AI StoryTeller

October 5, 2025

[7]:
```python
print("=" * 70)
print("  INTERACTIVE AI STORYTELLER")
print("=" * 70)
print("  Project initialized successfully!")
print("  Author: Khyati")
print("  Date:", "October 2025")
print("=" * 70)
```

```
======================================================================
  INTERACTIVE AI STORYTELLER
======================================================================
  Project initialized successfully!
  Author: Khyati
  Date: October 2025
======================================================================
```

[8]:
```python
import sys
import platform

print("\n  SYSTEM INFORMATION")
print("-" * 70)
print(f"Python Version: {sys.version}")
print(f"Platform: {platform.system()} {platform.release()}")
print(f"Architecture: {platform.machine()}")
print("-" * 70)

# Check Python version
if sys.version_info < (3, 7):
    print("  WARNING: Python 3.7+ is recommended!")
else:
    print("  Python version is compatible!")
```

```

  SYSTEM INFORMATION
----------------------------------------------------------------------
Python Version: 3.12.3 | packaged by conda-forge | (main, Apr 15 2024, 18:20:11)
[MSC v.1938 64 bit (AMD64)]
Platform: Windows 11
Architecture: AMD64
```

```
----------------------------------------------------------------------
 Python version is compatible!
```

[9]:
```python
print("\n  INSTALLING REQUIRED LIBRARIES...")
print("  This may take 2-5 minutes on first run. Please wait...")
print("-" * 70)

import subprocess
import sys

def pip_install(package):
    subprocess.check_call([sys.executable, "-m", "pip", "install", package])

# Updated versions where needed
required_packages = {
    "transformers": "transformers==4.36.0",
    "torch": "torch",
    "gradio": "gradio==4.8.0",
    "accelerate": "accelerate==0.25.0",
    "sentencepiece": "sentencepiece"
}

for name, pkg in required_packages.items():
    print(f"  Installing {name}...")
    try:
        pip_install(pkg)
    except Exception as e:
        print(f"  Failed to install {name}: {e}")

print("\n  All libraries installed (or attempted) successfully!")
print("=" * 70)
```

```

 INSTALLING REQUIRED LIBRARIES…
 This may take 2-5 minutes on first run. Please wait…
----------------------------------------------------------------------
 Installing transformers…
 Installing torch…
 Installing gradio…
 Installing accelerate…
 Installing sentencepiece…

 All libraries installed (or attempted) successfully!
======================================================================
```

[10]:
```python
print("\n  IMPORTING LIBRARIES...")
print("-" * 70)
```

```python
# Core libraries
import torch
import transformers
from transformers import (
    AutoTokenizer,
    AutoModelForCausalLM,
    pipeline,
    set_seed
)
import gradio as gr
import warnings
import time
from datetime import datetime
import logging

# Suppress general warnings
warnings.filterwarnings('ignore')

# Suppress transformers-specific logging
logging.getLogger("transformers").setLevel(logging.ERROR)

print("  All libraries imported successfully!")

print("\n  LIBRARY VERSIONS:")
print(f"   • PyTorch: {torch.__version__}")
print(f"   • Transformers: {transformers.__version__}")
print(f"   • Gradio: {gr.__version__}")

print(f"\n   CUDA Status: {'  Available' if torch.cuda.is_available() else '␣
 ↪Not Available (Using CPU)'}")
if torch.cuda.is_available():
    print(f"   • GPU Device: {torch.cuda.get_device_name(0)}")
print("=" * 70)
```

```
  IMPORTING LIBRARIES…
----------------------------------------------------------------------
  All libraries imported successfully!

  LIBRARY VERSIONS:
   • PyTorch: 2.3.1
   • Transformers: 4.36.0
   • Gradio: 4.8.0

   CUDA Status:   Not Available (Using CPU)
======================================================================
[11]: class AIStoryTeller:
          """
          AI StoryTeller class that handles model loading, story generation,
          and interactive features.

          This class encapsulates all the functionality needed to:
          - Load pre-trained language models
          - Generate creative stories based on prompts
          - Provide statistics about generated content
          """

          def _init_(self, model_name="gpt2"):
              """
              Initialize the AI StoryTeller with a language model.

              Args:
                  model_name (str): Name of the model from Hugging Face
                                  Options: "gpt2", "gpt2-medium", "distilgpt2"
              """
              print(f"\n  INITIALIZING AI STORYTELLER")
              print("-" * 70)
              print(f"  Loading model: {model_name}")
              print("  First-time download may take 2-3 minutes...")

              start_time = time.time()

              # Load tokenizer (converts text to numbers and back)
              print("     Loading tokenizer...")
              self.tokenizer = AutoTokenizer.from_pretrained(model_name)

              # Load the actual language model
              print("     Loading model...")
              self.model = AutoModelForCausalLM.from_pretrained(
                  model_name,
                  torch_dtype=torch.float16 if torch.cuda.is_available() else torch.
      ↪float32,
                  device_map="auto" if torch.cuda.is_available() else None
              )

              # Set padding token if not already set
              if self.tokenizer.pad_token is None:
                  self.tokenizer.pad_token = self.tokenizer.eos_token

              # Create text generation pipeline for easier use
              print("     Creating generation pipeline...")
              self.generator = pipeline(
                  "text-generation",
                  model=self.model,
                  tokenizer=self.tokenizer,
                  device=0 if torch.cuda.is_available() else -1
              )

              end_time = time.time()

              print(f"\n  Model loaded successfully!")
              print(f"   Loading time: {end_time - start_time:.2f} seconds")
              print("=" * 70)

              # Genre-specific prompts to improve story quality
              self.genre_prompts = {
                  "Fantasy": "In a magical realm where dragons soar and wizards cast␣
      ↪ancient spells,",
                  "Sci-Fi": "In the year 2157, as humanity expanded across the solar␣
      ↪system,",
                  "Mystery": "The detective examined the crime scene carefully,␣
      ↪knowing something wasn't right.",
                  "Horror": "The old mansion stood abandoned for decades, its dark␣
      ↪windows hiding secrets.",
                  "Romance": "Their eyes met across the crowded room, and in that␣
      ↪magical moment,",
                  "Adventure": "The ancient treasure map was authentic, but the␣
      ↪journey would be treacherous.",
                  "Comedy": "Nobody at the office could have predicted what happened␣
      ↪next at the party.",
                  "Thriller": "The phone rang at exactly midnight. The caller's voice␣
      ↪was distorted and threatening."
              }

          def generate_story(
              self,
              prompt,
              genre="General",
              max_length=300,
              temperature=0.8,
              top_k=50,
              top_p=0.95,
              num_return_sequences=1
          ):
              """
              Generate a creative story based on user input.

              Args:
                  prompt (str): The story starter/prompt from user
                  genre (str): Story genre (Fantasy, Sci-Fi, etc.)
                  max_length (int): Maximum length in tokens (100-500)
                  temperature (float): Creativity level (0.1-1.5)
                      - Lower = more focused and predictable
                      - Higher = more creative and random
                  top_k (int): Limits vocabulary choices (10-100)
                  top_p (float): Nucleus sampling parameter (0.5-1.0)
                  num_return_sequences (int): Number of stories to generate

              Returns:
                  list: Generated story texts
              """
              # Combine genre context with user prompt
              if genre != "General" and genre in self.genre_prompts:
                  full_prompt = f"{self.genre_prompts[genre]} {prompt}"
              else:
                  full_prompt = prompt

              # Set seed for reproducibility (optional)
              set_seed(42)

              print(f"\n  Generating {genre} story...")
              print(f"  Prompt: {prompt[:50]}...")

              try:
                  # Generate the story using the model
                  outputs = self.generator(
                      full_prompt,
                      max_length=max_length,
                      temperature=temperature,
                      top_k=top_k,
                      top_p=top_p,
                      num_return_sequences=num_return_sequences,
                      do_sample=True,
                      pad_token_id=self.tokenizer.eos_token_id,
                      repetition_penalty=1.2,  # Reduce repetitive text
                      no_repeat_ngram_size=3   # Avoid repeating 3-word phrases
                  )

                  # Extract generated text
                  stories = [output['generated_text'] for output in outputs]

                  print("  Story generated successfully!")
                  return stories

              except Exception as e:
                  error_msg = f"  Error generating story: {str(e)}"
                  print(error_msg)
                  return [error_msg]

          def get_story_statistics(self, story):
              """
              Calculate statistics about the generated story.

              Args:
                  story (str): The generated story text

              Returns:
                  dict: Dictionary containing various statistics
              """
              words = story.split()
              sentences = [s.strip() for s in story.split('.') if s.strip()]

              stats = {
                  "word_count": len(words),
                  "sentence_count": len(sentences),
                  "character_count": len(story),
                  "avg_word_length": sum(len(word) for word in words) / len(words) if␣
       ↪words else 0,
                  "unique_words": len(set(words))
              }

              return stats

      print("  AIStoryTeller class defined successfully!")
```

```
 AIStoryTeller class defined successfully!
```

[12]:
```
print("\n" + "=" * 70)
print("  INITIALIZING AI MODEL")
print("=" * 70)

# Optional: Suppress Hugging Face logs
from transformers.utils import logging as hf_logging
hf_logging.set_verbosity_error()

# Define the AIStoryTeller class
class AIStoryTeller:
    def __init__(self, model_name="gpt2"):
        self.model_name = model_name
        print(f"  Loading model '{model_name}'...")
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForCausalLM.from_pretrained(model_name)
        self.generator = pipeline("text-generation", model=self.model,␣
↪tokenizer=self.tokenizer)
        print("  Model loaded successfully!")

    def generate_story(self, prompt, max_length=200, temperature=0.9):
        set_seed(42)
        output = self.generator(
            prompt,
            max_length=max_length,
            temperature=temperature,
            num_return_sequences=1
        )
        return output[0]["generated_text"]

# Create an instance of the StoryTeller
storyteller = AIStoryTeller(model_name="gpt2")

print("\n  AI StoryTeller is ready to generate stories!")
print("=" * 70)
```

```
======================================================================
 INITIALIZING AI MODEL
======================================================================
 Loading model 'gpt2'…
 Model loaded successfully!

 AI StoryTeller is ready to generate stories!
======================================================================
```

[13]:
```
# 1. Run this updated class code cell first:
class AIStoryTeller:
    def __init__(self, model_name="gpt2"):
        print(f"  Loading model '{model_name}'...")
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForCausalLM.from_pretrained(model_name)
        self.generator = pipeline("text-generation", model=self.model,␣
↪tokenizer=self.tokenizer)
        print("  Model loaded successfully!")

    def generate_story(self, prompt, genre=None, max_length=200, temperature=0.
↪9, top_k=50, top_p=0.95, num_return_sequences=1):
        set_seed(42)
        if genre:
            prompt = f"[{genre} Story] {prompt}"
        output = self.generator(
            prompt,
        max_length=max_length,
        temperature=temperature,
        top_k=top_k,
        top_p=top_p,
        num_return_sequences=num_return_sequences,
        do_sample=True
        )
        return output[0]["generated_text"]

    def get_story_statistics(self, story_text):
        import re
        words = story_text.split()
        sentences = re.split(r'[.!?]+', story_text)
        characters = list(story_text.replace(" ", ""))
        unique_words = len(set(words))
        avg_word_length = sum(len(word) for word in words) / len(words) if␣
 ↪words else 0
        return {
            "word_count": len(words),
            "sentence_count": len([s for s in sentences if s.strip() != ""]),
            "character_count": len(characters),
            "unique_words": unique_words,
            "avg_word_length": sum(len(word) for word in words) / len(words) if␣
 ↪words else 0
        }

# 2. Then create the storyteller instance:
storyteller = AIStoryTeller(model_name="gpt2")

# 3. Now run your testing code
print("\n" + "=" * 70)
print("  TESTING THE STORYTELLER")
print("=" * 70)

test_prompt = "Once upon a time in a distant galaxy"
print(f"\n  Test Prompt: '{test_prompt}'")
print("\n  Generating test story...")

test_story = storyteller.generate_story(
    prompt=test_prompt,
    genre="Sci-Fi",
    max_length=150,
    temperature=0.8
)

print("\n" + "=" * 70)
print("  GENERATED TEST STORY:")
```

```python
print("=" * 70)
print(test_story)
print("=" * 70)

test_stats = storyteller.get_story_statistics(test_story)

print("\n  STORY STATISTICS:")
print(f"   • Words: {test_stats['word_count']}")
print(f"   • Sentences: {test_stats['sentence_count']}")
print(f"   • Characters: {test_stats['character_count']}")
print(f"   • Average word length: {test_stats['avg_word_length']:.1f}")
print("=" * 70)

print("\n  Test completed successfully! The model is working correctly.")
```

```
 Loading model 'gpt2'…
 Model loaded successfully!

======================================================================
 TESTING THE STORYTELLER
======================================================================

 Test Prompt: 'Once upon a time in a distant galaxy'

 Generating test story…

======================================================================
 GENERATED TEST STORY:
======================================================================
[Sci-Fi Story] Once upon a time in a distant galaxy, a massive black hole
exploded into space and began to emit a large amount of radiation. At first, the
radiation was so intense that no human was able to get close to it. This caused
a major power crisis. The first thing to do was to destroy the black hole. The
first thing to do was to try and get rid of it, to eliminate it by using a
meteorite. The meteorite was a star that had been created by the creation of the
black hole. The star had been created by the black hole, but instead of being a
small black hole, it was a large black hole. This star would explode into a
black hole, causing it to
======================================================================

 STORY STATISTICS:
   • Words: 130
   • Sentences: 8
   • Characters: 522
   • Average word length: 4.0
======================================================================

 Test completed successfully! The model is working correctly.
```

[14]:
```python
def create_story_interface(prompt, genre, length, creativity, top_k, top_p):
    """
    Wrapper function for Gradio interface.
    Takes user inputs and returns generated story with statistics.

    Args:
        prompt (str): User's story prompt
        genre (str): Selected genre
        length (int): Desired story length
        creativity (float): Temperature parameter
        top_k (int): Top-K sampling value
        top_p (float): Top-P sampling value

    Returns:
        tuple: (generated_story, statistics_text)
    """
    # Validate input
    if not prompt.strip():
        return "  Please enter a story prompt to get started!", ""

    # Show processing message
    processing_msg = f"  Generating your {genre} story...\n  Please wait..."

    try:
        # Generate the story
        stories = storyteller.generate_story(
            prompt=prompt,
            genre=genre,
            max_length=int(length),
            temperature=creativity,
            top_k=int(top_k),
            top_p=top_p,
            num_return_sequences=1
        )

        story = stories[0]

        # Calculate statistics
        stats = storyteller.get_story_statistics(story)

        # Format statistics nicely
        stats_text = f"""
  *Story Statistics:*

• *Total Words:* {stats['word_count']}
• *Sentences:* {stats['sentence_count']}
• *Characters:* {stats['character_count']}
• *Unique Words:* {stats['unique_words']}
• *Avg Word Length:* {stats['avg_word_length']:.1f} characters

 Generated at: {datetime.now().strftime('%I:%M:%S %p')}
        """

        return story, stats_text

    except Exception as e:
        error_msg = f"  Error: {str(e)}\n\nPlease try again with different␣
 ↪parameters."
        return error_msg, ""

print("  Interface function defined successfully!")
```

```
 Interface function defined successfully!
```

[15]:
```
print("\n" + "=" * 70)
print("  CREATING INTERACTIVE INTERFACE")
print("=" * 70)

# Create the Gradio interface with custom theme
with gr.Blocks(
    theme=gr.themes.Soft(),
    title="  AI StoryTeller",
    css="""
        .main-header {text-align: center; padding: 20px;}
        .generate-btn {background: linear-gradient(45deg, #667eea 0%, #764ba2␣
 ↪100%);}
    """
) as demo:

    # Header section
    gr.Markdown(
        """
        #   Interactive AI StoryTeller
        ### Powered by GPT-2 & Natural Language Processing

        Welcome! Enter a story prompt, choose your preferences, and watch as AI␣
 ↪brings your imagination to life.
        Perfect for creative writing, brainstorming, or just having fun!
        """,
        elem_classes="main-header"
    )

    # Main layout with two columns
    with gr.Row():
        # LEFT COLUMN - Input Controls
        with gr.Column(scale=1):
            gr.Markdown("###   Story Configuration")

            # Story prompt input
            prompt_input = gr.Textbox(
                label="Story Prompt",
                placeholder="Example: 'In a world where magic is forbidden, a␣
↪young girl discovers...'",
                lines=4,
                info="Enter the beginning of your story or a scenario"
            )

            # Genre selection
            genre_dropdown = gr.Dropdown(
                choices=[
                    "General",
                    "Fantasy",
                    "Sci-Fi",
                    "Mystery",
                    "Horror",
                    "Romance",
                    "Adventure",
                    "Comedy",
                    "Thriller"
                ],
                value="General",
                label="  Genre",
                info="Select the story genre for better context"
            )

            # Story length slider
            length_slider = gr.Slider(
                minimum=100,
                maximum=500,
                value=250,
                step=50,
                label="  Story Length (tokens)",
                info="Longer stories take more time to generate"
            )

            # Creativity slider
            creativity_slider = gr.Slider(
                minimum=0.1,
                maximum=1.5,
                value=0.8,
                step=0.1,
                label="  Creativity Level (temperature)",
                info="Lower = more focused, Higher = more creative"
            )

            # Advanced settings (collapsible)
            with gr.Accordion("  Advanced Settings", open=False):
                top_k_slider = gr.Slider(
                    minimum=10,
                    maximum=100,
                    value=50,
                    step=10,
                    label="Top-K Sampling",
                    info="Limits vocabulary choices (40-60 recommended)"
                )

                top_p_slider = gr.Slider(
                    minimum=0.5,
                    maximum=1.0,
                    value=0.95,
                    step=0.05,
                    label="Top-P (Nucleus Sampling)",
                    info="Controls diversity (0.9-0.95 recommended)"
                )

            # Generate button
            generate_btn = gr.Button(
                "  Generate Story",
                variant="primary",
                size="lg",
                elem_classes="generate-btn"
            )

            # Tips section
            gr.Markdown(
                """
                ###   Quick Tips:
                - *Higher creativity* = more random and imaginative
                - *Lower creativity* = more focused and coherent
                - *Longer length* = more detailed story (but slower)
                - Try different genres for varied storytelling styles!
                """
            )

        # RIGHT COLUMN - Output Display
        with gr.Column(scale=2):
            gr.Markdown("###   Your Generated Story")
            # Story output
            story_output = gr.Textbox(
                label="Generated Story",
                lines=18,
                show_copy_button=True,
                placeholder="Your story will appear here... Click 'Generate␣
↪Story' to begin!",
                info="Copy the story using the button in the top-right corner"
            )

            # Statistics output
            stats_output = gr.Markdown(
                "  Story statistics will appear here after generation..."
            )

    # Example prompts section
    gr.Markdown("###   Example Prompts to Get You Started")

    gr.Examples(
        examples=[
            [
                "The spaceship crashed on an unknown planet covered in purple␣
↪vegetation",
                "Sci-Fi",
                250,
                0.8,
                50,
                0.95
            ],
            [
                "She found a mysterious key in her grandmother's attic that␣
↪glowed in the dark",
                "Mystery",
                200,
                0.7,
                40,
                0.9
            ],
            [
                "The dragon awakened after a thousand years of slumber, hungry␣
↪and confused",
                "Fantasy",
                300,
                0.9,
                60,
                0.95
            ],
            [
                "He never believed in ghosts until that stormy night in the␣
↪abandoned hospital",
                "Horror",
                250,
                0.8,
                50,
                0.92
            ],
            [
                "The time machine worked, but she arrived in the wrong century",
                "Sci-Fi",
                280,
                0.85,
                55,
                0.93
            ],
            [
                "The cafe was closing when a stranger walked in with an unusual␣
↪request",
                "Romance",
                220,
                0.75,
                45,
                0.9
            ]
        ],
        inputs=[
            prompt_input,
            genre_dropdown,
            length_slider,
            creativity_slider,
            top_k_slider,
            top_p_slider
        ],
        label="Click any example to try it out!"
    )

    # Connect the generate button to the function
    generate_btn.click(
        fn=create_story_interface,
        inputs=[
            prompt_input,
            genre_dropdown,
            length_slider,
            creativity_slider,
            top_k_slider,
            top_p_slider
        ],
        outputs=[story_output, stats_output]
    )

print("  Interface created successfully!")
print("=" * 70)
```

```
======================================================================
  CREATING INTERACTIVE INTERFACE
======================================================================
  Interface created successfully!
======================================================================
IMPORTANT: You are using gradio version 4.8.0, however version 4.44.1 is
available, please upgrade.
--------
```

[ ]:
```
print("\n" + "=" * 70)
print("  LAUNCHING AI STORYTELLER APPLICATION")
print("=" * 70)
print("\n  Instructions:")
print("   1. The interface will open in a new browser tab")
print("   2. A public URL will be generated (valid for 72 hours)")
print("   3. Share the public URL with others to let them try it!")
print("   4. To stop: Click the   button or interrupt the kernel")
print("\n  Launching...")
print("=" * 70 + "\n")

# Launch the interface
demo.launch(
    share=True,          # Creates a public shareable link
    debug=True,          # Shows detailed error messages
    show_error=True,     # Displays errors in the interface
    inbrowser=True       # Automatically opens in browser
)
```

```
======================================================================
  LAUNCHING AI STORYTELLER APPLICATION
======================================================================

  Instructions:
   1. The interface will open in a new browser tab
   2. A public URL will be generated (valid for 72 hours)
   3. Share the public URL with others to let them try it!
   4. To stop: Click the   button or interrupt the kernel
```

```
 Launching…
======================================================================

Running on local URL:  http://127.0.0.1:7863

Could not create share link. Please check your internet connection or our status
page: https://status.gradio.app.

<IPython.core.display.HTML object>
```

```
[ ]:
```