Bedtime.ai: AI-Powered Personalized Bedtime Story System for Kids

Hoyath Ali
hsing093@ucr.edu

Arnav Panigrahi
apani017@ucr.edu

Parth Shinde
pshin022@ucr.edu

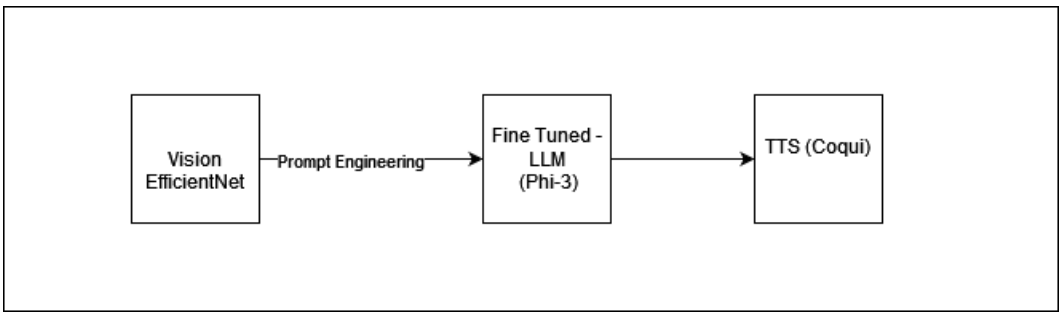
Bedtime.ai: Click here to access the application

Abstract

In today's technology-driven environment, children are increasingly exposed to AI-powered tools and platforms. However, these interactions can lack the emotional connection that traditional activities, such as bedtime storytelling with parents, provide. This project aims to bridge this gap by creating a personalized AI-driven bedtime story system that combines AI advancements with parental involvement. By narrating stories in a parent's cloned voice and tailoring stories to a child's drawings, our solution aims to create a warm, personalized experience that strengthens family bonds.

1 Introduction

In today's technology-driven world, children are increasingly engaging with AI-powered tools and platforms. While these interactions offer educational and interactive benefits, they often lack the emotional depth inherent in traditional activities like bedtime storytelling with parents. This project aims to bridge this gap by developing a **multimodal AI-driven bedtime story system** that integrates voice cloning and image analysis to create personalized and emotionally engaging narratives. By cloning a parent's voice and tailoring stories based on a child's drawings, the system provides a comforting and interactive experience that strengthens family bonds. Additionally, the implementation of content filtering ensures a safe and child-friendly environment. This innovative approach leverages advanced AI techniques to enhance the storytelling experience, fostering both emotional and cognitive development in children.



2 Objectives

The primary objectives of this project are:

- **Voice Cloning**: Develop a bedtime storytelling agent that narrates stories in a parent's voice to create a comforting and familiar experience for children.
- **Personalized Story Generation**: Enable story creation based on a child's interests, parents' preferences (e.g., educational themes), and children's drawings to ensure content relevance and engagement.
- **Content Safety**: Ensure a safe, child-friendly environment by fine-tuning language models to filter out inappropriate or harmful content.
- **User-Friendly Interface**: Develop an intuitive web interface that allows parents to manage preferences and enables children to interact seamlessly with the storytelling agent.

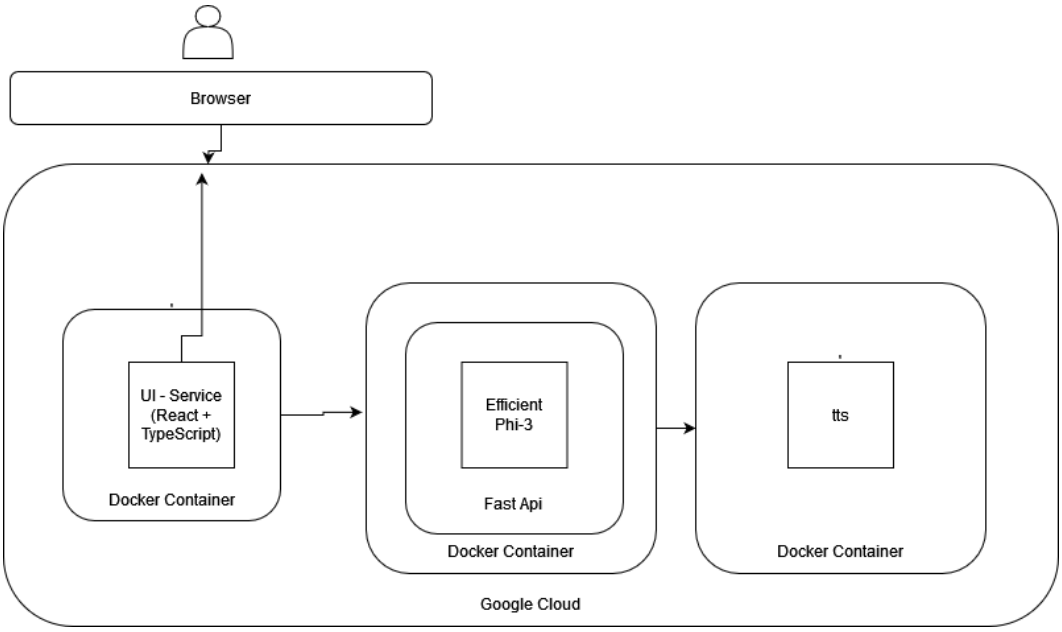


Figure 1: Application Architecture

3 Technologies and Methodologies

This project leverages a combination of advanced AI models and robust system infrastructure to create a personalized AI-driven bedtime story system. The key technologies and methodologies employed are outlined below:

3.1 AI Models and Techniques

- **Image Classification**: Utilized *EfficientNet*, a state-of-the-art convolutional neural network, to accurately classify children's drawings. This classification enables the system to generate stories that are tailored to the child's artistic expressions and interests.
- **Story Generation**: Fine-tuned a *Phi3* large language model (LLM) using Low-Rank Adaptation (LoRA) on a specialized story dataset. This enhancement allows the model to produce coherent and engaging bedtime stories that align with parental preferences and educational themes.
- **Voice Cloning**: Implemented the *Coqui TTS* model to clone parental voices. This capability ensures that stories are narrated in a familiar and comforting voice, enhancing the emotional connection between the child and the storytelling experience.

3.2 System Infrastructure

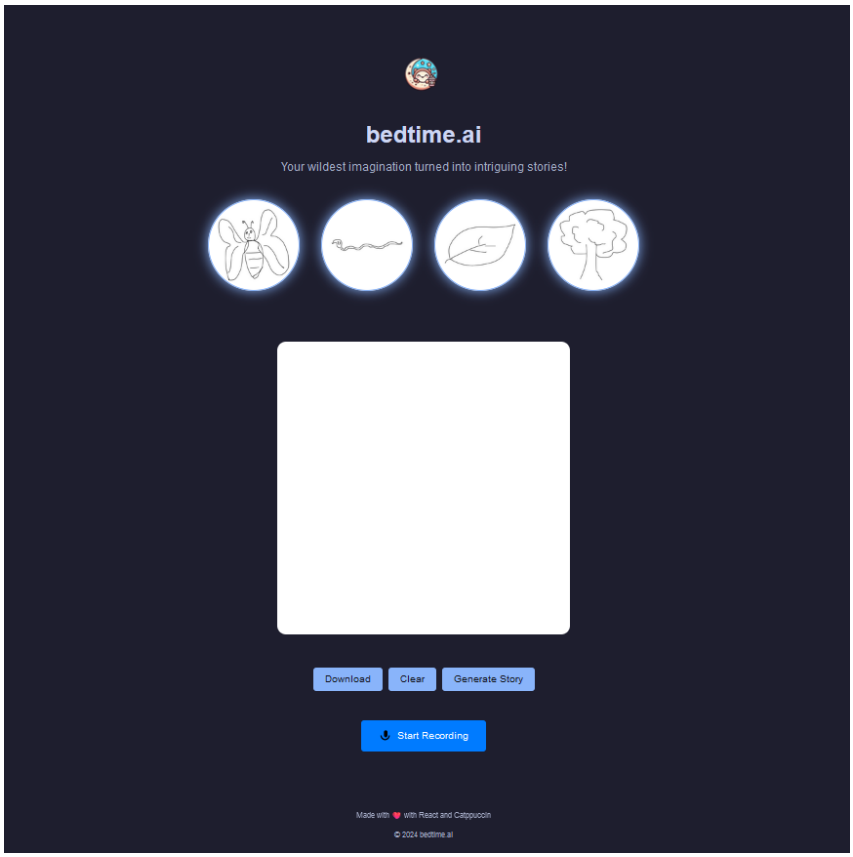
- **Frontend Development**: Developed an intuitive and responsive web interface using *React*, *Vite*, and *TypeScript*. This frontend allows parents to manage preferences and children to interact seamlessly with the storytelling agent.

- **Backend Development**: Built the backend using *FastAPI*, facilitating efficient and secure API interactions between the frontend and AI models. FastAPI ensures rapid response times and scalability to handle multiple user requests.

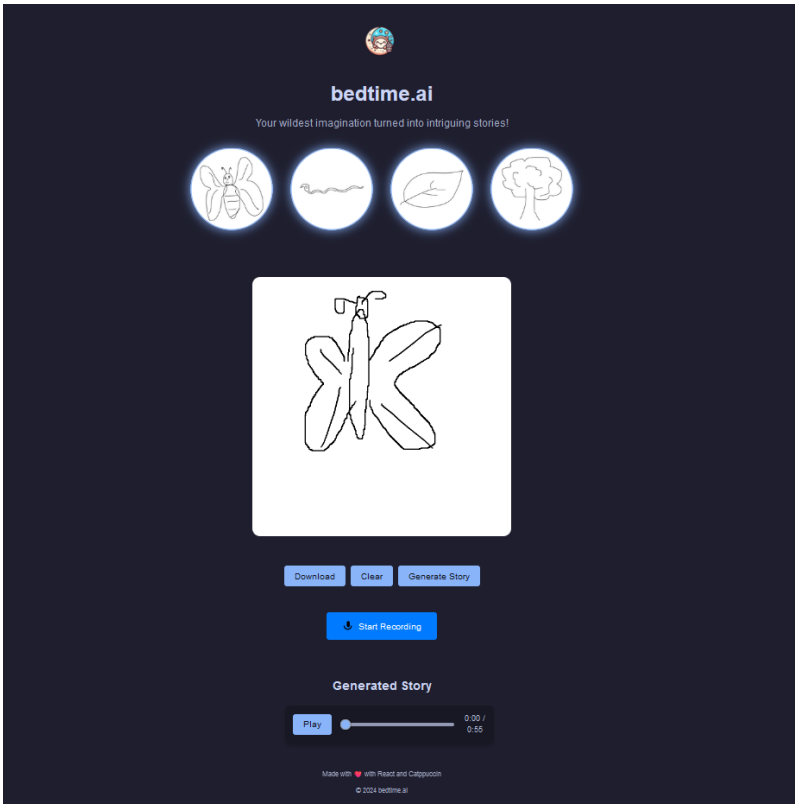
- **Containerization and Deployment**: Employed *Docker* for containerization, ensuring that all components of the system are portable and consistent across different environments. The entire application is hosted on *Google Cloud*, providing a reliable and scalable infrastructure to support the AI-driven storytelling services.

4 User Interface Overview

To provide a comprehensive understanding of *Bedtime.ai*'s functionality, we present screenshots of the application's user interface. These visuals illustrate the ease of navigation, customization options, and overall user experience designed for both parents and children.



These interfaces are meticulously designed to ensure a seamless and engaging experience, facilitating effortless interaction with the AI-driven storytelling functionalities.



5 Technologies and Methodologies

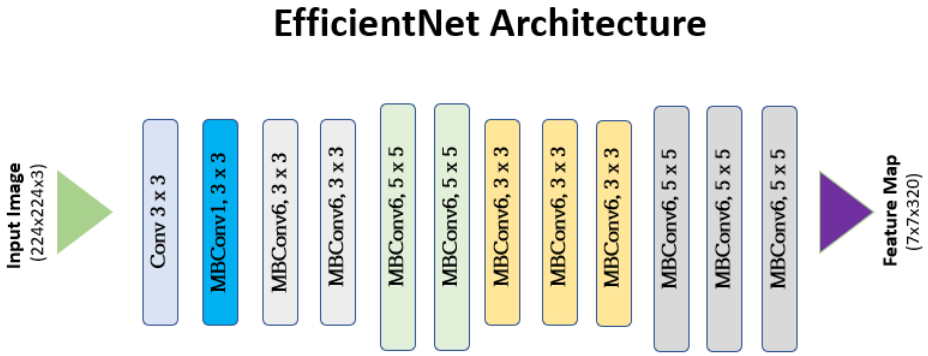
This project integrates advanced AI models and robust system infrastructure to develop a personalized AI-driven bedtime story system. The following subsections detail the key technologies and methodologies employed.

5.1 AI Models and Techniques

5.1.1 Image Classification

For the image classification component, we utilized the *EfficientNet-B0* architecture, renowned for its efficiency and high performance in image recognition tasks (2). EfficientNet-B0 was selected due to its balanced trade-off between accuracy and computational complexity.

Model Adaptation EfficientNet-B0 is originally designed for RGB images with three input channels. To accommodate grayscale drawings submitted by children, we modified the first convolutional layer to accept a single input channel. This adaptation enables the model to effectively process and classify grayscale images.



Implementation Below is the Python implementation of the modified EfficientNet-B0 model using PyTorch:

```python
import torch
import torch.nn as nn
from torchvision.models import efficientnet_b0, EfficientNet_B0_Weights

def load_model(num_classes, model_path="app/artifacts/model.pth"):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # Load pre-trained EfficientNet-B0
    model = efficientnet_b0(weights=EfficientNet_B0_Weights.IMAGENET1K_V1)

    # Modifying the first convolutional layer to accept 1 channel instead of 3
    model.features[0][0] = nn.Conv2d(
        in_channels=1,
        out_channels=model.features[0][0].out_channels,
        kernel_size=model.features[0][0].kernel_size,
        stride=model.features[0][0].stride,
        padding=model.features[0][0].padding,
        bias=False
    )

    # Modifying the classifier to match the number of classes
    model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes)

    # Load trained weights
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.to(device)
    model.eval()
    return model
```

Listing 1: Modified EfficientNet-B0 for Grayscale Image Classification

Dataset and Training We fine-tuned the modified EfficientNet-B0 model using the TU-Berlin dataset (1), a comprehensive collection of children's drawings. The fine-tuning process involved adjusting the model weights to specialize in classifying the diverse and creative representations found in the dataset.

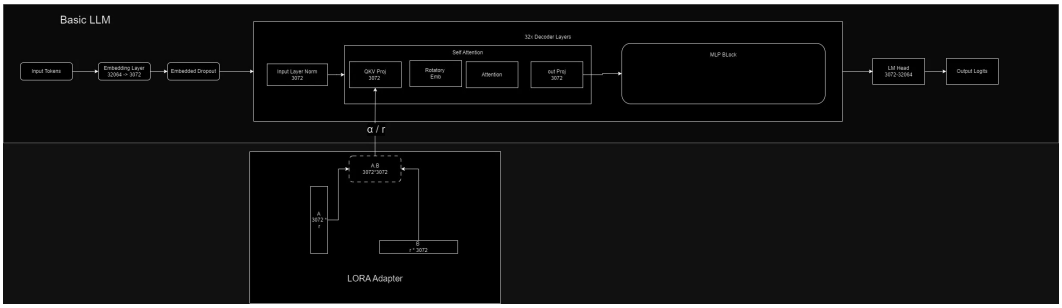
Performance The adapted EfficientNet-B0 model achieved an F1 score of **0.84**, demonstrating its effectiveness in accurately classifying children's drawings. This performance ensures reliable interpretation of visual inputs, enabling the system to generate personalized and engaging bedtime stories based on the classified drawings.

5.1.2 Story Generation

For the story generation component, we employed the *Phi-3 Mini-4K Instruct* large language model (LLM) developed by Microsoft. To optimize the model for our specific application, we implemented several advanced techniques to enhance its efficiency and performance.

Fine-Tuning with LoRA To adapt the Phi-3 model for generating engaging and contextually relevant bedtime stories, we fine-tuned it using Low-Rank Adaptation (LoRA). LoRA allows for efficient fine-tuning by injecting trainable rank-decomposition matrices into the existing model architecture, significantly reducing the number of trainable parameters and computational resources required. This approach enabled the model to learn storytelling nuances from a specialized story dataset, ensuring that the generated narratives align with both parental preferences and children's interests.

4-Bit Quantization To further optimize the model's performance and reduce its memory footprint, we applied 4-bit quantization using the *BitsAndBytes* library. This quantization technique compresses the model weights from higher precision (e.g., 32-bit floating-point) to 4-bit representations without substantial loss in accuracy. Specifically, we configured the quantization settings to compute in FP16 precision and utilized the 'nf4' quantization type for enhanced efficiency. This optimization facilitates faster inference times and enables deployment on resource-constrained environments, such as edge devices.



Selective Layer Adaptation Recognizing the importance of focusing on the most impactful layers for adaptation, we strategically selected the middle layers of the Phi-3 model for LoRA integration. Out of the model's 32 layers, we targeted layers 16 to 24, which are pivotal in capturing intricate language patterns and contextual dependencies essential for coherent story generation.

To determine the optimal layers for adaptation, we conducted extensive experiments by systematically targeting different sets of layers:

- **Starting Layers**: Initially, we integrated LoRA into the first 8 layers, which primarily focus on low-level feature extraction and token-level relationships. While this configuration demonstrated improvements in understanding basic language constructs, it struggled with maintaining coherence in longer, complex narratives, leading to higher perplexity scores.
- **Ending Layers**: Next, we applied LoRA to the final 8 layers, which are responsible for fine-tuning and generating outputs. While this approach improved the stylistic and syntactic quality of the generated stories, it lacked depth in contextual understanding and logical progression.
- **All Layers**: We also explored a broader approach by applying LoRA to all 32 layers. Although this maximized the model's capacity for adaptation, it significantly increased computational costs and overfitted the model to the training data, resulting in poorer generalization and only marginal gains in perplexity.

After evaluating these configurations, we identified that targeting the middle layers (16 to 24) provided the best balance of performance and efficiency. These layers act as a bridge between low-level token features and high-level narrative generation, capturing contextual dependencies and intricate language patterns effectively. This configuration yielded the lowest perplexity scores, indicating superior model performance in generating coherent and contextually rich stories.

By focusing our adaptation efforts on these middle layers, we achieved an optimal trade-off between model accuracy and computational efficiency, ensuring that the Phi-3 model was fine-tuned to generate engaging and personalized bedtime stories.

Data Preparation Effective story generation relies heavily on the quality and relevance of the training data. We utilized the *TinyStoriesWithTags* (3) dataset, which comprises short, imaginative bedtime stories annotated with relevant tags. To further enhance the dataset, we employed advanced natural language processing (NLP) techniques to generate additional tags from the stories. Specifically, we implemented the following:

- **Tag Extraction with NLP**: Using NLP libraries such as SpaCy and NLTK, we extracted nouns, verbs, and adjectives from the text of each story. This process involved tokenizing the text, applying part-of-speech (POS) tagging to identify relevant linguistic elements, and filtering out stop words and non-descriptive tokens. For example, from a story like "The brave little fox found a shiny red ball in the forest," tags such as *brave, fox, shiny, red, ball, forest* were generated. These descriptive tags were added as a new column in the dataset, enriching the metadata and providing valuable contextual cues for the model.
- **Semantic Refinement**: To improve tag quality, we applied lemmatization to standardize words to their base forms (e.g., "running" to "run") and used named entity recognition (NER) to identify specific entities such as characters, places, and objects. This ensured that the tags captured not only descriptive attributes but also key entities within each story.
- **Tag Deduplication and Noise Removal**: Post-extraction, we performed a deduplication process to eliminate redundant tags and employed text cleaning methods to remove irrelevant or noisy tags that could detract from the model's focus. For instance, generic words like "said" or "went" were excluded unless critical to the context.

The enriched dataset, now equipped with manually curated and NLP-generated tags, underwent additional preprocessing steps:

- **Special Token Integration**: To facilitate structured storytelling, we introduced additional special tokens such as ‘<instruction>‘ and ‘<story>‘ to demarcate different sections of the input data. This structuring aids the model in understanding the context and generating appropriately formatted narratives.
- **Tokenization and Padding**: We employed the tokenizer to convert textual data into token IDs, ensuring that both instructions and stories were appropriately truncated and padded to maintain uniform input lengths. Padding tokens were added to align sequences to a maximum length, and instruction tokens were masked in the labels to focus the model's learning on story generation.
- **Dataset Filtering**: To maintain data quality, we filtered out entries with missing tags or stories, ensuring that the model is trained on complete and relevant data points.

By integrating these NLP-driven enhancements, the dataset was made more robust and informative, enabling the Phi-3 model to generate personalized and engaging bedtime stories. The rich tags provided additional context, ensuring that the generated narratives aligned closely with each child's unique interests and artistic expressions.

5.2 Training Details

The training process was meticulously designed to ensure computational efficiency while maximizing the model's capacity to generate high-quality stories. We leveraged the Low-Rank Adaptation (LoRA) technique, which allows targeted fine-tuning of a small subset of parameters, significantly reducing the computational overhead. The hyperparameters utilized during training were as follows: `'r': 16, 'loraalpha': 32, 'loradropout': 0.5`, where:

- `r`: Represents the rank of the low-rank decomposition, controlling the dimensionality of the adaptation matrices. A value of 16 provided a good balance between model expressiveness and computational efficiency.
- `loraalpha`: A scaling factor used to amplify the low-rank updates. Setting it to 32 ensured stable gradients and effective learning without overwhelming the pre-trained weights.
- `loradropout`: Dropout rate applied to the adaptation layers, set at 0.5 to introduce regularization and prevent overfitting, especially given the small percentage of trainable parameters.

The resulting parameter statistics were as follows:

- **Trainable Parameters:** 1,753,088 This small subset of parameters was selectively fine-tuned to adapt the pre-trained Phi-3 model for story generation, focusing on middle layers as described in our selective adaptation approach.
- **Total Parameters:** 3,822,519,296 The sheer scale of total parameters highlights the richness of the base model's representation, which remains largely untouched during fine-tuning.
- **Percentage Trainable:** 0.0459Only a minuscule fraction of the parameters were trainable, showcasing the efficiency of the LoRA technique. This low percentage allowed us to achieve high-quality adaptation without requiring significant computational resources or risking overfitting.

5.3 Training Results

The model was trained over 30 epochs, with significant performance improvements observed during the initial epochs. Training and validation statistics are summarized below:

Epoch	Average Training Loss	Validation Loss
1	1.1414	1.0731
2	1.0319	1.0476
3	1.0093	1.0377
4	0.9937	1.0221
5	0.9792	1.0168
6	0.9680	1.0164
7	0.9583	1.0182
8	0.9501	1.0205
9	0.9428	1.0210
Early Stopping Triggered at Epoch 9		

Table 1: Training and Validation Loss Over Epochs with Early Stopping at Epoch 9

Validation loss consistently improved, with the model checkpoint saved whenever the validation loss decreased.

5.3.1 Voice Generation

To deliver the final personalized bedtime story in a comforting and familiar tone, we employed the *Coqui TTS* model for voice cloning. This component plays a pivotal role in creating an emotional connection between children and AI by narrating stories in a parent's cloned voice. By bridging the gap between textual storytelling and natural speech, this system enhances engagement, making the storytelling experience truly magical for children.

Model Selection and Integration The *Coqui TTS* framework was chosen for its state-of-the-art text-to-speech capabilities, offering exceptional performance in generating realistic and expressive speech. It supports GPU acceleration, enabling efficient voice synthesis even for large-scale or real-time applications. By leveraging its pre-trained TTS models, we implemented voice cloning tailored to the parent's voice.

Pipeline Overview The voice generation pipeline seamlessly integrated multiple components to deliver an end-to-end personalized storytelling experience:

- The story text, generated by the *EfficientNet*-driven drawing predictions and the *Phi-3* storytelling model, was first processed to ensure clarity and natural language fluency.
- This text was fed into the fine-tuned *Coqui TTS* model, which synthesized high-quality audio in the cloned parent's voice.
- The synthesized audio included subtle variations in tone and pacing to match the story's emotional highs and lows, ensuring an engaging and lifelike narration.
- Optional background sound effects and music were dynamically added based on story content, enhancing the multimodal storytelling experience and immersing the listener in the narrative world.

Impact on Storytelling Experience The integration of voice cloning with storytelling fundamentally transformed the bedtime story experience. The familiar voice of a parent narrating imaginative tales created a deep emotional connection, fostering comfort and joy for children. The lifelike quality of the cloned voice ensured that the narration was not just a replication but an authentic extension of parental presence.

This multimodal system, combining cutting-edge text generation and voice synthesis, elevated bedtime storytelling into a personalized, interactive, and heartwarming experience, making technology an enabler of cherished family moments.

6 Discussion and Challenges

During the development of this project, several technical and ethical challenges were encountered. This section outlines the key challenges and the solutions implemented to address them.

6.1 Challenges

Adapting EfficientNet for Grayscale Input EfficientNet is designed to process RGB images with three input channels, which posed a challenge for our application involving grayscale drawings. Modifying the initial convolutional layers to accept a single input channel required careful adjustments to ensure compatibility with the existing model architecture while maintaining performance. The modified model was then fine-tuned on a drawing dataset to achieve satisfactory accuracy and F1 score, demonstrating its effectiveness for grayscale inputs.

Fine-Tuning the Phi-3 Model with LoRA Fine-tuning the large Phi-3 language model presented challenges due to its size and computational demands. Fine-tuning the entire model was impractical, so we opted for Low-Rank Adaptation (LoRA). However, configuring LoRA involved several decisions, including selecting the appropriate layers for fine-tuning and determining the optimal rank size. After evaluating multiple configurations, we identified layers 16–24 as the most impactful and used a rank size of 16. This selective fine-tuning significantly reduced computational costs while maintaining the model's ability to generate high-quality stories.

Deploying the Voice Cloning System Hosting the Coqui TTS voice cloning model as an API service was another complex task. The system needed to handle both audio input for voice samples and text-to-speech generation efficiently. Building and deploying this service using Docker with GPU and CUDA support for faster inference required meticulous configuration to ensure reliability and scalability. Despite the initial challenges, the final deployment provided seamless integration with the storytelling pipeline, enabling real-time voice synthesis.

6.2 Solutions to Technical and Ethical Concerns

Technical Solutions For each technical challenge, specific solutions were implemented:

- For EfficientNet, we modified the initial layers and retrained the model on grayscale drawings, ensuring compatibility without compromising performance.

- For Phi-3, we utilized LoRA to fine-tune selected layers, reducing resource requirements while preserving storytelling quality.
- For voice cloning, we containerized the Coqui TTS model using Docker and optimized GPU usage with CUDA for efficient inference and scalability.

7 System Deployment

The deployment of the *Bedtime.ai* system involved integrating multiple technologies and frameworks to ensure scalability, efficiency, and robustness. This section details the steps and tools used for the seamless deployment of the system.

7.1 Containerization with Docker

To ensure portability and consistency across different environments, we containerized the application using *Docker*. Each component of the system, including the frontend, backend, and AI models, was encapsulated in separate Docker containers. This modular approach facilitated easy updates and debugging while isolating dependencies to avoid conflicts.

Key features of the Docker setup:

- A `Dockerfile` was created for each module to define the dependencies, environment configurations, and runtime instructions.
- `docker-compose` was used to orchestrate multi-container setups, allowing efficient communication between the frontend, backend, and model inference services.
- GPU support was enabled for AI models using NVIDIA's Docker runtime, ensuring efficient execution of computationally intensive tasks such as voice synthesis and story generation.

7.2 Backend Development and Deployment with FastAPI

The backend of the system was built using *FastAPI*, a high-performance web framework. FastAPI provided efficient API routing and response handling, enabling seamless interactions between the frontend and the underlying AI models.

Key aspects of the backend deployment:

- API endpoints were designed to handle story generation, drawing classification, and voice synthesis requests with low latency.
- Integration with the AI models was streamlined, with inference pipelines optimized for real-time processing.
- Authentication and security measures were implemented to ensure safe and secure interactions.

7.3 Cloud Hosting on Google Cloud

The system was deployed on a *Google Cloud GPU instance* to leverage its computational power and robust performance for running AI models efficiently. Key deployment steps included:

- Hosting the Docker containers on a GPU-enabled *Google Cloud Compute Engine* instance for high-performance execution of AI tasks.
- Using *Google Cloud Storage* to manage user-uploaded data, including voice samples and drawings.
- Configuring the instance's network settings and firewall rules to ensure secure and reliable access to the application.
- Setting up domain and DNS configurations for the application, providing an accessible and user-friendly interface.

7.4 Deployment Workflow

The deployment process involved the following workflow:

1. Each module (frontend, backend, and models) was containerized and tested locally using Docker.
2. The containers were deployed directly on a GPU-enabled Google Cloud Compute Engine instance, configured to handle the application's computational requirements.
3. Continuous Integration and Continuous Deployment (CI/CD) pipelines were established using *GitHub Actions* to automate testing, building, and deployment.
4. Health checks and monitoring were implemented using GCP's monitoring tools, ensuring system reliability and uptime.

8 Conclusion

In this project, we developed *Bedtime.ai*, an AI-powered personalized bedtime story system designed to enhance the bedtime experience for children by integrating advanced AI technologies with parental involvement. By leveraging voice cloning, image classification, and story generation, our system delivers customized and emotionally engaging narratives that foster stronger family bonds and support children's emotional and cognitive development.

The successful implementation of voice cloning using the *Coqui TTS* model ensures that stories are narrated in a familiar and comforting voice, while the fine-tuned *Phi-3* language model generates coherent and relevant stories based on children's drawings and preferences. Additionally, the robust system infrastructure, including a user-friendly web interface and scalable deployment on Google Cloud, guarantees a seamless and secure user experience.

Overall, *Bedtime.ai* bridges the gap between traditional storytelling and modern AI interactions, providing a unique and enriching bedtime routine that adapts to each child's individuality and fosters a nurturing environment.

9 Future Work

While *Bedtime.ai* presents a comprehensive solution for personalized bedtime storytelling, there are several avenues for future enhancement:

- **Multilingual Support**: Extending the system to support multiple languages would cater to a more diverse user base and support language learning for children.
- **Interactive Story Elements**: Incorporating interactive elements such as choosing story paths or integrating voice recognition to allow children to participate actively in the storytelling process.
- **Enhanced Personalization**: Utilizing more sophisticated AI techniques to analyze a wider range of inputs, such as voice tone or behavioral patterns, to further personalize story content and delivery.
- **Mobile Application Development**: Developing dedicated mobile applications for both iOS and Android platforms to increase accessibility and convenience for users.
- **Parental Feedback Integration**: Implementing mechanisms for parents to provide feedback on generated stories, enabling continuous improvement and adaptation of the storytelling algorithms.
- **Security and Privacy Enhancements**: Strengthening data protection measures to ensure the utmost security and privacy of user data, particularly given the involvement of children.

These enhancements will further refine the user experience, expand the system's capabilities, and ensure that *Bedtime.ai* remains a valuable tool for families seeking personalized and engaging bedtime routines.

10 Team Contributions

Name	Task 1	Task 2	Task 3
Hoyath	- Phi-3 Fine Tuning - Quantization - LoRA	- Deployment - Docker - Google Cloud	Host & DNS Setup
Arnav	- Vision - Data Pre-Processing - EfficientNet Fine Tuning	- UI Component - React - Vite - TypeScript	- Testing Backend
Parth	- Text-to-Speech Model Setup - Data Pre-Processing for LLM Fine Tuning	- Backend Integration - FastAPI - Inference Pipeline	- Testing Frontend

Table 2: Team Contributions

11 GPT and Claude Usage

GPT and Claude were utilized to generate boilerplate code for various aspects of the project, including setting up backend infrastructure, drafting foundational code for AI model implementations, and facilitating fine-tuning processes, enabling the team to focus on refining and customizing the generated code to meet project-specific requirements.

References

[1] TU Berlin Research Team. The TU Berlin Dataset: A Large-Scale Collection of Hand-Drawn Sketches. Available at: `https://huggingface.co/datasets/sdiaeyu6n/tu-berlin`, accessed November 29, 2024.

[2] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Available at: `https://huggingface.co/docs/transformers/en/model_doc/efficientnet`, accessed November 29, 2024.

[3] Ronen Eldan and others. *TinyStories: Dataset containing synthetically generated (by GPT-3.5 and GPT-4) short stories that only use a small vocabulary*. Available at: `https://huggingface.co/datasets/roneneldan/TinyStories`, accessed December 7, 2024. Described in the paper "TinyStories: Synonym-Limited Short Stories Dataset," available at `https://arxiv.org/abs/2305.07759`.