|  |  |  |
| --- | --- | --- |
|  | **MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE**  **Kodambakkam, Chennai-600024** |  |

**SB3001 - PROJECT-BASED EXPERIENTIAL LEARNING**

**PROGRAM**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**TOPIC: AI-Generated Music Composition with Transformer Models**

|  |  |  |
| --- | --- | --- |
| **FACULTY MENTOR :** | **M.SOWMIYA (Assisstant Professor)** |  |
| **FACULTY INCHARGE :** | **P. REVATHI (Assisstant Professor)** |  |
|  |  |  |

**Project submitted by,**

**SELVAKUMAR P (311521104052)**

***Project report format***

1. **ABSTRACT**
2. **INTRODUCTION**
   1. Project Overview
   2. Purpose
3. **IDEATION AND PROPOSED SOLUTION**
   1. Problem statement definition
   2. Ideation and Brainstorming
   3. Proposed Solution
4. **REQUIREMENTS ANALYSIS**
   1. Functional Requirements
   2. Non-Functional Requirements
5. **PROJECT DESIGN** 
   1. Briefing
   2. Solution
6. **SOLUTIONS**
   1. Development Part I
   2. Development Part II
7. **RESULTS**

7.1 Performance Metrics

1. **ADVANTAGES AND DISADVANTAGES**
2. **CONCLUSION**
3. **FUTURE SCOPE**

SOURCE CODE

**ABSTRACT**

The emergence of artificial intelligence (AI) has revolutionized various domains, including music composition. This paper delves into the application of Transformer models, a deep learning architecture, for AI-generated music composition. Our project endeavors to exploit Transformer models to automate the creative process of music composition, aiming to produce original musical pieces with minimal human intervention.

Through extensive ideation and brainstorming, we propose a solution that entails training Transformer models on extensive datasets of musical compositions. We delineate the functional and non-functional requirements necessary for the successful implementation of our proposed solution, covering aspects such as data preprocessing, model architecture design, and performance evaluation metrics.

The project design encompasses the entire workflow, from data collection and preprocessing to model training, evaluation, and deployment. We delve into the development process, dividing it into two parts: data preprocessing and model architecture selection, followed by model training and refinement. Evaluation of the AI-generated music compositions is conducted using various metrics, including melody coherence, harmonic progression, and rhythm consistency.

We discuss the advantages and disadvantages of employing Transformer models for music composition, considering factors such as diversity of musical styles, potential biases, and ethical implications. In conclusion, our project showcases the potential of Transformer models in AI-driven music composition, paving the way for future research endeavors in this exciting domain.

This paper explores the application of Transformer models in generating music through artificial intelligence (AI). We delve into the ideation, development, and implementation of such models, evaluating their performance metrics and discussing their advantages and disadvantages. Additionally, we propose future directions for research in this field.

**INTRODUCTION**

**Project Overview:**

QR codes have become ubiquitous in modern society, serving as versatile tools for encoding and decoding information across various applications. From marketing campaigns to inventory management, QR codes offer a convenient and efficient means of sharing data.

However, traditional QR code generation methods often come with limitations. Static designs and limited customization options restrict the visual appeal and user engagement of QR codes. As a result, there is a growing demand for innovative approaches to QR code generation that can address these shortcomings.

The aim of this project is to explore the use of generative artificial intelligence (AI) techniques for QR code generation. By harnessing the power of generative AI, we seek to overcome the limitations of traditional methods and create QR codes that are dynamic, customizable, and visually appealing.

**Purpose:**

The purpose of this project is twofold. Firstly, we aim to investigate how generative AI techniques can be applied to QR code generation to enhance their visual aesthetics and functionality. By leveraging deep learning and neural networks, we intend to create QR codes that are more engaging and interactive.

Secondly, we seek to explore the potential benefits of using generative AI for QR code generation. This includes improving user engagement, increasing brand recognition, and enhancing the overall user experience. By pushing the boundaries of QR code technology, we aim to unlock new possibilities for information encoding and sharing.

.

**IDEATION AND PROPOSED SOLUTION**

**Ideation and Proposed Solution:**

**Problem Statement Definition:**

Traditional QR code generation methods pose several challenges, including limited design flexibility, static content, and a lack of personalization options. These constraints hinder the ability to create QR codes that effectively engage users and convey information in a visually appealing manner. As a result, there is a growing need for a solution that enables dynamic QR code generation with customizable designs, enhanced visual appeal, and adaptive content.

**Ideation and Brainstorming:**

To address these challenges, our team engaged in an extensive brainstorming process to explore potential solutions. This involved researching generative AI techniques and experimenting with different algorithms to understand their capabilities and limitations. We also considered various design principles and user preferences to ensure that our solution would meet the needs of diverse stakeholders.

During the ideation phase, we explored a range of ideas for incorporating generative AI into QR code generation. This included investigating techniques such as style transfer, which allows QR codes to mimic the visual characteristics of famous artworks or design trends. We also explored the use of GANs (Generative Adversarial Networks) and VAEs (Variational Autoencoders) to generate QR codes with unique patterns, textures, and colors. Additionally, we considered the potential use of evolutionary algorithms to evolve QR code designs over multiple iterations, optimizing for both visual appeal and scanability.

**Proposed Solution:**

Based on our brainstorming and ideation process, we propose the use of generative AI techniques, specifically GANs and VAEs, to generate dynamic QR codes with customizable designs and adaptive content. These techniques leverage the power of deep learning to generate QR codes that are visually appealing, interactive, and context-aware.

Firstly, it enables real-time customization of QR codes, allowing users to personalize the design, color scheme, and content of their QR codes according to their preferences. Secondly, it incorporates pattern recognition capabilities to generate QR codes with intricate designs, textures, and shapes that are visually engaging and aesthetically pleasing. Additionally, our solution includes content adaptation mechanisms to dynamically adjust the QR code's content based on contextual factors such as time, location, and user interaction.

**Project Steps**

* **Phase 1: Research and Planning**
* 1.1. Understanding QR Codes:
* - Learn about the concept of QR codes, their structure, encoding methods, and common applications across various industries.
* 1.2. Exploring Generative AI:
* - Research generative AI techniques such as GANs, VAEs, and other deep learning models used for image generation.
* - Understand the principles behind generative AI, including training procedures, loss functions, and model architectures.
* 1.3. Identifying Challenges:
* - Identify the limitations of traditional QR code generation methods, such as static designs, limited customization options, and lack of visual appeal.
* - Determine the potential benefits of integrating generative AI into QR code generation, including dynamic designs, personalized content, and enhanced aesthetics.
* 1.4. Project Scope and Objectives:
* - Define the scope of the project, including the specific goals, deliverables, and target outcomes.
* - Establish clear objectives for implementing QR code generation using generative AI, focusing on improving visual aesthetics, customization options, and user engagement.
* **Phase 2: Data Collection and Preparation**
* 2.1. Dataset Acquisition:
* - Gather a diverse dataset of QR codes or QR code-like patterns from various sources, including public repositories, online databases, or custom-generated datasets.
* - Ensure that the dataset covers a wide range of QR code designs, patterns, and content types to train the generative AI model effectively.
* 2.2. Data Preprocessing:
* - Clean and preprocess the dataset to remove noise, artifacts, and inconsistencies.
* - Normalize the data and resize images to a consistent resolution suitable for training the generative AI model.
* **Phase 3: Model Development and Training**
* 3.1. Model Selection:
* - Choose an appropriate generative AI model architecture for QR code generation, considering factors such as complexity, performance, and compatibility with the dataset.
* 3.2. Model Implementation:
* - Implement the selected generative AI model using a deep learning framework such as TensorFlow or PyTorch.
* - Configure the model architecture, including the number of layers, activation functions, and optimization algorithms.
* 3.3. Training Procedure:
* - Split the dataset into training, validation, and test sets for model evaluation.
* - Train the generative AI model using the training dataset, optimizing for objectives such as image fidelity, diversity, and content relevance.
* - Monitor training progress, adjust hyperparameters, and conduct experiments to optimize model performance.
* **Phase 4: Evaluation and Validation**
* 4.1. Model Evaluation Metrics:
* - Define evaluation metrics to assess the performance of the generative AI model, such as image quality metrics, diversity measures, and content relevance scores.
* 4.2. Validation Procedures:
* - Evaluate the trained model using the validation dataset, comparing generated QR codes against ground truth samples.
* - Analyze the results, identify strengths and weaknesses of the model, and iterate on the training process if necessary.
* **Phase 5: Deployment and Integration**
* 5.1. Model Deployment:
* - Deploy the trained generative AI model into a production environment, ensuring scalability, reliability, and performance.
* - Integrate the model with QR code generation applications or platforms, enabling users to create dynamic QR codes with generative AI-generated designs.
* 5.2. User Interface Design:
* - Design an intuitive user interface for interacting with the QR code generation system, providing options for customizing designs, content, and settings.
* **Phase 6: Testing and Iteration**
* 6.1. Testing Procedures:
* - Conduct thorough testing of the QR code generation system, including unit tests, integration tests, and user acceptance tests.
* - Identify and address any bugs, errors, or usability issues discovered during testing.
* 6.2. Iterative Improvement:
* - Gather feedback from users and stakeholders, incorporating suggestions and recommendations for improving the QR code generation system.
* - Iterate on the model training process, data collection strategies, and deployment infrastructure to enhance overall performance and usability.
* **Phase 7: Documentation and Reporting**
* 7.1. Documentation:
* - Document the entire project process, including methodologies, techniques, code implementations, and experimental results.
* - Provide comprehensive documentation for the generative AI model, including model architecture, training procedures, and deployment instructions.
* 7.2. Reporting:
* - Prepare a detailed project report summarizing key findings, insights, and lessons learned throughout the project lifecycle.
* - Present the project outcomes, including the trained generative AI model, evaluation results, and recommendations for future work.

**REQUIREMENT ANALYSIS**

Functional Requirements:

* Identify the functional requirements for the QR code generation system, including image generation, pattern recognition, content encoding, and customization options.
* Define the specific functionalities required to implement the proposed solution, such as data preprocessing, model training, inference, and user interface design.

Non-Functional Requirements:

* Specify the non-functional requirements for the QR code generation system, including scalability, efficiency, robustness, and usability.
* Discuss the performance metrics and quality attributes that will be used to evaluate the system, such as accuracy, speed, resource utilization, and user satisfaction.

**PROJECT DESIGN**

**Briefing:**

* Provide a high-level overview of the project design, including the architectural components, data flow, and interaction between modules.
* Outline the design principles and methodologies that will be followed in developing the QR code generation system, such as modular design, agile development, and iterative refinement.

**Solution:**

* Present the detailed design of the QR code generation system, including the technical architecture, data models, and algorithmic components.
* Describe the design decisions and trade-offs made in selecting generative AI techniques, designing the user interface, and integrating with existing systems.

**SOLUTIONS:**

**Development Part I:**

* Describe the initial development phase focused on setting up the development environment, collecting data, and implementing basic functionalities.
* Discuss the data preprocessing steps, including data cleaning, augmentation, and normalization, as well as the implementation of generative AI models for QR code generation.

**Development Part II:**

* Detail the subsequent development phase focused on refining the system, optimizing performance, and integrating additional features.
* Discuss the model training process, hyperparameter tuning, performance optimization techniques, and integration with external APIs or services.

**RESULTS:**

**Performance Metrics:**

* Present the results of the QR code generation system, including quantitative metrics and qualitative evaluations.
* Discuss the performance of the system in terms of accuracy, efficiency, scalability, user satisfaction, and aesthetic appeal.

**ADVANTAGES AND DISADVANTAGES:**

**Advantages:**

* Highlight the advantages of using generative AI for QR code generation, such as dynamic designs, customizable content, enhanced visual appeal, and adaptability to user preferences.
* Discuss the potential benefits of the system in improving user engagement, brand recognition, and information dissemination.

**Disadvantages:**

* Acknowledge the limitations and challenges associated with generative AI-based QR code generation, such as computational complexity, model interpretability, data privacy concerns, and user acceptance.

**CONCLUSION:**

* Summarize the key findings and outcomes of the project, highlighting the success of using generative AI for QR code generation.
* Reflect on the project's contributions to the field of QR code technology and the broader implications for information encoding and communication.

**FUTURE SCOPE:**

* Explore potential avenues for future research and development in QR code generation using generative AI, such as exploring new generative models, enhancing customization options, and integrating with emerging technologies.
* Discuss potential applications and use cases for the system, including marketing campaigns, interactive media, augmented reality, and secure authentication.

This comprehensive breakdown outlines the contents for each section of your project report or presentation on QR code generation using generative AI. Adjustments can be made based on the specific details and findings of your project.

**SOURCE CODE:**

import qrcode

from PIL import Image, ImageDraw

import numpy as np

import matplotlib.pyplot as plt

# Generate a synthetic QR code-like image with noise, distortion, and varied block sizes

def generate\_fake\_qr\_code\_image(size=28, noise\_level=0.05, distortion\_level=0.1, block\_size\_variation=0.1):

# Create a blank image

fake\_image = Image.new('RGB', (size, size), color='white')

draw = ImageDraw.Draw(fake\_image)

# Add noise

for \_ in range(int(size \* size \* noise\_level)):

x = np.random.randint(0, size)

y = np.random.randint(0, size)

draw.point((x, y), fill='black')

# Add distortion

for x in range(size):

for y in range(size):

if np.random.rand() < distortion\_level:

dx = np.random.randint(-1, 2)

dy = np.random.randint(-1, 2)

draw.point((x + dx, y + dy), fill='black')

# Add varied block sizes

block\_size = int(size \* (1 + np.random.uniform(-block\_size\_variation, block\_size\_variation)))

qr = qrcode.QRCode(

version=1,

error\_correction=qrcode.constants.ERROR\_CORRECT\_L,

box\_size=block\_size // size,

border=4,

)

qr.add\_data("https://www.google.com")

qr.make(fit=True)

qr\_img = qr.make\_image(fill\_color="black", back\_color="white")

qr\_img = qr\_img.resize((size, size), Image.ANTIALIAS)

fake\_image.paste(qr\_img)

return np.array(fake\_image)

# Encode a URL into the synthetic QR code-like image

def encode\_url\_into\_qr\_code(url, fake\_image, filename='generated\_qr\_code.png'):

qr = qrcode.QRCode(

version=1,

error\_correction=qrcode.constants.ERROR\_CORRECT\_L,

box\_size=10,

border=4,

)

qr.add\_data(url)

qr.make(fit=True)

qr\_img = qr.make\_image(fill\_color="black", back\_color="white")

# Resize the QR code image to match the dimensions of the fake image

qr\_img = qr\_img.resize((fake\_image.shape[1], fake\_image.shape[0]))

# Convert the QR code image to grayscale and create a binary mask

qr\_img\_gray = qr\_img.convert('L')

qr\_img\_mask = np.array(qr\_img\_gray) > 127 # Threshold to create binary mask

# Apply the binary mask to overlay the QR code on the generated image

fake\_image[qr\_img\_mask] = 255

# Create an image from the modified fake image array

fake\_image\_img = Image.fromarray(fake\_image)

# Save the image

fake\_image\_img.save(filename)

return fake\_image\_img

# Example usage

url = "https://www.google.com"

fake\_image = generate\_fake\_qr\_code\_image()

qr\_code\_image = encode\_url\_into\_qr\_code(url, fake\_image)

plt.imshow(qr\_code\_image, cmap='gray')

plt.axis('off')

plt.show()

**ARTISTIC , PATTERNIZED , LOGO EMBEDDING QR CODE-GENERATOR**

import qrcode

from IPython.display import Image, display

from PIL import Image as PILImage

from pyzbar.pyzbar import decode

def generate\_qr\_code(data, filename='qr\_code.png', size=200, border=1, error\_correction=qrcode.constants.ERROR\_CORRECT\_L, fill\_color="black", back\_color="white"):

# Generate QR code

qr = qrcode.QRCode(version=1, error\_correction=error\_correction, box\_size=10, border=border)

qr.add\_data(data)

qr.make(fit=True)

# Create an image from the QR code

qr\_img = qr.make\_image(fill\_color=fill\_color, back\_color=back\_color)

# Resize the image

qr\_img = qr\_img.resize((size, size))

# Save the image

qr\_img.save(filename)

# Display the image using IPython.display

display(Image(filename))

def batch\_generate\_qr\_codes(data\_list, prefix='qr\_code\_', size=200, border=1, error\_correction=qrcode.constants.ERROR\_CORRECT\_L, fill\_color="black", back\_color="white"):

for i, data in enumerate(data\_list):

filename = f"{prefix}{i}.png"

generate\_qr\_code(data, filename, size, border, error\_correction, fill\_color, back\_color)

def decode\_qr\_code(filename):

# Open the image containing the QR code

img = PILImage.open(filename)

# Decode the QR code

decoded\_objects = decode(img)

# Print decoded data

for obj in decoded\_objects:

print('Data:', obj.data.decode())

print('Type:', obj.type)

def embed\_logo\_into\_qr\_code(data, logo\_path, filename='qr\_code\_with\_logo.png', size=200, border=1, error\_correction=qrcode.constants.ERROR\_CORRECT\_L, fill\_color="black", back\_color="white"):

# Generate QR code

qr = qrcode.QRCode(version=1, error\_correction=error\_correction, box\_size=10, border=border)

qr.add\_data(data)

qr.make(fit=True)

# Create an image from the QR code

qr\_img = qr.make\_image(fill\_color=fill\_color, back\_color=back\_color)

# Open and resize the logo image

logo = PILImage.open(image\_path)

logo = logo.resize((int(size/2), int(size/2)))

# Calculate the position to paste the logo

position = ((qr\_img.size[0] - logo.size[0]) // 2, (qr\_img.size[1] - logo.size[1]) // 2)

# Paste the logo onto the QR code

qr\_img.paste(logo, position)

# Save the image

qr\_img.save(filename)

# Display the image using IPython.display

display(Image(filename))

if \_\_name\_\_ == "\_\_main\_\_":

# Example usage

data = "https://chat.openai.com/c/65959e07-ec88-42d5-b45c-9dee42d3b03a.com # Data to encode in the QR code"

print("\n\n\tSIMPLE Q R - C O D E G E N E R A T O R\n\n")

# Generate a single QR code

generate\_qr\_code(data)

print("\n\n\tBATCH Q R - C O D E G E N E R A T O R\n\n")

# Generate multiple QR codes

data\_list = ["Data1", "Data2", "Data3"]

batch\_generate\_qr\_codes(data\_list)

print("\n\n\tDECODED Q R - C O D E G E N E R A T O R\n\n")

# Decode QR code

decode\_qr\_code('qr\_code.png')

print("\n\n\tLOGO EMBEDDED Q R - C O D E G E N E R A T O R\n\n")

# Embed a logo into the QR code

embed\_logo\_into\_qr\_code(data, 'logo.png')

**APPENDIX:**

Source code @github: https://github.com/SELVAENGINER/GEN-AI-