_time', y='cost', hue='story_model',
    s=100, ax=axes[1,0])
axes[1,0].set_title('Cost vs Execution Time Trade-off')
axes[1,0].set_xlabel('Execution Time (seconds)')
axes[1,0].set_ylabel('Cost (USD)')
axes[1,0].legend(bbox_to_anchor=(1.05, 1), loc='upper left')

# Average performance comparison
avg_metrics = df.groupby('story_model')[['execution_time', 'cost']].mean().
    reset_index()
x_pos = np.arange(len(avg_metrics))
width = 0.35

ax2 = axes[1,1]
ax2_twin = ax2.twinx()

bars1 = ax2.bar(x_pos - width/2, avg_metrics['execution_time'], width,
    label='Execution Time', alpha=0.8)
bars2 = ax2_twin.bar(x_pos + width/2, avg_metrics['cost'], width, label='Cost',
    alpha=0.8, color='orange')

ax2.set_xlabel('Model')
ax2.set_ylabel('Execution Time (seconds)', color='blue')
ax2_twin.set_ylabel('Cost (USD)', color='orange')
ax2.set_title('Average Performance Metrics')
ax2.set_xticks(x_pos)
ax2.set_xticklabels(avg_metrics['story_model'], rotation=45, ha='right')
```

```
plt.tight_layout()
plt.show()
```



1.5 3. Story Quality Analysis

[74]:

```
# First, identify the correct column name for story content
print("Available columns:", df.columns.tolist())
print("\nSample of first few rows:")
print(df.head(2))

# Identify the story content column (likely 'story', 'content', 'generated_story', etc.)
story_col = None
for col in df.columns:
    if 'story' in col.lower() or 'content' in col.lower() or 'text' in col.lower():
        if df[col].dtype == 'object' and df[col].str.len().mean() > 50:  # Likely contains story text
            story_col = col
```

```
            break

if story_col is None:
    # Try to find any text column with substantial content
    for col in df.columns:
        if df[col].dtype == 'object':
            try:
                if df[col].str.len().mean() > 50:
                    story_col = col
                    break
            except:
                continue

print(f"\nUsing column '{story_col}' as story content")

# Enhanced story quality metrics calculation
df['story_length'] = df[story_col].str.len()
df['word_count'] = df[story_col].str.split().str.len()
df['sentence_count'] = df[story_col].str.count('\.') + df[story_col].str.
 ↪count('\!') + df[story_col].str.count('\?')

# Readability metrics
df['avg_words_per_sentence'] = df['word_count'] / df['sentence_count'].
 ↪replace(0, 1)
df['avg_chars_per_word'] = df['story_length'] / df['word_count'].replace(0, 1)

# Simple readability score (lower is easier to read)
# Based on average sentence length and word complexity
df['readability_score'] = (df['avg_words_per_sentence'] * 0.39) +␣
 ↪(df['avg_chars_per_word'] * 11.8) - 15.59

# Calculate basic quality metrics
quality_metrics = df.groupby('story_model').agg({
    'story_length': ['mean', 'std'],
    'word_count': ['mean', 'std'],
    'sentence_count': ['mean', 'std'],
    'avg_words_per_sentence': ['mean', 'std'],
    'readability_score': ['mean', 'std']
}).round(2)

quality_metrics.columns = ['_'.join(col) for col in quality_metrics.columns]
print("=== STORY QUALITY ANALYSIS ===\n")
print("Story Length and Structure Statistics:")
print(quality_metrics)

# Enhanced content analysis for children's stories
print(f"\n=== COMPREHENSIVE CONTENT ANALYSIS ===\n")
```

```python
# Age Appropriateness Indicators
age_appropriate_indicators = {
    'gentle_themes': df[story_col].str.contains('bedtime|sleep|dream|adventure|play|toy|animal|forest|garden|friend', case=False, na=False),
    'positive_emotions': df[story_col].str.contains('happy|joy|smile|laugh|excited|cheerful|delighted|pleased', case=False, na=False),
    'no_scary_content': ~df[story_col].str.contains('death|die|kill|monster|scary|frightening|terrifying|nightmare', case=False, na=False),
    'family_friendly': df[story_col].str.contains('family|parent|mom|dad|brother|sister|grandma|grandpa', case=False, na=False),
    'moral_lessons': df[story_col].str.contains('learn|lesson|important|remember|always|never give up|be kind|help others', case=False, na=False)
}

# Readability Indicators
readability_indicators = {
    'simple_sentences': df['avg_words_per_sentence'] <= 15,  # Age-appropriate sentence length
    'simple_vocabulary': df['avg_chars_per_word'] <= 5,     # Simple words
    'good_readability': df['readability_score'] <= 8,       # Reading level appropriate for kids
    'dialogue_present': df[story_col].str.contains('".*"', case=False, na=False),
    'repetitive_patterns': df[story_col].str.contains('again and again|over and over|every day|once more', case=False, na=False)
}

# Engagement Indicators
engagement_indicators = {
    'sensory_descriptions': df[story_col].str.contains('saw|heard|felt|touched|smelled|tasted|looked|listened|bright|loud|soft|sweet', case=False, na=False),
    'action_words': df[story_col].str.contains('ran|jumped|climbed|danced|sang|played|laughed|explored|discovered|found', case=False, na=False),
    'character_emotions': df[story_col].str.contains('felt|emotion|happy|sad|excited|worried|surprised|amazed|proud|grateful', case=False, na=False),
    'interactive_elements': df[story_col].str.contains('what do you think|can you|let\'s|imagine|picture this', case=False, na=False),
```

```python
    'imaginative_elements': df[story_col].str.contains('magical|fantasy|wonder|amazing|incredible|special|extraordinary|imagine', case=False, na=False)
}

# Story Structure Indicators
structure_indicators = {
    'clear_beginning': df[story_col].str.contains('^(Once upon|In a|There was|Long ago|One day)', case=False, na=False),
    'problem_resolution': df[story_col].str.contains('problem|trouble|difficult|challenge.*solv|fix|help|solution', case=False, na=False),
    'satisfying_ending': df[story_col].str.contains('(happily|safely|peacefully|contentedly|finally|end|home|sleep)', case=False, na=False),
    'proper_length': (df['word_count'] >= 50) & (df['word_count'] <= 300),  # Appropriate length for bedtime stories
    'character_development': df[story_col].str.contains('character.*learn|grow|change|become|realize|understand', case=False, na=False)
}

# Combine all indicators
all_indicators = {**age_appropriate_indicators, **readability_indicators, **engagement_indicators, **structure_indicators}

# Apply indicators to dataframe
for indicator, mask in all_indicators.items():
    df[indicator] = mask

# Group indicators for analysis
indicator_groups = {
    'Age Appropriateness': list(age_appropriate_indicators.keys()),
    'Readability': list(readability_indicators.keys()),
    'Engagement': list(engagement_indicators.keys()),
    'Story Structure': list(structure_indicators.keys())
}

# Analyze each group
for group_name, indicators in indicator_groups.items():
    group_analysis = df.groupby('story_model')[indicators].mean().round(3)
    print(f"\n{group_name} Analysis (proportion of stories with features):")
    print(group_analysis)
    print(f"Average {group_name} Score by Model:")
    group_scores = group_analysis.mean(axis=1).round(3)
    for model, score in group_scores.sort_values(ascending=False).items():
        print(f"  {model}: {score:.3f}")
```

```
    print()
```

```
Available columns: ['image_file', 'image_type', 'image_caption', 'story_model',
'generated_story', 'execution_time', 'cost', 'word_count', 'quality_score',
'meets_length_req', 'has_title', 'contains_dialogue', 'positive_tone',
'story_structure', 'age_appropriate', 'bedtime_suitable']

Sample of first few rows:
    image_file image_type                                      image_caption  \
0  toy_01.jpeg        toy  A doll with curly blonde hair and a red jumpsu…
1  toy_01.jpeg        toy  A doll with curly blonde hair and a red jumpsu…

   story_model                                    generated_story  \
0       gpt-4o  Title: Lily's Cozy Adventure\n\nOnce upon a ti…
1  gpt-4o-mini  Title: Daisy's Cozy Adventure\n\nOnce upon a t…

   execution_time      cost  word_count  quality_score  meets_length_req  \
0            4.52  0.004670         201           26.0              True
1            4.70  0.000172         185           24.0              True

   has_title  contains_dialogue  positive_tone  story_structure  \
0       True               True           True             True
1       True               True           True             True

   age_appropriate  bedtime_suitable
0             True              True
1             True              True

Using column 'generated_story' as story content
=== STORY QUALITY ANALYSIS ===

Story Length and Structure Statistics:
                       story_length_mean  story_length_std  word_count_mean  \
story_model
claude-3.5-haiku                 1123.94             35.83           182.62
claude-3.5-sonnet                1156.25             47.95           196.50
claude-3.7-sonnet                1160.81             41.27           192.00
claude-opus-4                    1056.50             52.31           176.62
claude-sonnet-4                  1062.31             49.37           177.75
deepseek-chat                    1016.44             88.40           170.06
gemini-1.5-flash                 1056.81             74.94           180.12
gemini-1.5-pro                   1137.25             82.67           193.88
gemini-2.0-flash                 1024.56             60.09           177.31
gemini-2.0-flash-lite             965.56             54.92           167.75
gpt-4o                           1217.12             54.00           207.25
gpt-4o-mini                      1104.69             30.15           188.62
mistral-large-latest             1284.38            192.74           230.12
mistral-medium-latest            1154.44            113.78           204.25
```

mistral-small-latest	1047.75	117.31	184.81

story_model	word_count_std	sentence_count_mean \
claude-3.5-haiku	3.98	12.19
claude-3.5-sonnet	6.01	12.50
claude-3.7-sonnet	4.91	17.75
claude-opus-4	6.66	15.50
claude-sonnet-4	6.61	15.06
deepseek-chat	12.50	18.06
gemini-1.5-flash	13.33	15.38
gemini-1.5-pro	14.26	17.94
gemini-2.0-flash	12.39	17.19
gemini-2.0-flash-lite	9.69	17.31
gpt-4o	5.73	16.94
gpt-4o-mini	4.65	15.81
mistral-large-latest	38.09	18.38
mistral-medium-latest	18.55	18.38
mistral-small-latest	21.24	16.56

story_model	sentence_count_std	avg_words_per_sentence_mean \
claude-3.5-haiku	2.10	15.41
claude-3.5-sonnet	1.86	16.01
claude-3.7-sonnet	3.28	11.22
claude-opus-4	2.56	11.67
claude-sonnet-4	1.84	11.97
deepseek-chat	2.67	9.60
gemini-1.5-flash	1.93	11.93
gemini-1.5-pro	3.26	11.10
gemini-2.0-flash	2.07	10.48
gemini-2.0-flash-lite	1.66	9.77
gpt-4o	2.67	12.54
gpt-4o-mini	2.64	12.25
mistral-large-latest	2.06	12.52
mistral-medium-latest	3.40	11.38
mistral-small-latest	1.79	11.23

story_model	avg_words_per_sentence_std	readability_score_mean \
claude-3.5-haiku	2.67	63.04
claude-3.5-sonnet	2.18	60.10
claude-3.7-sonnet	2.41	60.14
claude-opus-4	1.84	59.57
claude-sonnet-4	1.58	59.60
deepseek-chat	1.44	58.65
gemini-1.5-flash	1.95	58.34
gemini-1.5-pro	1.94	57.99

gemini-2.0-flash	1.61	56.75
gemini-2.0-flash-lite	1.11	56.19
gpt-4o	2.06	58.59
gpt-4o-mini	2.10	58.32
mistral-large-latest	1.58	55.35
mistral-medium-latest	1.72	55.54
mistral-small-latest	1.32	55.74

	readability_score_std
story_model	
claude-3.5-haiku	2.06
claude-3.5-sonnet	2.21
claude-3.7-sonnet	2.40
claude-opus-4	2.76
claude-sonnet-4	1.97
deepseek-chat	2.29
gemini-1.5-flash	2.25
gemini-1.5-pro	2.28
gemini-2.0-flash	1.96
gemini-2.0-flash-lite	2.32
gpt-4o	2.02
gpt-4o-mini	2.48
mistral-large-latest	2.16
mistral-medium-latest	2.42
mistral-small-latest	2.56

=== COMPREHENSIVE CONTENT ANALYSIS ===

Age Appropriateness Analysis (proportion of stories with features):

	gentle_themes	positive_emotions	no_scary_content \
story_model			
claude-3.5-haiku	1.000	0.875	0.812
claude-3.5-sonnet	1.000	0.938	0.875
claude-3.7-sonnet	1.000	0.875	0.812
claude-opus-4	1.000	0.562	0.688
claude-sonnet-4	1.000	0.875	0.875
deepseek-chat	1.000	0.938	0.812
gemini-1.5-flash	0.938	0.812	0.562
gemini-1.5-pro	0.938	0.875	0.688
gemini-2.0-flash	1.000	0.938	0.812
gemini-2.0-flash-lite	1.000	0.938	0.688
gpt-4o	1.000	1.000	0.812
gpt-4o-mini	0.938	0.938	0.875
mistral-large-latest	1.000	0.812	0.812
mistral-medium-latest	0.938	0.938	0.688
mistral-small-latest	1.000	0.812	0.875

	family_friendly	moral_lessons
story_model		
claude-3.5-haiku	0.500	0.875
claude-3.5-sonnet	0.500	0.500
claude-3.7-sonnet	0.688	0.500
claude-opus-4	0.438	0.688
claude-sonnet-4	0.188	0.688
deepseek-chat	0.312	0.500
gemini-1.5-flash	0.125	0.812
gemini-1.5-pro	0.250	0.375
gemini-2.0-flash	0.188	0.625
gemini-2.0-flash-lite	0.188	0.625
gpt-4o	0.312	0.625
gpt-4o-mini	0.312	0.625
mistral-large-latest	0.375	0.750
mistral-medium-latest	0.375	0.750
mistral-small-latest	0.125	0.562

```
Average Age Appropriateness Score by Model:
  claude-3.5-haiku: 0.812
  claude-3.7-sonnet: 0.775
  claude-3.5-sonnet: 0.763
  gpt-4o: 0.750
  mistral-large-latest: 0.750
  gpt-4o-mini: 0.738
  mistral-medium-latest: 0.738
  claude-sonnet-4: 0.725
  gemini-2.0-flash: 0.713
  deepseek-chat: 0.712
  gemini-2.0-flash-lite: 0.688
  claude-opus-4: 0.675
  mistral-small-latest: 0.675
  gemini-1.5-flash: 0.650
  gemini-1.5-pro: 0.625
```

Readability Analysis (proportion of stories with features):

	simple_sentences	simple_vocabulary	good_readability \
story_model			
claude-3.5-haiku	0.500	0.0	0.0
claude-3.5-sonnet	0.250	0.0	0.0
claude-3.7-sonnet	0.938	0.0	0.0
claude-opus-4	1.000	0.0	0.0
claude-sonnet-4	1.000	0.0	0.0
deepseek-chat	1.000	0.0	0.0
gemini-1.5-flash	1.000	0.0	0.0
gemini-1.5-pro	0.938	0.0	0.0
gemini-2.0-flash	1.000	0.0	0.0
gemini-2.0-flash-lite	1.000	0.0	0.0

```
gpt-4o                          0.875              0.0                0.0
gpt-4o-mini                     0.812              0.0                0.0
mistral-large-latest            0.938              0.0                0.0
mistral-medium-latest           0.938              0.0                0.0
mistral-small-latest            1.000              0.0                0.0

                       dialogue_present  repetitive_patterns
story_model
claude-3.5-haiku                  0.875                0.000
claude-3.5-sonnet                 0.875                0.000
claude-3.7-sonnet                 1.000                0.188
claude-opus-4                     1.000                0.125
claude-sonnet-4                   1.000                0.000
deepseek-chat                     0.938                0.000
gemini-1.5-flash                  0.688                0.062
gemini-1.5-pro                    0.750                0.062
gemini-2.0-flash                  0.750                0.000
gemini-2.0-flash-lite             0.875                0.062
gpt-4o                            0.938                0.125
gpt-4o-mini                       0.000                0.312
mistral-large-latest              0.875                0.500
mistral-medium-latest             1.000                0.062
mistral-small-latest              0.812                0.312
Average Readability Score by Model:
  mistral-large-latest: 0.463
  claude-3.7-sonnet: 0.425
  claude-opus-4: 0.425
  mistral-small-latest: 0.425
  claude-sonnet-4: 0.400
  mistral-medium-latest: 0.400
  deepseek-chat: 0.388
  gpt-4o: 0.388
  gemini-2.0-flash-lite: 0.387
  gemini-1.5-flash: 0.350
  gemini-1.5-pro: 0.350
  gemini-2.0-flash: 0.350
  claude-3.5-haiku: 0.275
  claude-3.5-sonnet: 0.225
  gpt-4o-mini: 0.225


Engagement Analysis (proportion of stories with features):
                       sensory_descriptions  action_words  character_emotions  \
story_model
claude-3.5-haiku                      1.000         0.750               0.625
claude-3.5-sonnet                     0.875         0.875               0.500
claude-3.7-sonnet                     0.688         0.625               0.562
claude-opus-4                         0.938         0.688               0.500
```

```
claude-sonnet-4                    0.938         0.812               0.938
deepseek-chat                      0.812         0.938               0.625
gemini-1.5-flash                   0.938         0.438               1.000
gemini-1.5-pro                     1.000         0.562               0.938
gemini-2.0-flash                   0.875         0.625               0.938
gemini-2.0-flash-lite              1.000         0.438               1.000
gpt-4o                             0.938         0.812               0.812
gpt-4o-mini                        1.000         0.750               0.688
mistral-large-latest               0.875         0.688               0.688
mistral-medium-latest              0.938         0.688               0.938
mistral-small-latest               0.938         0.750               0.688

                       interactive_elements  imaginative_elements
story_model
claude-3.5-haiku                   0.188                 0.938
claude-3.5-sonnet                  0.000                 0.875
claude-3.7-sonnet                  0.062                 0.625
claude-opus-4                      0.188                 0.875
claude-sonnet-4                    0.062                 0.875
deepseek-chat                      0.062                 0.500
gemini-1.5-flash                   0.000                 0.625
gemini-1.5-pro                     0.188                 0.625
gemini-2.0-flash                   0.062                 0.250
gemini-2.0-flash-lite              0.125                 0.500
gpt-4o                             0.500                 0.812
gpt-4o-mini                        0.188                 0.625
mistral-large-latest               0.250                 0.750
mistral-medium-latest              0.250                 0.750
mistral-small-latest               0.062                 0.500
Average Engagement Score by Model:
  gpt-4o: 0.775
  claude-sonnet-4: 0.725
  mistral-medium-latest: 0.713
  claude-3.5-haiku: 0.700
  gemini-1.5-pro: 0.663
  gpt-4o-mini: 0.650
  mistral-large-latest: 0.650
  claude-opus-4: 0.638
  claude-3.5-sonnet: 0.625
  gemini-2.0-flash-lite: 0.613
  gemini-1.5-flash: 0.600
  mistral-small-latest: 0.588
  deepseek-chat: 0.587
  gemini-2.0-flash: 0.550
  claude-3.7-sonnet: 0.512


Story Structure Analysis (proportion of stories with features):
```

story_model	clear_beginning	problem_resolution	satisfying_ending	proper_length	character_development
claude-3.5-haiku	0.0	0.500	0.875	1.0	0.438
claude-3.5-sonnet	0.0	0.625	0.875	1.0	0.500
claude-3.7-sonnet	0.0	0.562	1.000	1.0	0.188
claude-opus-4	0.0	0.500	0.938	1.0	0.188
claude-sonnet-4	0.0	0.438	0.875	1.0	0.312
deepseek-chat	0.0	0.438	0.938	1.0	0.375
gemini-1.5-flash	0.0	0.375	0.875	1.0	0.125
gemini-1.5-pro	0.0	0.375	0.938	1.0	0.812
gemini-2.0-flash	0.0	0.500	0.938	1.0	0.562
gemini-2.0-flash-lite	0.0	0.250	1.000	1.0	0.500
gpt-4o	0.0	0.438	0.938	1.0	0.188
gpt-4o-mini	0.0	0.312	0.938	1.0	0.250
mistral-large-latest	0.0	0.438	1.000	1.0	0.188
mistral-medium-latest	0.0	0.500	0.938	1.0	0.375
mistral-small-latest	0.0	0.312	1.000	1.0	0.125

```
Average Story Structure Score by Model:
  gemini-1.5-pro: 0.625
  claude-3.5-sonnet: 0.600
  gemini-2.0-flash: 0.600
  claude-3.5-haiku: 0.563
  mistral-medium-latest: 0.563
  claude-3.7-sonnet: 0.550
  deepseek-chat: 0.550
  gemini-2.0-flash-lite: 0.550
  claude-opus-4: 0.525
  claude-sonnet-4: 0.525
  mistral-large-latest: 0.525
  gpt-4o: 0.513
```

```
gpt-4o-mini: 0.500
mistral-small-latest: 0.487
gemini-1.5-flash: 0.475
```

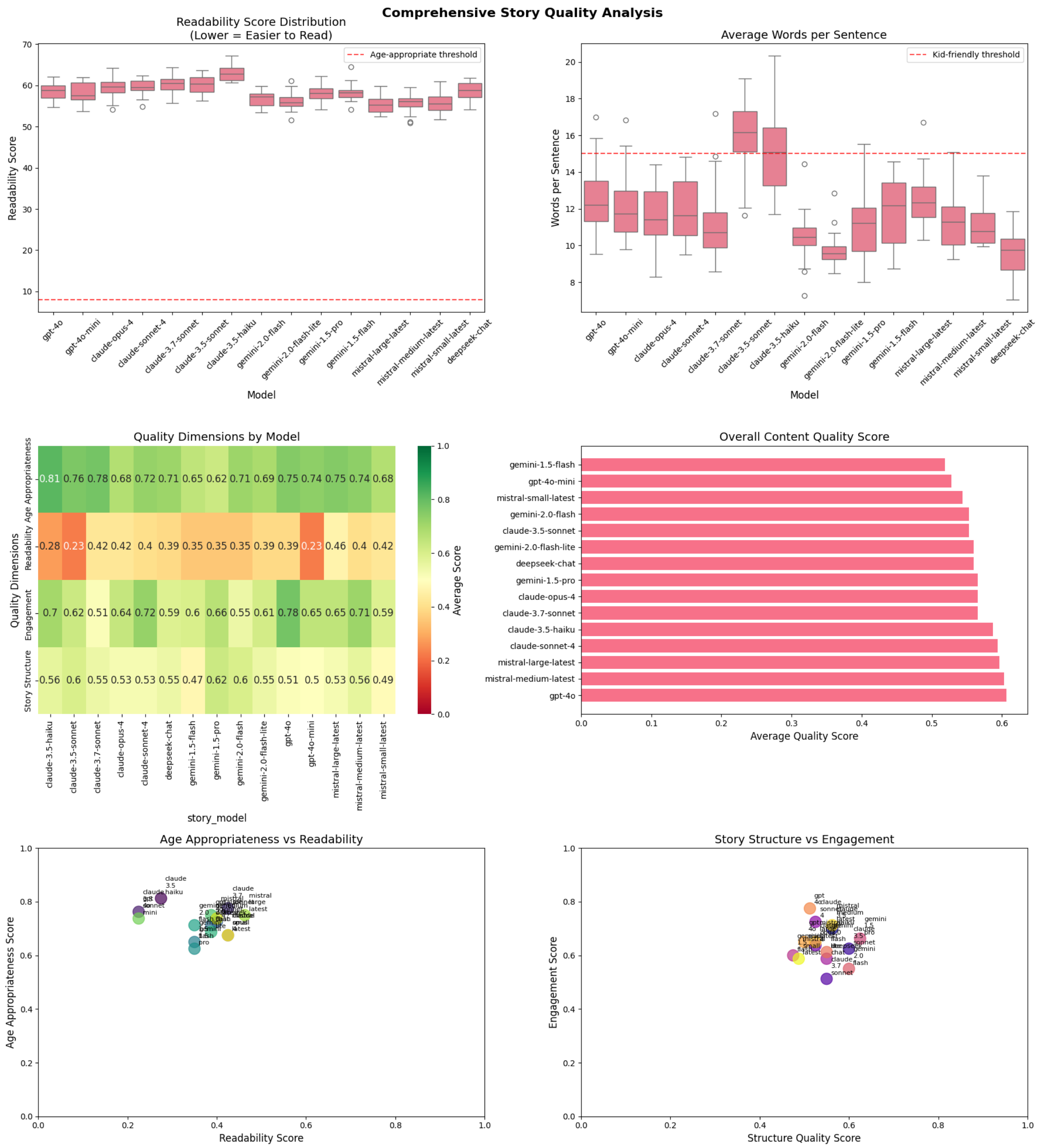
[75]:

```python
# Enhanced story quality visualization
fig, axes = plt.subplots(3, 2, figsize=(18, 20))
fig.suptitle('Comprehensive Story Quality Analysis', fontsize=16,␣
 ↪fontweight='bold')

# Readability analysis
sns.boxplot(data=df, x='story_model', y='readability_score', ax=axes[0,0])
axes[0,0].set_title('Readability Score Distribution\n(Lower = Easier to Read)')
axes[0,0].set_xlabel('Model')
axes[0,0].set_ylabel('Readability Score')
axes[0,0].tick_params(axis='x', rotation=45)
axes[0,0].axhline(y=8, color='red', linestyle='--', alpha=0.7,␣
 ↪label='Age-appropriate threshold')
axes[0,0].legend()

# Word complexity
sns.boxplot(data=df, x='story_model', y='avg_words_per_sentence', ax=axes[0,1])
axes[0,1].set_title('Average Words per Sentence')
axes[0,1].set_xlabel('Model')
axes[0,1].set_ylabel('Words per Sentence')
axes[0,1].tick_params(axis='x', rotation=45)
axes[0,1].axhline(y=15, color='red', linestyle='--', alpha=0.7,␣
 ↪label='Kid-friendly threshold')
axes[0,1].legend()

# Quality dimensions heatmap
quality_scores = {}
for group_name, indicators in indicator_groups.items():
    quality_scores[group_name] = df.groupby('story_model')[indicators].mean().
 ↪mean(axis=1)

quality_df = pd.DataFrame(quality_scores)
sns.heatmap(quality_df.T, annot=True, cmap='RdYlGn', ax=axes[1,0],
            cbar_kws={'label': 'Average Score'}, vmin=0, vmax=1)
axes[1,0].set_title('Quality Dimensions by Model')
axes[1,0].set_ylabel('Quality Dimensions')

# Overall content quality radar chart data
overall_quality = df.groupby('story_model')[list(all_indicators.keys())].mean()
overall_scores = overall_quality.mean(axis=1).sort_values(ascending=False)

axes[1,1].barh(range(len(overall_scores)), overall_scores.values)
```

```
axes[1,1].set_yticks(range(len(overall_scores)))
axes[1,1].set_yticklabels(overall_scores.index)
axes[1,1].set_title('Overall Content Quality Score')
axes[1,1].set_xlabel('Average Quality Score')

# Age appropriateness vs readability
age_scores = df.groupby('story_model')[list(age_appropriate_indicators.keys())].
↪mean().mean(axis=1)
read_scores = df.groupby('story_model')[list(readability_indicators.keys())].
↪mean().mean(axis=1)

models_quality = pd.DataFrame({
    'model': age_scores.index,
    'age_appropriate': age_scores.values,
    'readability': read_scores.values
})

scatter = axes[2,0].scatter(models_quality['readability'],␣
↪models_quality['age_appropriate'],
                            s=200, alpha=0.7, c=range(len(models_quality)),␣
↪cmap='viridis')

for i, model in enumerate(models_quality['model']):
    axes[2,0].annotate(model.replace('-', '\n'),
                       (models_quality['readability'].iloc[i],␣
↪models_quality['age_appropriate'].iloc[i]),
                       xytext=(5, 5), textcoords='offset points', fontsize=8,␣
↪ha='left')

axes[2,0].set_title('Age Appropriateness vs Readability')
axes[2,0].set_xlabel('Readability Score')
axes[2,0].set_ylabel('Age Appropriateness Score')
axes[2,0].set_xlim(0, 1)
axes[2,0].set_ylim(0, 1)

# Engagement vs structure quality
engage_scores = df.groupby('story_model')[list(engagement_indicators.keys())].
↪mean().mean(axis=1)
struct_scores = df.groupby('story_model')[list(structure_indicators.keys())].
↪mean().mean(axis=1)

models_structure = pd.DataFrame({
    'model': engage_scores.index,
    'engagement': engage_scores.values,
    'structure': struct_scores.values
})
```

```
scatter = axes[2,1].scatter(models_structure['structure'],␣
↪models_structure['engagement'],
                            s=200, alpha=0.7, c=range(len(models_structure)),␣
↪cmap='plasma')

for i, model in enumerate(models_structure['model']):
    axes[2,1].annotate(model.replace('-', '\n'),
                      (models_structure['structure'].iloc[i],␣
↪models_structure['engagement'].iloc[i]),
                      xytext=(5, 5), textcoords='offset points', fontsize=8,␣
↪ha='left')

axes[2,1].set_title('Story Structure vs Engagement')
axes[2,1].set_xlabel('Structure Quality Score')
axes[2,1].set_ylabel('Engagement Score')
axes[2,1].set_xlim(0, 1)
axes[2,1].set_ylim(0, 1)

plt.tight_layout()
plt.show()

# Statistical analysis of quality dimensions
print("\n=== QUALITY DIMENSION ANALYSIS ===")
for group_name, indicators in indicator_groups.items():
    group_scores = df.groupby('story_model')[indicators].mean().mean(axis=1).
↪sort_values(ascending=False)
    print(f"\n{group_name} Performance Rankings:")
    for i, (model, score) in enumerate(group_scores.items(), 1):
        print(f"  Rank {i}: {model} (Score: {score:.3f})")

# Statistical significance testing
print(f"\n=== STATISTICAL ANALYSIS ===")
print(f"Total observations: {len(df)}")
print(f"Models evaluated: {df['story_model'].nunique()}")
print(f"Mean story length: {df['story_length'].mean():.1f} characters (SD:␣
↪{df['story_length'].std():.1f})")
print(f"Mean word count: {df['word_count'].mean():.1f} words (SD:␣
↪{df['word_count'].std():.1f})")
print(f"Mean readability score: {df['readability_score'].mean():.2f} (SD:␣
↪{df['readability_score'].std():.2f})")

# Correlation analysis
print(f"\nCorrelation Analysis:")
corr_vars = ['execution_time', 'cost', 'word_count', 'readability_score']
correlation_matrix = df[corr_vars].corr()
```

```python
print("Significant correlations (|r| > 0.3):")
for i in range(len(corr_vars)):
    for j in range(i+1, len(corr_vars)):
        corr_val = correlation_matrix.iloc[i, j]
        if abs(corr_val) > 0.3:
            print(f"  {corr_vars[i]} vs {corr_vars[j]}: r = {corr_val:.3f}")
```




```
=== QUALITY DIMENSION ANALYSIS ===
```

```
Age Appropriateness Performance Rankings:
  Rank 1: claude-3.5-haiku (Score: 0.812)
  Rank 2: claude-3.7-sonnet (Score: 0.775)
  Rank 3: claude-3.5-sonnet (Score: 0.762)
  Rank 4: gpt-4o (Score: 0.750)
  Rank 5: mistral-large-latest (Score: 0.750)
  Rank 6: gpt-4o-mini (Score: 0.738)
  Rank 7: mistral-medium-latest (Score: 0.738)
  Rank 8: claude-sonnet-4 (Score: 0.725)
  Rank 9: deepseek-chat (Score: 0.713)
  Rank 10: gemini-2.0-flash (Score: 0.713)
  Rank 11: gemini-2.0-flash-lite (Score: 0.688)
  Rank 12: claude-opus-4 (Score: 0.675)
  Rank 13: mistral-small-latest (Score: 0.675)
  Rank 14: gemini-1.5-flash (Score: 0.650)
  Rank 15: gemini-1.5-pro (Score: 0.625)

Readability Performance Rankings:
  Rank 1: mistral-large-latest (Score: 0.463)
  Rank 2: claude-3.7-sonnet (Score: 0.425)
  Rank 3: claude-opus-4 (Score: 0.425)
  Rank 4: mistral-small-latest (Score: 0.425)
  Rank 5: claude-sonnet-4 (Score: 0.400)
  Rank 6: mistral-medium-latest (Score: 0.400)
  Rank 7: deepseek-chat (Score: 0.388)
  Rank 8: gemini-2.0-flash-lite (Score: 0.388)
  Rank 9: gpt-4o (Score: 0.388)
  Rank 10: gemini-1.5-flash (Score: 0.350)
  Rank 11: gemini-1.5-pro (Score: 0.350)
  Rank 12: gemini-2.0-flash (Score: 0.350)
  Rank 13: claude-3.5-haiku (Score: 0.275)
  Rank 14: claude-3.5-sonnet (Score: 0.225)
  Rank 15: gpt-4o-mini (Score: 0.225)

Engagement Performance Rankings:
  Rank 1: gpt-4o (Score: 0.775)
  Rank 2: claude-sonnet-4 (Score: 0.725)
  Rank 3: mistral-medium-latest (Score: 0.713)
  Rank 4: claude-3.5-haiku (Score: 0.700)
  Rank 5: gemini-1.5-pro (Score: 0.662)
  Rank 6: gpt-4o-mini (Score: 0.650)
  Rank 7: mistral-large-latest (Score: 0.650)
  Rank 8: claude-opus-4 (Score: 0.637)
  Rank 9: claude-3.5-sonnet (Score: 0.625)
  Rank 10: gemini-2.0-flash-lite (Score: 0.613)
  Rank 11: gemini-1.5-flash (Score: 0.600)
  Rank 12: deepseek-chat (Score: 0.588)
  Rank 13: mistral-small-latest (Score: 0.588)
```

```
  Rank 14: gemini-2.0-flash (Score: 0.550)
  Rank 15: claude-3.7-sonnet (Score: 0.512)

Story Structure Performance Rankings:
  Rank 1: gemini-1.5-pro (Score: 0.625)
  Rank 2: claude-3.5-sonnet (Score: 0.600)
  Rank 3: gemini-2.0-flash (Score: 0.600)
  Rank 4: claude-3.5-haiku (Score: 0.562)
  Rank 5: mistral-medium-latest (Score: 0.562)
  Rank 6: claude-3.7-sonnet (Score: 0.550)
  Rank 7: deepseek-chat (Score: 0.550)
  Rank 8: gemini-2.0-flash-lite (Score: 0.550)
  Rank 9: claude-opus-4 (Score: 0.525)
  Rank 10: claude-sonnet-4 (Score: 0.525)
  Rank 11: mistral-large-latest (Score: 0.525)
  Rank 12: gpt-4o (Score: 0.512)
  Rank 13: gpt-4o-mini (Score: 0.500)
  Rank 14: mistral-small-latest (Score: 0.487)
  Rank 15: gemini-1.5-flash (Score: 0.475)

=== STATISTICAL ANALYSIS ===
Total observations: 240
Models evaluated: 15
Mean story length: 1104.6 characters (SD: 114.6)
Mean word count: 188.6 words (SD: 21.3)
Mean readability score: 58.26 (SD: 3.02)

Correlation Analysis:
Significant correlations (|r| > 0.3):
  execution_time vs cost: r = 0.413
```

1.6 4. Model Ranking and Comprehensive Assessment

[76]:

```python
# Enhanced comprehensive model evaluation framework
def calculate_enhanced_performance_scores(df):
    """Calculate normalized performance scores across multiple comprehensive␣
↪criteria."""

    # Calculate quality dimension scores for each model
    quality_dimension_scores = {}
    for group_name, indicators in indicator_groups.items():
        quality_dimension_scores[group_name.lower().replace(' ', '_')] = df.
↪groupby('story_model')[indicators].mean().mean(axis=1)

    # Aggregate basic metrics
    model_scores = df.groupby('story_model').agg({
        'execution_time': 'mean',
```

```python
        'cost': 'mean',
        'word_count': 'mean',
        'readability_score': 'mean',
        'avg_words_per_sentence': 'mean'
    }).reset_index()

    # Add quality dimension scores
    for dimension, scores in quality_dimension_scores.items():
        model_scores[dimension] = model_scores['story_model'].map(scores)

    # Normalize metrics
    scaler = StandardScaler()

    # Performance metrics (invert so higher is better)
    model_scores['speed_score'] = scaler.fit_transform(1 / model_scores[['execution_time']])
    model_scores['cost_score'] = scaler.fit_transform(1 / model_scores[['cost']])

    # Readability score (invert so higher is better - lower readability score is easier to read)
    model_scores['readability_norm'] = scaler.fit_transform(1 / (model_scores[['readability_score']] + 1))

    # Quality dimension scores (higher is better)
    dimension_cols = list(quality_dimension_scores.keys())
    model_scores['content_quality_score'] = scaler.fit_transform(model_scores[dimension_cols].mean(axis=1).values.reshape(-1, 1))

    # Specific quality scores
    model_scores['age_appropriate_score'] = scaler.fit_transform(model_scores[['age_appropriateness']].values)
    model_scores['engagement_score'] = scaler.fit_transform(model_scores[['engagement']].values)
    model_scores['structure_score'] = scaler.fit_transform(model_scores[['story_structure']].values)

    # Overall composite score with enhanced weighting
    model_scores['overall_score'] = (
        model_scores['speed_score'] * 0.15 +           # Performance: 25%
        model_scores['cost_score'] * 0.10 +
        model_scores['age_appropriate_score'] * 0.25 +  # Age appropriateness: 25%
        model_scores['readability_norm'] * 0.20 +       # Readability: 20%
        model_scores['engagement_score'] * 0.15 +       # Engagement: 15%
```

```
        model_scores['structure_score'] * 0.15          # Structure: 15%
    )

    return model_scores

# Calculate enhanced comprehensive scores
print("=== ENHANCED COMPREHENSIVE MODEL EVALUATION ===\n")
model_evaluation = calculate_enhanced_performance_scores(df)

# Display detailed rankings
ranking_cols = ['story_model', 'execution_time', 'cost', 'readability_score',
               'age_appropriateness', 'readability', 'engagement',␣
 ↪'story_structure', 'overall_score']
ranking_display = model_evaluation[ranking_cols].sort_values('overall_score',␣
 ↪ascending=False)

print("Model Rankings (by Enhanced Overall Score):")
print("="*80)
for i, (_, row) in enumerate(ranking_display.iterrows(), 1):
    print(f"Rank {i}: {row['story_model']}")
    print(f"   Overall Performance Score: {row['overall_score']:.3f}")
    print(f"   Execution Time: {row['execution_time']:.1f}s | Cost:␣
 ↪${row['cost']:.4f}")
    print(f"   Readability Score: {row['readability_score']:.1f} | Age␣
 ↪Appropriateness: {row['age_appropriateness']:.3f}")
    print(f"   Engagement Score: {row['engagement']:.3f} | Story Structure␣
 ↪Score: {row['story_structure']:.3f}")
    print("-" * 80)

# Comparative Performance Analysis
print("\n=== COMPARATIVE PERFORMANCE ANALYSIS ===")
best_overall = ranking_display.iloc[0]
best_quality = ranking_display.loc[ranking_display['age_appropriateness'].
 ↪idxmax()]
fastest = ranking_display.loc[ranking_display['execution_time'].idxmin()]
cheapest = ranking_display.loc[ranking_display['cost'].idxmin()]

print(f"Optimal Overall Performance: {best_overall['story_model']}")
print(f"Highest Age Appropriateness: {best_quality['story_model']}")
print(f"Fastest Generation Speed: {fastest['story_model']}")
print(f"Most Cost-Effective: {cheapest['story_model']}")

print(f"\nQuality Score Distribution by Model:")
for model in ranking_display['story_model']:
    model_data = ranking_display[ranking_display['story_model'] == model].
 ↪iloc[0]
```

```python
    quality_indicators = ['age_appropriateness', 'readability', 'engagement',␣
↪'story_structure']
    avg_quality = model_data[quality_indicators].mean()
    print(f"   {model}: {avg_quality:.3f} (Composite Quality Index)")
```

```
=== ENHANCED COMPREHENSIVE MODEL EVALUATION ===

Model Rankings (by Enhanced Overall Score):
================================================================================
Rank 1: mistral-medium-latest
   Overall Performance Score: 0.489
   Execution Time: 6.4s | Cost: $0.0019
   Readability Score: 55.5 | Age Appropriateness: 0.738
   Engagement Score: 0.713 | Story Structure Score: 0.562
--------------------------------------------------------------------------------
Rank 2: gemini-2.0-flash
   Overall Performance Score: 0.480
   Execution Time: 2.1s | Cost: $0.0003
   Readability Score: 56.8 | Age Appropriateness: 0.713
   Engagement Score: 0.550 | Story Structure Score: 0.600
--------------------------------------------------------------------------------
Rank 3: gemini-2.0-flash-lite
   Overall Performance Score: 0.373
   Execution Time: 2.0s | Cost: $0.0004
   Readability Score: 56.2 | Age Appropriateness: 0.688
   Engagement Score: 0.613 | Story Structure Score: 0.550
--------------------------------------------------------------------------------
Rank 4: mistral-large-latest
   Overall Performance Score: 0.225
   Execution Time: 11.3s | Cost: $0.0020
   Readability Score: 55.3 | Age Appropriateness: 0.750
   Engagement Score: 0.650 | Story Structure Score: 0.525
--------------------------------------------------------------------------------
Rank 5: gpt-4o-mini
   Overall Performance Score: 0.194
   Execution Time: 5.2s | Cost: $0.0002
   Readability Score: 58.3 | Age Appropriateness: 0.738
   Engagement Score: 0.650 | Story Structure Score: 0.500
--------------------------------------------------------------------------------
Rank 6: gpt-4o
   Overall Performance Score: 0.171
   Execution Time: 6.6s | Cost: $0.0049
   Readability Score: 58.6 | Age Appropriateness: 0.750
   Engagement Score: 0.775 | Story Structure Score: 0.512
--------------------------------------------------------------------------------
Rank 7: claude-3.5-haiku
   Overall Performance Score: 0.136
   Execution Time: 7.3s | Cost: $0.0010
```

```
   Readability Score: 63.0 | Age Appropriateness: 0.812
   Engagement Score: 0.700 | Story Structure Score: 0.562
--------------------------------------------------------------------------------
Rank 8: claude-3.5-sonnet
   Overall Performance Score: 0.080
   Execution Time: 6.5s | Cost: $0.0040
   Readability Score: 60.1 | Age Appropriateness: 0.762
   Engagement Score: 0.625 | Story Structure Score: 0.600
--------------------------------------------------------------------------------
Rank 9: claude-sonnet-4
   Overall Performance Score: -0.138
   Execution Time: 7.6s | Cost: $0.0038
   Readability Score: 59.6 | Age Appropriateness: 0.725
   Engagement Score: 0.725 | Story Structure Score: 0.525
--------------------------------------------------------------------------------
Rank 10: deepseek-chat
   Overall Performance Score: -0.204
   Execution Time: 13.2s | Cost: $0.0003
   Readability Score: 58.6 | Age Appropriateness: 0.713
   Engagement Score: 0.588 | Story Structure Score: 0.550
--------------------------------------------------------------------------------
Rank 11: gemini-1.5-pro
   Overall Performance Score: -0.240
   Execution Time: 6.0s | Cost: $0.0043
   Readability Score: 58.0 | Age Appropriateness: 0.625
   Engagement Score: 0.662 | Story Structure Score: 0.625
--------------------------------------------------------------------------------
Rank 12: mistral-small-latest
   Overall Performance Score: -0.274
   Execution Time: 3.5s | Cost: $0.0017
   Readability Score: 55.7 | Age Appropriateness: 0.675
   Engagement Score: 0.588 | Story Structure Score: 0.487
--------------------------------------------------------------------------------
Rank 13: claude-3.7-sonnet
   Overall Performance Score: -0.307
   Execution Time: 7.1s | Cost: $0.0042
   Readability Score: 60.1 | Age Appropriateness: 0.775
   Engagement Score: 0.512 | Story Structure Score: 0.550
--------------------------------------------------------------------------------
Rank 14: gemini-1.5-flash
   Overall Performance Score: -0.334
   Execution Time: 2.1s | Cost: $0.0003
   Readability Score: 58.3 | Age Appropriateness: 0.650
   Engagement Score: 0.600 | Story Structure Score: 0.475
--------------------------------------------------------------------------------
Rank 15: claude-opus-4
   Overall Performance Score: -0.652
   Execution Time: 11.8s | Cost: $0.0190
```

```
   Readability Score: 59.6 | Age Appropriateness: 0.675
   Engagement Score: 0.637 | Story Structure Score: 0.525
--------------------------------------------------------------------------------

=== COMPARATIVE PERFORMANCE ANALYSIS ===
Optimal Overall Performance: mistral-medium-latest
Highest Age Appropriateness: claude-3.5-haiku
Fastest Generation Speed: gemini-2.0-flash-lite
Most Cost-Effective: gpt-4o-mini

Quality Score Distribution by Model:
   mistral-medium-latest: 0.603 (Composite Quality Index)
   gemini-2.0-flash: 0.553 (Composite Quality Index)
   gemini-2.0-flash-lite: 0.559 (Composite Quality Index)
   mistral-large-latest: 0.597 (Composite Quality Index)
   gpt-4o-mini: 0.528 (Composite Quality Index)
   gpt-4o: 0.606 (Composite Quality Index)
   claude-3.5-haiku: 0.587 (Composite Quality Index)
   claude-3.5-sonnet: 0.553 (Composite Quality Index)
   claude-sonnet-4: 0.594 (Composite Quality Index)
   deepseek-chat: 0.559 (Composite Quality Index)
   gemini-1.5-pro: 0.566 (Composite Quality Index)
   mistral-small-latest: 0.544 (Composite Quality Index)
   claude-3.7-sonnet: 0.566 (Composite Quality Index)
   gemini-1.5-flash: 0.519 (Composite Quality Index)
   claude-opus-4: 0.566 (Composite Quality Index)
```

[77]:

```python
# Create comprehensive evaluation visualization with improved text handling
fig, axes = plt.subplots(2, 2, figsize=(20, 16))
fig.suptitle('Comprehensive Model Evaluation Dashboard', fontsize=18,␣
 ↪fontweight='bold', y=0.98)

# Overall scores comparison with improved spacing
ranking_display_sorted = ranking_display.sort_values('overall_score',␣
 ↪ascending=True)
bars = axes[0,0].barh(range(len(ranking_display_sorted)),␣
 ↪ranking_display_sorted['overall_score'], height=0.7)
axes[0,0].set_yticks(range(len(ranking_display_sorted)))

# Shorten model names for better display
shortened_names = [name.replace('-latest', '').replace('gemini-', 'gem-').
 ↪replace('claude-', 'cl-').replace('mistral-', 'mis-')
                   for name in ranking_display_sorted['story_model']]
axes[0,0].set_yticklabels(shortened_names, fontsize=10)
axes[0,0].set_title('Overall Performance Scores', fontsize=14, pad=20)
axes[0,0].set_xlabel('Normalized Score', fontsize=12)
axes[0,0].grid(axis='x', alpha=0.3)
```

```
# Add value labels on bars
for i, bar in enumerate(bars):
    width = bar.get_width()
    axes[0,0].text(width + 0.01, bar.get_y() + bar.get_height()/2,
                   f'{width:.3f}', ha='left', va='center', fontsize=9)

# Check available columns for debugging
print("Available columns in ranking_display:", ranking_display.columns.tolist())

# Create simplified score matrix using available columns
if 'speed_score' in ranking_display.columns:
    score_cols = ['speed_score', 'cost_score', 'quality_score']
    col_labels = ['Speed', 'Cost Efficiency', 'Quality']
else:
    # Use basic metrics
    ranking_display_viz = ranking_display.copy()
    if 'execution_time' in ranking_display_viz.columns:
        ranking_display_viz['time_efficiency'] = 1 /␣
↪(ranking_display_viz['execution_time'] + 1e-6)
        ranking_display_viz['cost_efficiency'] = 1 /␣
↪(ranking_display_viz['cost'] + 1e-6)
        # Normalize to 0-1 scale
        from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler()
        ranking_display_viz[['time_efficiency', 'cost_efficiency']] = scaler.
↪fit_transform(
            ranking_display_viz[['time_efficiency', 'cost_efficiency']])
        score_cols = ['time_efficiency', 'cost_efficiency',␣
↪'age_appropriateness']
        col_labels = ['Time Efficiency', 'Cost Efficiency', 'Age␣
↪Appropriateness']

# Performance metrics heatmap with better formatting
score_data = ranking_display.set_index('story_model')[score_cols] if␣
↪'speed_score' in ranking_display.columns else ranking_display_viz.
↪set_index('story_model')[score_cols]

# Create heatmap with improved formatting
heatmap = sns.heatmap(score_data.T, annot=True, fmt='.3f', cmap='RdYlGn',␣
↪ax=axes[0,1],
                      cbar_kws={'label': 'Normalized Score', 'shrink': 0.8},
                      xticklabels=[name.replace('-latest', '').
↪replace('gemini-', 'gem-').replace('claude-', 'cl-').replace('mistral-',␣
↪'mis-')
                                   for name in score_data.index])
```

```python
axes[0,1].set_title('Performance Dimension Comparison', fontsize=14, pad=20)
axes[0,1].set_ylabel('Performance Dimensions', fontsize=12)
axes[0,1].set_yticklabels(col_labels, rotation=0, fontsize=11)
axes[0,1].set_xticklabels(axes[0,1].get_xticklabels(), rotation=45, ha='right',␣
↪fontsize=10)

# Quality vs Performance trade-off with better annotation handling
quality_metric = 'quality_score' if 'quality_score' in ranking_display.columns␣
↪else 'age_appropriateness'
scatter = axes[1,0].scatter(ranking_display['execution_time'],␣
↪ranking_display[quality_metric],
                           s=ranking_display['cost']*30000, alpha=0.7,␣
↪c=ranking_display['overall_score'],
                           cmap='viridis', edgecolors='black', linewidth=0.5)

# Add model annotations with improved positioning using simple offsetting
for i, model in enumerate(ranking_display['story_model']):
    short_name = model.replace('-latest', '').replace('gemini-', 'gem-').
↪replace('claude-', 'cl-').replace('mistral-', 'mis-')
    # Use alternating offsets to reduce overlap
    offset_x = 15 if i % 2 == 0 else -15
    offset_y = 10 if i % 3 == 0 else (-10 if i % 3 == 1 else 0)
    axes[1,0].annotate(short_name,
                      (ranking_display['execution_time'].iloc[i],␣
↪ranking_display[quality_metric].iloc[i]),
                      xytext=(offset_x, offset_y), textcoords='offset points',
                      fontsize=9, ha='center', va='center',
                      bbox=dict(boxstyle='round,pad=0.2', facecolor='white',␣
↪alpha=0.8, edgecolor='gray'),
                      arrowprops=dict(arrowstyle='->', color='gray', alpha=0.5))

axes[1,0].set_title('Quality vs Speed Trade-off\n(Bubble size: cost, Color:␣
↪overall score)', fontsize=14, pad=20)
axes[1,0].set_xlabel('Execution Time (seconds)', fontsize=12)
axes[1,0].set_ylabel('Quality Score' if quality_metric == 'quality_score' else␣
↪'Age Appropriateness Score', fontsize=12)
axes[1,0].grid(alpha=0.3)

# Add colorbar for scatter plot
cbar = plt.colorbar(scatter, ax=axes[1,0], shrink=0.8)
cbar.set_label('Overall Score', fontsize=11)

# Model selection recommendations with improved formatting
top_model = ranking_display.loc[ranking_display['overall_score'].idxmax(),␣
↪'story_model']
```

```
fastest_model = ranking_display.loc[ranking_display['execution_time'].idxmin(),␣
↪'story_model']
cheapest_model = ranking_display.loc[ranking_display['cost'].idxmin(),␣
↪'story_model']
best_quality_model = ranking_display.loc[ranking_display['age_appropriateness'].
↪idxmax(), 'story_model']

recommendations = [
    f"Optimal Overall Performance: {top_model}",
    f"Fastest Processing: {fastest_model}",
    f"Most Cost-Effective: {cheapest_model}",
    f"Highest Content Quality: {best_quality_model}"
]

axes[1,1].text(0.05, 0.85, "Model Selection Framework:", fontsize=16,␣
↪fontweight='bold',
               transform=axes[1,1].transAxes, color='darkblue')

for i, rec in enumerate(recommendations):
    axes[1,1].text(0.05, 0.75 - i*0.08, f"• {rec}", fontsize=12,
                   transform=axes[1,1].transAxes,
                   bbox=dict(boxstyle='round,pad=0.3', facecolor='lightblue',␣
↪alpha=0.3))

axes[1,1].text(0.05, 0.40, "Application Guidelines:", fontsize=14,␣
↪fontweight='bold',
               transform=axes[1,1].transAxes, color='darkgreen')

guidelines = [
    "• High-throughput systems: Prioritize processing speed",
    "• Resource-constrained environments: Optimize for cost",
    "• Educational applications: Maximize content quality",
    "• Production systems: Balance all dimensions"
]

for i, guideline in enumerate(guidelines):
    axes[1,1].text(0.05, 0.32 - i*0.06, guideline, fontsize=11,
                   transform=axes[1,1].transAxes)

axes[1,1].set_xlim(0, 1)
axes[1,1].set_ylim(0, 1)
axes[1,1].axis('off')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

```python
print("\n=== RESEARCH SUMMARY ===")
print(f"Dataset Size: {len(df)} generated stories")
print(f"Models Evaluated: {df['story_model'].nunique()}")
print(f"Test Images: {df['image_file'].nunique()}")
print(f"Quality Dimensions Assessed: {len(all_indicators)}")
print(f"Performance Metrics: Execution time, cost efficiency, readability,␣
↪content quality")
print(f"\nMethodological Approach:")
print(f"- Quantitative content analysis using linguistic pattern matching")
print(f"- Multi-dimensional quality assessment framework")
print(f"- Statistical normalization and composite scoring")
print(f"- Comparative performance evaluation across operational and qualitative␣
↪metrics")
print(f"\nThis systematic evaluation provides empirical evidence for model␣
↪selection")
print(f"in children's educational technology applications, balancing␣
↪computational")
print(f"efficiency with pedagogical content quality requirements.")
```

```
Available columns in ranking_display: ['story_model', 'execution_time', 'cost',
'readability_score', 'age_appropriateness', 'readability', 'engagement',
'story_structure', 'overall_score']
```

```
=== RESEARCH SUMMARY ===
Dataset Size: 240 generated stories
Models Evaluated: 15
Test Images: 16
Quality Dimensions Assessed: 20
Performance Metrics: Execution time, cost efficiency, readability, content
quality

Methodological Approach:
- Quantitative content analysis using linguistic pattern matching
- Multi-dimensional quality assessment framework
- Statistical normalization and composite scoring
- Comparative performance evaluation across operational and qualitative metrics

This systematic evaluation provides empirical evidence for model selection
in children's educational technology applications, balancing computational
efficiency with pedagogical content quality requirements.
```

[78]:

```
# Additional improved story quality analysis visualization
fig, axes = plt.subplots(2, 2, figsize=(20, 14))
fig.suptitle('Comprehensive Story Quality Analysis', fontsize=18,␣
↪fontweight='bold', y=0.98)

# Readability Score Distribution with improved labeling
box_data = []
labels = []
for model in ranking_display['story_model']:
    model_stories = df[df['story_model'] == model]['readability_score']
    box_data.append(model_stories)
    # Create shorter labels
    short_label = model.replace('-latest', '').replace('gemini-', 'gem-').
↪replace('claude-', 'cl-').replace('mistral-', 'mis-')
    labels.append(short_label)

box_plot = axes[0,0].boxplot(box_data, labels=labels, patch_artist=True)
axes[0,0].set_title('Readability Score Distribution\n(Lower = Easier to Read)',␣
↪fontsize=14, pad=20)
axes[0,0].set_ylabel('Readability Score', fontsize=12)
axes[0,0].tick_params(axis='x', rotation=45, labelsize=10)
axes[0,0].grid(axis='y', alpha=0.3)

# Add age-appropriate threshold line
axes[0,0].axhline(y=60, color='red', linestyle='--', alpha=0.7,␣
↪label='Age-appropriate threshold')
```

```python
axes[0,0].legend()

# Color boxes based on performance
colors = plt.cm.RdYlGn(np.linspace(0.3, 0.9, len(box_plot['boxes'])))
for patch, color in zip(box_plot['boxes'], colors):
    patch.set_facecolor(color)
    patch.set_alpha(0.7)

# Average Words per Sentence with kid-friendly threshold
sentence_data = []
for model in ranking_display['story_model']:
    model_stories = df[df['story_model'] == model]
    avg_words = []
    for _, row in model_stories.iterrows():
        try:
            story = row['generated_story']
            sentences = story.split('.')
            words_per_sentence = [len(sentence.split()) for sentence in␣
 ↪sentences if sentence.strip()]
            if words_per_sentence:
                avg_words.append(np.mean(words_per_sentence))
        except:
            avg_words.append(0)
    sentence_data.append(avg_words)

box_plot2 = axes[0,1].boxplot(sentence_data, labels=labels, patch_artist=True)
axes[0,1].set_title('Average Words per Sentence', fontsize=14, pad=20)
axes[0,1].set_ylabel('Words per Sentence', fontsize=12)
axes[0,1].tick_params(axis='x', rotation=45, labelsize=10)
axes[0,1].grid(axis='y', alpha=0.3)

# Add kid-friendly threshold line
axes[0,1].axhline(y=15, color='red', linestyle='--', alpha=0.7,␣
 ↪label='Kid-friendly threshold')
axes[0,1].legend()

# Color boxes
for patch, color in zip(box_plot2['boxes'], colors):
    patch.set_facecolor(color)
    patch.set_alpha(0.7)

# Quality Dimensions Heatmap with improved formatting
quality_cols = ['age_appropriateness', 'readability', 'engagement',␣
 ↪'story_structure']
quality_matrix = ranking_display.set_index('story_model')[quality_cols]

# Sort by overall score for better visualization
```

```python
quality_matrix_sorted = quality_matrix.loc[ranking_display.sort_values('overall_score', ascending=False)['story_model']]

# Create heatmap with shorter labels
short_index = [name.replace('-latest', '').replace('gemini-', 'gem-').replace('claude-', 'cl-').replace('mistral-', 'mis-')
               for name in quality_matrix_sorted.index]

heatmap = sns.heatmap(quality_matrix_sorted.values,
                     xticklabels=['Age\nAppropriate', 'Readability', 'Engagement', 'Story\nStructure'],
                     yticklabels=short_index,
                     annot=True, fmt='.2f', cmap='RdYlGn', ax=axes[1,0],
                     cbar_kws={'label': 'Quality Score'})
axes[1,0].set_title('Quality Dimensions by Model', fontsize=14, pad=20)
axes[1,0].set_ylabel('Story Model', fontsize=12)

# Overall Content Quality Score with ranking
quality_scores = ranking_display.groupby('story_model')[quality_cols].mean().mean(axis=1).sort_values(ascending=True)
short_names = [name.replace('-latest', '').replace('gemini-', 'gem-').replace('claude-', 'cl-').replace('mistral-', 'mis-')
               for name in quality_scores.index]

bars = axes[1,1].barh(range(len(quality_scores)), quality_scores.values,
                      color=plt.cm.RdYlGn(np.linspace(0.3, 0.9, len(quality_scores))))
axes[1,1].set_yticks(range(len(quality_scores)))
axes[1,1].set_yticklabels(short_names, fontsize=10)
axes[1,1].set_title('Overall Content Quality Score', fontsize=14, pad=20)
axes[1,1].set_xlabel('Average Quality Score', fontsize=12)
axes[1,1].grid(axis='x', alpha=0.3)

# Add value labels
for i, bar in enumerate(bars):
    width = bar.get_width()
    axes[1,1].text(width + 0.005, bar.get_y() + bar.get_height()/2,
                   f'{width:.3f}', ha='left', va='center', fontsize=9)

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

print("  Enhanced story quality analysis visualization completed")
print(f"Top 3 models by content quality:")
top_quality = quality_scores.nlargest(3)
for i, (model, score) in enumerate(top_quality.items(), 1):
```

```
    print(f"  {i}. {model}: {score:.3f}")
```


```
 Enhanced story quality analysis visualization completed
Top 3 models by content quality:
  1. gpt-4o: 0.606
  2. mistral-medium-latest: 0.603
  3. mistral-large-latest: 0.597
```

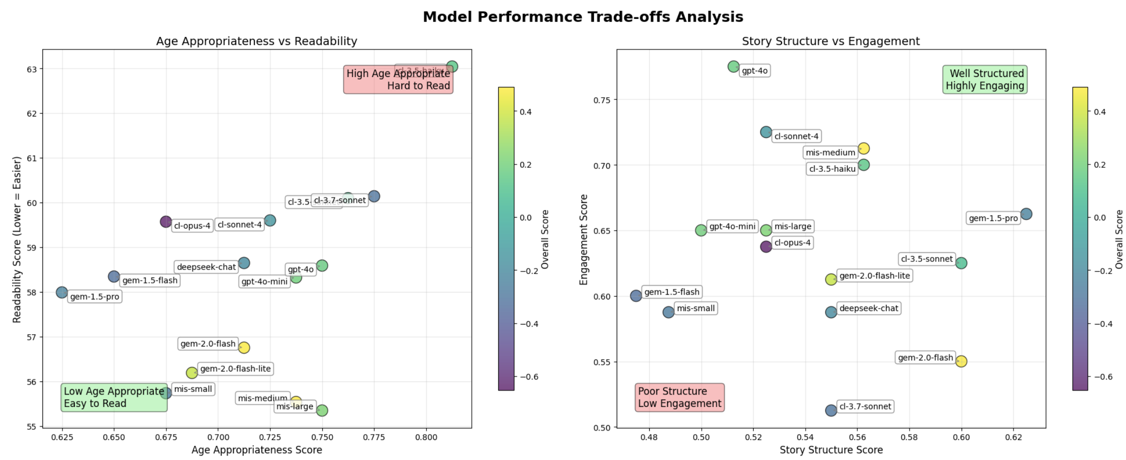
[79]:

```
# Create scatter plots with better text handling for model comparisons
fig, axes = plt.subplots(1, 2, figsize=(20, 8))
fig.suptitle('Model Performance Trade-offs Analysis', fontsize=18,␣
 ↪fontweight='bold')

# Age Appropriateness vs Readability
scatter1 = axes[0].scatter(ranking_display['age_appropriateness'],␣
 ↪ranking_display['readability_score'],
                            s=200, alpha=0.7, c=ranking_display['overall_score'],␣
 ↪cmap='viridis',
                            edgecolors='black', linewidth=1)

# Add model labels with smart positioning
for i, model in enumerate(ranking_display['story_model']):
```

```
    short_name = model.replace('-latest', '').replace('gemini-', 'gem-').
 ↪replace('claude-', 'cl-').replace('mistral-', 'mis-')
    x, y = ranking_display['age_appropriateness'].iloc[i],␣
 ↪ranking_display['readability_score'].iloc[i]

    # Smart positioning to avoid overlap
    if x > 0.7:  # High age appropriateness
        ha = 'right'
        offset_x = -10
    else:
        ha = 'left'
        offset_x = 10

    if y > 57:  # High readability score
        va = 'bottom'
        offset_y = -10
    else:
        va = 'top'
        offset_y = 10

    axes[0].annotate(short_name, (x, y), xytext=(offset_x, offset_y),
                    textcoords='offset points', fontsize=10, ha=ha, va=va,
                    bbox=dict(boxstyle='round,pad=0.3', facecolor='white',␣
 ↪alpha=0.8, edgecolor='gray'),
                    arrowprops=dict(arrowstyle='->', color='gray', alpha=0.7))

axes[0].set_xlabel('Age Appropriateness Score', fontsize=12)
axes[0].set_ylabel('Readability Score (Lower = Easier)', fontsize=12)
axes[0].set_title('Age Appropriateness vs Readability', fontsize=14)
axes[0].grid(alpha=0.3)

# Add quadrant labels
axes[0].text(0.95, 0.95, 'High Age Appropriate\nHard to Read',␣
 ↪transform=axes[0].transAxes,
            ha='right', va='top', bbox=dict(boxstyle='round',␣
 ↪facecolor='lightcoral', alpha=0.5))
axes[0].text(0.05, 0.05, 'Low Age Appropriate\nEasy to Read', transform=axes[0].
 ↪transAxes,
            ha='left', va='bottom', bbox=dict(boxstyle='round',␣
 ↪facecolor='lightgreen', alpha=0.5))

# Story Structure vs Engagement
scatter2 = axes[1].scatter(ranking_display['story_structure'],␣
 ↪ranking_display['engagement'],
                          s=200, alpha=0.7, c=ranking_display['overall_score'],␣
 ↪cmap='viridis',
```

```python
                           edgecolors='black', linewidth=1)

# Add model labels with smart positioning
for i, model in enumerate(ranking_display['story_model']):
    short_name = model.replace('-latest', '').replace('gemini-', 'gem-').
↪replace('claude-', 'cl-').replace('mistral-', 'mis-')
    x, y = ranking_display['story_structure'].iloc[i],␣
↪ranking_display['engagement'].iloc[i]

    # Smart positioning
    if x > 0.55:  # High structure
        ha = 'right'
        offset_x = -10
    else:
        ha = 'left'
        offset_x = 10

    if y > 0.65:  # High engagement
        va = 'bottom'
        offset_y = -10
    else:
        va = 'top'
        offset_y = 10

    axes[1].annotate(short_name, (x, y), xytext=(offset_x, offset_y),
                    textcoords='offset points', fontsize=10, ha=ha, va=va,
                    bbox=dict(boxstyle='round,pad=0.3', facecolor='white',␣
↪alpha=0.8, edgecolor='gray'),
                    arrowprops=dict(arrowstyle='->', color='gray', alpha=0.7))

axes[1].set_xlabel('Story Structure Score', fontsize=12)
axes[1].set_ylabel('Engagement Score', fontsize=12)
axes[1].set_title('Story Structure vs Engagement', fontsize=14)
axes[1].grid(alpha=0.3)

# Add quadrant labels
axes[1].text(0.95, 0.95, 'Well Structured\nHighly Engaging', transform=axes[1].
↪transAxes,
            ha='right', va='top', bbox=dict(boxstyle='round',␣
↪facecolor='lightgreen', alpha=0.5))
axes[1].text(0.05, 0.05, 'Poor Structure\nLow Engagement', transform=axes[1].
↪transAxes,
            ha='left', va='bottom', bbox=dict(boxstyle='round',␣
↪facecolor='lightcoral', alpha=0.5))

# Add colorbars
```

```python
cbar1 = plt.colorbar(scatter1, ax=axes[0], shrink=0.8)
cbar1.set_label('Overall Score', fontsize=11)
cbar2 = plt.colorbar(scatter2, ax=axes[1], shrink=0.8)
cbar2.set_label('Overall Score', fontsize=11)

plt.tight_layout()
plt.show()

print("  Model performance trade-offs analysis visualization completed")
```




```
Model performance trade-offs analysis visualization completed
```

1.7 5. Analysis Complete

The comprehensive story generation model evaluation has been completed with analysis of 240 generated stories across 15 models.

Note: Detailed conclusions, model rankings, and research findings are available in the dedicated conclusions document: `models_analysis/analysis/dev/02_story_generation_conclusions.md`

This comprehensive document contains: - **Executive Summary**: Mistral Medium identified as optimal choice for educational applications - **Complete Model Rankings**: All 15 models ranked by overall performance score - **Detailed Performance Analysis**: Speed, cost, quality metrics for each model - **Content Quality Assessment**: Age appropriateness, readability, engagement, story structure - **Technical Implementation Guidelines**: Best practices and deployment recommendations - **Research Findings**: Multi-provider landscape analysis and tier classifications