**Heritage - إرث**

**Reviving the Voices of Arabic Heritage**

"Heritage - إرث" is an innovative character simulation system that brings the voices of Arabic culture and history to life. By combining state-of-the-art technologies in OCR, text processing, and advanced Arabic language modeling, "Heritage - إرث" allows users to interact with simulated characters that embody the rich linguistic and cultural heritage of the Arabic world. This project aims to preserve, celebrate, and explore the Arabic language through realistic and contextually accurate dialogues, making it a powerful tool for education, entertainment, and cultural preservation.

**Contents**

[**Introduction** 3](#_Toc177356700)

[**Project Workflow Overview** 4](#_Toc177356701)

[**Data Collection** 5](#_Toc177356702)

[**OCR (Optical Character Recognition)** 8](#_Toc177356703)

[**Text Processing** 10](#_Toc177356704)

[**Spelling Correction and Parsing (Advanced Models)** 13](#_Toc177356705)

[**Spelling Correction:** 13](#_Toc177356706)

[**Parsing (Transforming Unstructured Text into Structured Data):** 13](#_Toc177356707)

[**Embedding** 16](#_Toc177356708)

[**Vector Database** 19](#_Toc177356709)

[**Prompt Engineering with Allam** 23](#_Toc177356710)

[**Retrieval-Augmented Generation (RAG) with Allam** 26](#_Toc177356711)

[**API** 30](#_Toc177356712)

[**User Interface (UI)** 34](#_Toc177356713)

[**Conclusion** 38](#_Toc177356714)

## **Introduction**

The aim of this project is to develop an advanced character simulation system in Arabic, capable of mimicking human-like interactions, dialogues, and responses. Character simulation involves creating digital representations of personalities or historical figures that can engage in realistic and contextually accurate conversations. This project is specifically tailored for the Arabic language, addressing its unique linguistic complexities such as script variations, morphological richness, and contextual nuances.

To achieve this, the project integrates several cutting-edge technologies, starting with Optical Character Recognition (OCR) to convert scanned or handwritten Arabic text into machine-readable format. This is followed by text processing steps, including normalization, tokenization, and grammar correction, to enhance the quality and readability of the text.

Advanced models like Allam are utilized for spelling correction, parsing, and enhancing sentence structure, ensuring the text accurately reflects natural language use. Embedding techniques are employed to transform the processed text into numerical vectors, which are then stored in a vector database to facilitate quick and efficient retrieval.

A key component of this simulation is the use of Prompt Engineering and Retrieval-Augmented Generation (RAG). By using Allam, the system retrieves relevant information and generates responses that are contextually appropriate and linguistically accurate. This combination allows the simulated character to engage in dynamic and context-aware conversations, closely resembling human interaction.

Finally, the project includes the development of an API and a user interface, allowing easy integration and interaction with the character simulation system. The API handles the backend processing, while the UI, built with tools like Streamlit or Gradio, provides an interactive platform for users to engage with the simulated character.

This comprehensive approach ensures that the character simulation not only meets the linguistic and contextual demands of the Arabic language but also delivers an engaging and realistic user experience, making it a powerful tool for education, entertainment, and cultural preservation.

## **Project Workflow Overview**

|  |  |  |
| --- | --- | --- |
| Step | Description | Tools/Models |
| |  | | --- | | 1. **Data Collection** | | Gather comprehensive and accurate historical data from trusted sources for simulations. | King Abdulaziz Foundation, Qatar Digital Library, Web Scraping, API Integration |
| |  | | --- | | 1. **OCR (Optical Character Recognition)** |  |  | | --- | |  | | Convert scanned documents, images, or handwritten Arabic text into machine-readable text. | Tesseract OCR, Google Cloud Vision API, Microsoft Azure Computer Vision |
| 1. Text Processing | Clean, normalize, and prepare the OCR text for further processing. | Farasa, CAMeL Tools, NLTK |
| 1. Spelling Correction and Parsing | Correct spelling errors and enhance sentence structure using advanced models tailored for Arabic. | Allam, Cohere, GPT-4 Mini |
| 1. Embedding | Convert processed Arabic text into numerical vectors for further analysis and simulation. | AraBERT, Nomic-Embed-Text, Hugging Face Transformers |
| 1. Vector Database | Store and manage embeddings for fast retrieval during character simulation. | Qdrant, Faiss, Milvus |
| 1. Prompt Engineering | Craft specific inputs (prompts) to guide the model in generating contextually accurate outputs. | Allam |
| 1. Retrieval-Augmented Generation (RAG) | Combine retrieval of relevant information with generation tasks for contextual responses. | Allam, Qdrant, Faiss, Haystack |
| 1. API Development | Build an interface to integrate the simulation into other systems or applications. | FastAPI, Flask |
| 1. User Interface (UI) | Design an interactive dashboard to showcase and interact with the simulated characters. | Streamlit, Gradio |

## **Data Collection**

Data collection is a fundamental step in the "Heritage - إرث" project, serving as the backbone for building accurate and contextually rich simulations of historical figures and events. Collecting high-quality data is crucial for ensuring that the information presented is reliable, authentic, and reflective of the rich history and culture of the Arabic-speaking world. In "Heritage - إرث," data collection focuses on sourcing textual data from historical documents, manuscripts, books, scholarly articles, and trusted digital archives, particularly those relevant to figures like "محمد بن سعود" and other notable personalities.

**Purpose:**

* The goal of data collection is to gather comprehensive and accurate information that can be used to train models, populate vector databases, and enhance the overall simulation experience. This data serves as the foundation for generating responses, providing historical insights, and ensuring the authenticity of the simulations.

**Steps in the Data Collection Process:**

1. **Identify Data Sources:**
   * **Process:** The first step is to identify reputable and relevant sources of historical data, focusing on content that is rich in detail and authenticity. This includes libraries, digital archives, academic publications, and online repositories that specialize in Arabic history and culture.
   * **Example Sources:**
     + **King Abdulaziz Foundation for Research and Archives (Darah):** Provides access to historical manuscripts, documents, and books related to Saudi history.
     + **Qatar Digital Library:** Contains digitized records, manuscripts, and texts from the Arab world, including historical accounts of prominent figures.
     + **Arabic Wikipedia and Scholarly Articles:** Can be used for supplementary information but must be cross-verified for accuracy.
2. **Data Collection Techniques:**
   * **Manual Collection:** Sourcing documents, books, and manuscripts manually from libraries and archives. This approach ensures the quality of the data but can be time-consuming.
   * **Web Scraping:** Automated extraction of data from trusted websites and digital archives, ensuring that data is collected systematically and efficiently.
   * **API Integration:** Utilizing APIs provided by digital libraries and archives to access large volumes of data programmatically, ensuring consistency and ease of data management.
3. **Data Cleaning and Preprocessing:**
   * **Process:** After collecting the raw data, it undergoes cleaning and preprocessing to remove noise, irrelevant content, and formatting errors. This step ensures that the data is ready for analysis and embedding.
   * **Techniques:**
     + Removing HTML tags, metadata, and unwanted symbols.
     + Normalizing text to standardize variations in spelling, especially for historical Arabic names and places.
     + Handling OCR errors if the data is extracted from scanned documents.
4. **Data Annotation and Structuring:**
   * **Process:** Structuring the data into usable formats such as JSON or CSV, tagging relevant information such as names, dates, locations, and significant events. Annotating the data helps in training the language models and embedding systems accurately.
   * **Example:** A historical document about "محمد بن سعود" might be annotated with tags for his name, birth date, significant achievements, and locations associated with his life.
5. **Quality Assurance:**
   * **Process:** Conducting a thorough review of the collected data to ensure its accuracy, relevance, and authenticity. Cross-referencing the information against multiple sources helps validate the data.
   * **Techniques:** Peer review, expert validation, and automated checks for consistency and accuracy are used to maintain high data quality.

**Scenario: Collecting Data on "محمد بن سعود"**

1. **Source Identification:**
   * Historical manuscripts from the King Abdulaziz Foundation and relevant academic articles.
2. **Data Extraction:**
   * Web scraping tools are used to gather text from verified digital archives.
3. **Cleaning and Structuring:**
   * Text about "محمد بن سعود" is cleaned, removing any OCR errors, normalizing Arabic script variations, and structuring into JSON format.
4. **Annotation:**
   * Key information is tagged, such as "name": "محمد بن سعود", "role": "مؤسس الدولة السعودية الأولى".
5. **Validation:**
   * Data is cross-checked against multiple sources to ensure historical accuracy.

**Benefits of Effective Data Collection:**

* **Accuracy and Authenticity:** High-quality data ensures that the simulations are reliable and reflect historical truths, enhancing user trust and engagement.
* **Enhanced Model Performance:** Clean, annotated data improves the training and accuracy of language models like Allam, leading to better responses and simulations.
* **Comprehensive Content:** Diverse data sources provide a rich tapestry of information that allows for detailed simulations and interactive experiences, deepening the user’s understanding of Arabic heritage.

**References:**

* **King Abdulaziz Foundation for Research and Archives (Darah):** [Darah](https://www.darah.org.sa/)
* **Qatar Digital Library:** Qatar Digital Library
* **Arabic Wikipedia:** [Arabic Wikipedia](https://ar.wikipedia.org/)
* **Google Books and Academic Articles:** Trusted sources for supplementary historical information.

## **OCR (Optical Character Recognition)**

Optical Character Recognition (OCR) is a technology used to convert different types of documents, such as scanned paper documents, PDF files, or images captured by a digital camera, into editable and searchable data. In the context of the "Heritage - إرث" project, OCR plays a crucial role in transforming Arabic texts, whether handwritten, printed, or digitized, into machine-readable text that can be further processed and analyzed.

**How It Works:** OCR technology works by analyzing the patterns of light and dark pixels in an image and then identifying the shapes of individual characters based on predefined rules or trained models. Here’s how the process generally works:

1. **Image Preprocessing:**
   * The image is first preprocessed to enhance the quality of the text for OCR. This includes steps like resizing, contrast adjustment, noise reduction, and binarization (converting the image to black and white).
   * **Example:** If we have a scanned document with faded ink, the preprocessing might involve increasing contrast and reducing noise to make the text more distinguishable.
2. **Text Detection:**
   * The OCR software detects areas of the image that contain text. This involves segmenting the image into sections that likely contain characters, such as lines of text or individual words.
   * **Example:** For an image of a historical manuscript, the OCR system will identify the lines and word blocks, distinguishing them from non-text elements like illustrations.
3. **Character Recognition:**
   * Once the text regions are identified, the OCR engine attempts to recognize each character. This is typically done using pattern recognition or machine learning techniques.
   * **Pattern Recognition:** Compares the shapes of characters to a database of known character shapes.
   * **Machine Learning:** Uses trained models, such as neural networks, to recognize characters based on patterns learned from large datasets.
   * **Example:** The OCR software identifies an Arabic word, such as "تراث" (heritage), by matching the shapes of the characters against its database or using a trained model.
4. **Post-Processing:**
   * After recognizing the characters, the OCR engine applies various post-processing techniques to correct errors and improve accuracy. This might include spelling correction, contextual analysis, and formatting.
   * **Example:** If the OCR mistakenly reads "ترث" instead of "تراث," post-processing algorithms might correct this based on the context of surrounding words or a dictionary lookup.
5. **Output:**
   * The final output is a machine-readable text that can be edited, searched, and processed further. The text can be exported to various formats such as plain text, Word, or PDF.

**Example Application in "Heritage - إرث":** Suppose we have a scanned image of an ancient Arabic manuscript. The OCR process would involve preprocessing the image to enhance text visibility, detecting lines of text, recognizing the individual Arabic characters, and then producing a clean, searchable text version of the manuscript. This text can then be used in further stages of the project, such as text processing, embedding, and simulation.

**Tools and Libraries:**

* **Tesseract OCR**: An open-source OCR engine that supports Arabic and many other languages. Tesseract can be trained on custom datasets to improve accuracy for specific types of documents.
* **Google Cloud Vision API**: A powerful OCR service provided by Google that offers high accuracy and is capable of processing complex images with Arabic text.
* **Microsoft Azure Computer Vision**: Another robust OCR tool with specific language support for Arabic, offering cloud-based OCR services with easy integration into applications.

**References:**

* **Tesseract OCR Documentation**: [Tesseract GitHub](https://github.com/tesseract-ocr/tesseract)
* **Google Cloud Vision OCR**: [Google Cloud Vision OCR Documentation](https://cloud.google.com/use-cases/ocr?gad_source=1&gclid=Cj0KCQjwi5q3BhCiARIsAJCfuZl2UhO-0OBxJHtg43SZB-bIxJYd6PRB6wScAqktbrX3R9ySqL-zpNgaApfPEALw_wcB&gclsrc=aw.ds)
* **Microsoft Azure OCR**: [Azure Computer Vision Documentation](https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/)

## **Text Processing**

Text processing is a crucial step in the "Heritage - إرث" project, following OCR to ensure that the recognized text is clean, standardized, and ready for further analysis and simulation. This stage involves a series of techniques to improve the quality, readability, and usability of the text, particularly important when dealing with the complexities of the Arabic language, such as diacritics, different character forms, and various writing styles.

1. **Normalization:**
   * **Purpose:** To standardize the text by removing inconsistencies such as diacritics (harakat) and normalizing various forms of the same character.
   * **Process:** Arabic text often contains diacritics that indicate vowel sounds but are not necessary for meaning in most processing tasks. Normalization involves converting different forms of the same letter (e.g., converting "أ," "إ," and "آ" to "ا").
   * **Example:** The word "الكتابة" (writing) might appear with various diacritics or character forms. Normalization will convert it to a standard form like "الكتابه."
   * **Tools:** Libraries like Farasa and CAMeL Tools provide specific functions for Arabic text normalization.
2. **Tokenization:**
   * **Purpose:** To split the text into smaller units, such as words or sentences, which makes it easier to analyze and process.
   * **Process:** Tokenization involves identifying boundaries between words and sentences. In Arabic, this can be challenging due to the script's lack of clear word separators and the use of clitics (attached particles).
   * **Example:** A sentence like "مرحباً بكم في إرث" (Welcome to Heritage) would be split into individual tokens: ["مرحباً", "بكم", "في", "إرث"].
   * **Tools:** Farasa, CAMeL Tools, and NLTK are popular for their advanced tokenization capabilities tailored for Arabic.
3. **Stopword Removal:**
   * **Purpose:** To remove common Arabic stopwords (e.g., "من," "في," "على") that do not contribute significant meaning to the text analysis.
   * **Process:** Stopwords are filtered out using predefined lists, enhancing the quality of text processing by focusing on meaningful words.
   * **Example:** In the sentence "هذا الكتاب في المكتبة" (This book is in the library), "في" (in) would be identified and removed as a stopword.
   * **Tools:** Predefined stopword lists from libraries like NLTK or custom lists specific to our project's needs.
4. **Lemmatization and Stemming:**
   * **Purpose:** To reduce words to their base or root forms, simplifying analysis by treating different morphological forms of a word as a single item.
   * **Process:**
     + **Lemmatization** identifies the base form of a word based on its meaning and context.
     + **Stemming** reduces words to their root form, often by removing common prefixes and suffixes.
   * **Example:** The word "كتابات" (writings) would be reduced to its lemma "كتابة" (writing) or its root "كتب" (write).
   * **Tools:** Farasa offers a lemmatizer and stemmer specifically designed for Arabic, providing accurate results in morphological analysis.

**Example Application in "Heritage - إرث":** Suppose we have an OCR output from an ancient manuscript that reads "قد كتب المؤرخ قصته بحروفٍ مزخرفةٍ." Text processing will first normalize the text to a standard form, tokenize the sentence into individual words, remove stopwords like "قد" and "بحروفٍ," and then apply lemmatization to get the base forms of the remaining words. This processed text is now cleaner, more concise, and ready for embedding and further analysis.

**Tools and Libraries:**

* **Farasa**: A comprehensive tool for Arabic text processing, including tokenization, stemming, and lemmatization.
* **CAMeL Tools**: A suite of tools designed for Arabic NLP, providing advanced functions for normalization, tokenization, and more.
* **NLTK (Natural Language Toolkit)**: While primarily designed for English, NLTK supports Arabic through custom tokenizers and stopword lists.

**References:**

* **Farasa Website**: [Farasa](https://farasa.qcri.org/)
* **CAMeL Tools GitHub**: [CAMeL Tools](https://github.com/CAMeL-Lab/camel_tools)
* **NLTK Documentation**: [NLTK](https://www.nltk.org/)

## **Spelling Correction and Parsing (Advanced Models)**

Spelling correction and parsing are pivotal stages in the "Heritage - إرث" project, ensuring that the extracted text is both accurate and structured, enhancing its usability for simulation and analysis. The advanced models utilized, such as Allam, Cohere, and GPT-4 Mini, are specifically tailored to handle the complexities of Arabic language processing, including correcting spelling errors and transforming unstructured text into structured formats like JSON.

### **Spelling Correction:**

* **Purpose:** The primary aim is to detect and correct spelling mistakes in the Arabic text, which often result from OCR inaccuracies. This step ensures that the text is coherent, grammatically correct, and contextually accurate.
* **Process:**
  + **Error Detection:** The model scans the text to identify likely spelling mistakes by comparing the detected words against a comprehensive vocabulary or learned language patterns.
  + **Correction Suggestion:** Based on the context of the sentence, the model suggests the most appropriate corrections, enhancing overall readability and accuracy.
* **Example:**
  + **Input Text:** "محمد بن سعود ولد سنه ١٦٩٧ في الدرعيه وهو مؤسص الدوله السغودىه الاولي وحاكم الدرعيه."
  + **Corrected Output:** "محمد بن سعود ولد سنة ١٦٩٧ في الدرعية وهو مؤسس الدولة السعودية الأولى وحاكم الدرعية."

### **Parsing (Transforming Unstructured Text into Structured Data):**

* **Purpose:** Parsing involves converting unstructured text into a structured format like JSON, making the information easy to manipulate, integrate, and utilize within the simulation system.
* **Process:**
  + **Text Analysis:** The model identifies key entities within the text, such as names, dates, locations, and roles.
  + **Data Structuring:** The identified information is organized into a structured format, enabling efficient data management and retrieval.
* **Example Application in "Heritage - إرث":**
  + **Corrected Text:** "محمد بن سعود ولد سنة ١٦٩٧ في الدرعية وهو مؤسس الدولة السعودية الأولى وحاكم الدرعية."
  + **Parsed JSON Output:**

{

"name": "محمد بن سعود",

"birth\_year": 1697,

"birth\_place": "الدرعية",

"title": "مؤسس الدولة السعودية الأولى",

"position": "حاكم الدرعية"

}

**Advanced Models Used:**

1. **Allam:**
   * Allam is a cutting-edge Arabic language model designed for spelling correction and parsing tasks. It effectively handles the nuances of Arabic morphology and syntax, making it highly accurate in correcting text and extracting structured data.
2. **Cohere:**
   * Cohere provides advanced NLP tools for grammar correction, spelling checks, and semantic parsing. Although not exclusively Arabic-focused, it delivers high-quality text refinement through context-aware processing.
3. **GPT-4 Mini:**
   * This model is a versatile, lightweight version of GPT-4, capable of fine-tuning on Arabic data for tasks such as spelling correction and structured parsing. It excels in context-sensitive corrections, making it highly adaptable.

**Implementation Steps:**

1. **Model Selection:** Choose the model based on the specific requirements of spelling correction and parsing. Allam is often preferred for its Arabic specialization.
2. **Integration:** Integrate the chosen model into the text processing pipeline. The model automatically identifies and corrects errors and parses text into structured formats like JSON.
3. **Evaluation:** Continuously evaluate the corrected and parsed outputs to ensure high accuracy and alignment with project objectives.

**Benefits for "Heritage - إرث":**

* **Improved Data Accuracy:** Ensures that the corrected text accurately reflects historical content, enhancing the authenticity of character simulations.
* **Efficient Data Management:** Structured data formats like JSON facilitate easy storage, retrieval, and manipulation, streamlining the overall workflow.
* **Enhanced User Experience:** Clean and well-structured data improve the quality of character interactions, making the simulation more engaging and informative.

**References:**

* **Allam Documentation**: Specialized tools and models for Arabic language correction and parsing.
* **Cohere**: Advanced language processing tools that enhance text coherence.
* **GPT-4 Mini**: Available on platforms like Hugging Face for fine-tuning on specific language tasks.

## **Embedding**

Embedding is a crucial step in the "Heritage - إرث" project, as it transforms processed Arabic text into numerical representations (vectors) that can be used for further analysis, simulation, and retrieval tasks. Embeddings capture the semantic meaning and contextual nuances of text, allowing machine learning models to understand and work with language in a quantitative form. This process is especially important for Arabic text, given its rich morphology, contextual dependencies, and syntactic complexities.

**Purpose:**

* The primary goal of embedding is to convert text into vectors that represent the meaning of words, phrases, or sentences in a multi-dimensional space. These vectors enable tasks such as similarity search, clustering, classification, and input into more complex models like Retrieval-Augmented Generation (RAG).

**How It Works:**

1. **Tokenization:**
   * **Process:** The text is first tokenized into smaller units (tokens), which could be words or sub-words, depending on the embedding model used.
   * **Example:** For the phrase "محمد بن سعود," the text would be split into tokens: ["محمد", "بن", "سعود"].
2. **Embedding Generation:**
   * **Process:** The tokens are passed through an embedding model, which converts each token into a vector of numbers. These vectors capture both the syntactic and semantic properties of the tokens.
   * **Example:** The token "محمد" might be represented as a vector like [0.12, -0.34, 0.56, ...] in a high-dimensional space, reflecting its meaning and context.
3. **Vector Composition:**
   * **Process:** For sentences or larger chunks of text, individual token embeddings are combined to form a single vector that represents the entire input. This composition can be done through averaging, concatenation, or more complex neural network layers.
   * **Example:** The phrase "محمد بن سعود ولد سنة ١٦٩٧" would be represented by a vector that encapsulates the contextual relationship between all tokens.

**Tools and Models Used for Embedding:**

1. **Sentence Transformers (e.g., AraBERT):**
   * **Description:** AraBERT is a transformer model specifically pre-trained on Arabic text. It generates contextual embeddings that capture the nuances of the Arabic language, including morphological variations and sentence-level meaning.
   * **Application:** Best suited for tasks requiring deep semantic understanding of Arabic, such as similarity searches, classification, and text generation.
   * **Example Output:** Given the input "محمد بن سعود," AraBERT generates a high-dimensional vector reflecting the historical and linguistic context of the name.
2. **Nomic-Embed-Text:**
   * **Description:** Nomic-Embed-Text offers robust embeddings for Arabic, providing high-quality representations that are suitable for various NLP tasks. It can handle the complexities of Arabic script and produce vectors that effectively capture meaning.
   * **Application:** Suitable for embedding tasks where accurate semantic representation is crucial, such as entity recognition or search engine indexing.
   * **Example:** Embeds sentences about historical events related to "محمد بن سعود" to improve search and retrieval accuracy.
3. **Hugging Face Transformers:**
   * **Description:** A broad collection of pre-trained models, including many specifically tuned for Arabic, available through the Hugging Face platform. These models provide easy access to high-quality embeddings with options for fine-tuning.
   * **Application:** Used for embedding large corpora of text or integrating embeddings into downstream tasks such as question answering or dialogue generation.
   * **Example:** Uses the "camelbert" model to embed paragraphs describing "محمد بن سعود," capturing both linguistic and contextual features.

**Example Application in "Heritage - إرث":**

Suppose the text about "محمد بن سعود" has been processed and corrected. Embedding this text would involve passing the cleaned data through a model like AraBERT, which would generate vectors representing each part of the text.

**Sample Text:**

محمد بن سعود ولد سنة ١٦٩٧ في الدرعية وهو مؤسس الدولة السعودية الأولى وحاكم الدرعية.

**Embedding Output (Simplified Representation):**

{

"sentence\_embedding": [0.45, -0.12, 0.34, ..., 0.89],

"token\_embeddings": {

"محمد": [0.12, -0.34, 0.56, ...],

"بن": [0.01, -0.03, 0.04, ...],

"سعود": [0.20, -0.22, 0.33, ...],

...

}

}

**Benefits of Embedding:**

* **Semantic Understanding:** Embeddings enable the system to understand the contextual meaning of text, making it suitable for advanced tasks like RAG and content generation.
* **Enhanced Retrieval:** Vectorized text can be stored in vector databases, allowing for fast and accurate similarity searches, which is crucial for simulating historical figures in realistic contexts.
* **Improved Analysis:** Embeddings support various downstream analyses, such as clustering similar texts or identifying key themes within large datasets.

**References:**

* **AraBERT on Hugging Face**: AraBERT
* **Nomic-Embed-Text Documentation**: Access via Hugging Face and NLP libraries.
* **Hugging Face Transformers**: Hugging Face Models.

## **Vector Database**

A vector database is a specialized database designed to store and manage high-dimensional vectors generated by embedding models. In the context of the "Heritage - إرث" project, vector databases play a crucial role in efficiently retrieving relevant information by performing similarity searches on embedded text representations. This functionality is essential for tasks such as finding related historical documents, querying information about specific figures like "محمد بن سعود," and supporting the Retrieval-Augmented Generation (RAG) pipeline.

**Purpose:**

* The primary purpose of a vector database is to store embeddings generated from text data and perform quick, accurate searches to find vectors (and their corresponding data) that are most similar to a query vector. This is particularly useful for applications like content recommendation, information retrieval, and interactive simulations.

**How It Works:**

1. **Vector Indexing:**
   * **Process:** When a new embedding is created, it is indexed within the vector database. Indexing involves organizing the vectors in a way that makes similarity searches efficient.
   * **Example:** Embeddings of texts describing "محمد بن سعود" are indexed to allow fast retrieval when similar queries are made.
2. **Similarity Search:**
   * **Process:** When a query is made, the vector database searches for vectors that are closest to the query vector using metrics like cosine similarity or Euclidean distance.
   * **Example:** A query about "مؤسس الدولة السعودية" would retrieve vectors (and their associated texts) that closely match this concept, such as information about "محمد بن سعود."
3. **Retrieval and Integration:**
   * **Process:** The retrieved vectors are then used to fetch the original text data or to generate responses in applications such as RAG. This enables the system to provide contextually relevant information quickly.
   * **Example:** The vector corresponding to the embedding of "حاكم الدرعية" would pull related information about the leadership of "محمد بن سعود."

**Tools and Vector Databases Used:**

1. **Qdrant:**
   * **Description:** Qdrant is an open-source vector database designed for fast similarity search and scalable vector storage. It supports high-dimensional vectors and integrates easily with various machine learning workflows.
   * **Application:** Ideal for managing embeddings in "Heritage - إرث," Qdrant enables efficient retrieval of relevant historical data based on similarity to input queries.
   * **Features:** Supports advanced indexing techniques like HNSW (Hierarchical Navigable Small World) graphs for fast, approximate nearest neighbor searches.
   * **Example Use Case:** Storing embeddings of historical texts and quickly finding the most relevant passages when querying about "محمد بن سعود."
2. **Faiss (Facebook AI Similarity Search):**
   * **Description:** Faiss is a library developed by Facebook AI that is optimized for high-dimensional similarity searches. It provides efficient indexing and searching capabilities, making it suitable for handling large datasets.
   * **Application:** Faiss can be used within "Heritage - إرث" for rapid retrieval of similar texts, particularly when dealing with large volumes of embedded data.
   * **Features:** Offers a variety of indexing options, including flat (exact search), IVF (inverted file), and PQ (product quantization) for trade-offs between speed and accuracy.
   * **Example Use Case:** Indexing the entire collection of texts related to Saudi history and retrieving relevant segments when exploring topics like the establishment of the Saudi state.
3. **Milvus:**
   * **Description:** Milvus is a highly scalable, cloud-native vector database optimized for machine learning applications. It is designed to handle massive datasets, making it a good choice for projects with extensive data needs.
   * **Application:** In "Heritage - إرث," Milvus can manage embeddings for millions of text entries, providing robust search capabilities to support complex queries.
   * **Features:** Supports distributed architecture, which is beneficial for scaling up as the project grows.
   * **Example Use Case:** Handling vast archives of historical documents and performing complex similarity searches to support character simulations.

**Example Application in "Heritage - إرث":**

Suppose embeddings have been generated for a set of texts about historical figures, including "محمد بن سعود." These embeddings are stored in Qdrant, and a user queries information about "مؤسس الدولة السعودية الأولى." The vector database searches for similar vectors and retrieves the most relevant texts.

**Step-by-Step Workflow:**

1. **Indexing:** Embeddings for texts like "محمد بن سعود ولد سنة ١٦٩٧ في الدرعية" are stored in the vector database.
2. **Query:** A search is initiated for "مؤسس الدولة السعودية الأولى."
3. **Similarity Search:** The database identifies vectors that are closest to the query vector.
4. **Retrieval:** The system retrieves the associated texts, allowing the simulation to display accurate and contextually relevant information about "محمد بن سعود."

**Benefits of Using Vector Databases:**

* **Efficiency:** Provides fast and scalable search capabilities, crucial for handling large datasets of embedded text.
* **Accuracy:** Ensures high relevance in retrieved data by leveraging advanced similarity metrics, improving the quality of simulations.
* **Scalability:** Supports growing data needs as more historical texts and embeddings are added to the project.

**References:**

* **Qdrant Documentation**: Qdrant
* **Faiss GitHub**: [Faiss](https://github.com/facebookresearch/faiss)
* **Milvus Documentation**: Milvus

## **Prompt Engineering with Allam**

Prompt engineering is a technique used to design specific inputs (prompts) that guide language models to generate desired outputs. In the context of "Heritage - إرث," prompt engineering with Allam—a specialized Arabic language model—plays a critical role in generating accurate, contextually relevant, and culturally reflective responses. By crafting effective prompts, Allam can simulate realistic character dialogues, provide historical insights, and enhance the overall user interaction experience.

**Purpose:**

* The goal of prompt engineering is to tailor the inputs given to Allam so that the model generates outputs that align with the intended task, whether it’s answering a question, summarizing a historical event, or simulating a dialogue with a historical figure like "محمد بن سعود."

**How It Works:**

1. **Task Definition:**
   * **Process:** Clearly define what we want Allam to achieve, such as providing a historical overview, engaging in dialogue, or answering specific queries.
   * **Example Task:** Generate a brief description of "محمد بن سعود" and his role in founding the first Saudi state.
2. **Designing the Prompt:**
   * **Process:** Create a structured prompt that includes clear instructions and context to guide Allam’s output. This can involve direct questions, instructional commands, or few-shot examples.
   * **Example Prompt:**
     + Direct Question: "من هو محمد بن سعود؟ وما دوره في تأسيس الدولة السعودية الأولى؟"
     + Instructional Prompt: "أكتب وصفًا موجزًا لمحمد بن سعود ودوره في التاريخ السعودي."
3. **Iteration and Testing:**
   * **Process:** Test the initial prompts with Allam, evaluate the quality of the generated outputs, and refine the prompts as needed to improve accuracy, coherence, and relevance.
   * **Example Iteration:** If the response is too general, add more specific instructions like: "أذكر سنة ولادته ومكان حكمه."
4. **Few-Shot Learning (Providing Examples):**
   * **Process:** Include examples within the prompt to guide the model on how to respond, especially for complex or context-specific tasks. This technique can help Allam better understand the type of response expected.
   * **Example Prompt with Few-Shot Learning:**

السؤال: من هو عبد الله بن سعود؟

الإجابة: عبد الله بن سعود كان آخر حاكم للدولة السعودية الأولى، تولى الحكم بعد والده في بداية القرن التاسع عشر

السؤال: من هو محمد بن سعود؟

**Examples of Prompt Engineering with Allam:**

1. **Simulating Dialogue with Historical Figures:**
   * **Task:** Create an interactive dialogue with "محمد بن سعود" that provides insights into his life and achievements.
   * **Prompt:**

أنت تمثل محمد بن سعود، مؤسس الدولة السعودية الأولى. تحدث عن دورك في تأسيس الدولة ولماذا اخترت الدرعية كمكان للحكم

* + **Expected Output:** A simulated response as if it’s from "محمد بن سعود," reflecting his perspective and historical context.

1. **Generating Summaries:**
   * **Task:** Summarize the key events in the life of "محمد بن سعود."
   * **Prompt:**

أكتب ملخصًا لأهم الأحداث في حياة محمد بن سعود، متضمنًا سنة الولادة، مكان الحكم، وأهم إنجازاته

* + **Expected Output:** A concise summary highlighting his birth, governance, and achievements in founding the Saudi state.

1. **Answering Questions:**
   * **Task:** Provide informative answers about "محمد بن سعود" when queried by users.
   * **Prompt:**

السؤال: متى ولد محمد بن سعود وما دوره في التاريخ؟

* + **Expected Output:** An accurate answer detailing his birth year and historical significance.

**Benefits of Prompt Engineering with Allam:**

* **Improved Accuracy:** By carefully crafting prompts, Allam can generate more precise and contextually appropriate responses, minimizing errors.
* **Enhanced Engagement:** Effective prompts enable the simulation to interact dynamically with users, providing realistic and insightful content.
* **Cultural and Linguistic Relevance:** Allam’s capabilities, when guided by well-designed prompts, ensure that the output respects the nuances of Arabic language and culture, making the interaction more authentic.

**Application in "Heritage - إرث":**

Prompt engineering with Allam allows "Heritage - إرث" to bring historical figures to life, engage users in educational dialogues, and provide in-depth historical content in a compelling and interactive manner. For instance, by simulating the responses of "محمد بن سعود," users can explore the historical context of the first Saudi state in a personalized and engaging way.

**References:**

* **Allam Model Documentation**: Available through Arabic NLP research collaborations and platforms.
* **Prompt Engineering Techniques**: Explore latest research on prompt engineering at platforms like ArXiv and Hugging Face.

## **Retrieval-Augmented Generation (RAG) with Allam**

Retrieval-Augmented Generation (RAG) is an advanced approach that combines information retrieval with text generation to produce highly accurate and contextually relevant responses. In the "Heritage - إرث" project, RAG with Allam is used to enhance the simulation of historical figures by retrieving relevant information from a knowledge base and generating responses that are both informative and contextually aligned with the user's query. This approach allows for the integration of vast historical data, enabling Allam to generate responses that are grounded in accurate and relevant facts.

**Purpose:**

* The main purpose of RAG is to provide accurate and detailed responses by augmenting the generative capabilities of Allam with relevant information retrieved from a database. This ensures that the responses are not only coherent but also factually correct and contextually meaningful, especially when simulating dialogues or providing historical insights.

**How It Works:**

1. **Retrieval Step:**
   * **Process:** The system first retrieves relevant information from a pre-indexed vector database (such as Qdrant or Faiss) that contains embedded historical texts, documents, or other related data.
   * **Example:** When a user asks about "محمد بن سعود," the system retrieves relevant passages about his life, governance, and contributions from the vector database.
2. **Generation Step:**
   * **Process:** The retrieved information is then fed into Allam, which uses this context to generate a response that is informative, accurate, and contextually appropriate.
   * **Example:** The retrieved text about "محمد بن سعود" provides Allam with the necessary background to generate a detailed answer that reflects historical facts.
3. **Integration of Retrieval and Generation:**
   * **Process:** The integration of retrieval and generation ensures that the output is not solely dependent on the model's training data but is dynamically updated with the latest and most relevant information from the database.
   * **Example:** If new historical research or data is added to the database, RAG can leverage this to provide updated responses, keeping the simulation relevant and accurate.

**Example of RAG with Allam in "Heritage - إرث"**

1. **User Query:**
   * **Input:** "أخبرني عن محمد بن سعود ودوره في تأسيس الدولة السعودية الأولى."
2. **Retrieval:**
   * **Process:** The system searches the vector database for embeddings related to "محمد بن سعود" and his role in the establishment of the Saudi state.
   * **Retrieved Text:** "محمد بن سعود ولد في الدرعية عام 1697 وهو مؤسس الدولة السعودية الأولى وحاكم الدرعية."
3. **Generation:**
   * **Process:** Allam uses the retrieved text to generate a response that accurately reflects the historical context.
   * **Generated Response:** "محمد بن سعود ولد في الدرعية عام 1697 وكان له دور كبير في تأسيس الدولة السعودية الأولى حيث أصبح حاكم الدرعية وأسس قواعد الحكم في شبه الجزيرة العربية."
4. **Final Output:**
   * The response is presented to the user, enriched with factual information retrieved directly from the database, ensuring a high level of accuracy and relevance.

**Benefits of RAG with Allam:**

* **Enhanced Accuracy:** By leveraging a retrieval step, RAG ensures that the generated responses are grounded in up-to-date and relevant data, reducing the likelihood of inaccuracies.
* **Contextual Relevance:** The integration of retrieval allows Allam to generate responses that are directly relevant to the user’s query, enhancing the simulation’s engagement and authenticity.
* **Scalability:** As new data is added to the database, the system can easily incorporate this information into its responses, making it adaptable to new findings or updated historical research.

**Implementation Steps:**

1. **Set Up Retrieval System:** Use a vector database like Qdrant or Faiss to store and manage embeddings of relevant texts. Index the data to allow fast and accurate similarity searches.
2. **Integrate Retrieval with Allam:** Connect the retrieval component to Allam, ensuring that retrieved texts are seamlessly incorporated into the generative process.
3. **Design RAG Workflow:**
   * **Input:** User’s question or prompt.
   * **Retrieve:** Fetch relevant information from the vector database.
   * **Generate:** Use Allam to produce a response based on the retrieved content.
4. **Evaluate and Refine:** Regularly test the system to ensure outputs are accurate and contextually appropriate. Fine-tune the retrieval parameters and prompts to improve performance.

**Example Workflow in "Heritage - إرث":**

* **Step 1:** A user asks, "كيف أسس محمد بن سعود الدولة السعودية الأولى؟"
* **Step 2:** The retrieval system identifies relevant passages that discuss his role in founding the state.
* **Step 3:** Allam uses these passages to generate a comprehensive response, combining factual retrieval with natural language generation.
* **Step 4:** The final output provides the user with a detailed and historically accurate account of "محمد بن سعود’s" contributions.

**References:**

* **Qdrant Documentation:** Qdrant
* **Faiss GitHub:** [Faiss](https://github.com/facebookresearch/faiss)
* **Allam Model Information:** Accessible through Arabic NLP research collaborations and platforms.

## **API**

An API (Application Programming Interface) acts as a bridge that allows different software components to communicate and interact seamlessly. In the context of the "Heritage - إرث" project, the API is essential for integrating various functionalities, such as accessing the character simulation, retrieving data, and interacting with the user interface. It provides a standardized way for external applications, including web interfaces, mobile apps, and other systems, to connect with the simulation backend.

**Purpose:**

* The primary purpose of the API in "Heritage - إرث" is to facilitate interaction between the user and the underlying simulation system. It handles requests from the user interface, processes these requests, and returns appropriate responses generated by models like Allam.

**How It Works:**

1. **Endpoint Definition:**
   * **Process:** The API defines specific endpoints—URL paths that correspond to different functions of the system. Each endpoint is designed to handle a particular type of request, such as retrieving information, generating text, or interacting with simulated characters.
   * **Example Endpoint:**
     + /api/character: Fetches data about a specific historical figure.
     + /api/generate: Generates a response based on a prompt using Allam.
     + /api/retrieve: Retrieves relevant historical data from the vector database.
2. **Request Handling:**
   * **Process:** When a request is made to an endpoint, the API processes the request, extracts relevant parameters, and directs the request to the appropriate service (e.g., Allam model, vector database).
   * **Example Request:** A user asks, "من هو محمد بن سعود؟" through the UI, which sends a request to /api/generate with the query as a parameter.
3. **Processing and Response Generation:**
   * **Process:** The API coordinates the retrieval of information, processes it with the appropriate model (e.g., Allam for generation), and formats the response to be returned to the user.
   * **Example Processing:** The API retrieves data about "محمد بن سعود," uses Allam to generate a detailed description, and sends the response back to the user.
4. **Response Delivery:**
   * **Process:** The API returns the processed data or generated text to the frontend, where it is displayed to the user. This ensures a seamless interaction between the backend and the user interface.
   * **Example Response:** A structured response detailing "محمد بن سعود's" life, role, and historical significance.

**Tools for Building the API:**

1. **FastAPI:**
   * **Description:** FastAPI is a modern, high-performance web framework for building APIs with Python. It is known for its speed, ease of use, and powerful capabilities, making it ideal for integrating machine learning models like Allam.
   * **Application in "Heritage - إرث":** FastAPI can be used to build the entire API infrastructure, handling everything from simple data retrieval to complex RAG workflows.
   * **Features:** Includes built-in validation, automatic documentation generation, and asynchronous request handling for improved performance.
   * **Example Use Case:** FastAPI handles requests to generate responses based on Allam’s output, manages user inputs, and retrieves relevant data from the vector database.
2. **Flask:**
   * **Description:** Flask is another lightweight web framework for building APIs in Python. It offers flexibility and simplicity, making it a good alternative for smaller or less complex API needs.
   * **Application:** Can be used to quickly set up endpoints for basic interactions with the simulation system.
   * **Features:** Simple to set up, customizable, and has a wide range of plugins for additional functionalities.
   * **Example Use Case:** Flask can manage basic requests such as fetching character data or handling simple generation tasks with Allam.

**Example API Workflow in "Heritage - إرث":**

1. **Step 1: User Interaction**
   * **Scenario:** A user enters a query: "ما دور محمد بن سعود في التاريخ السعودي؟"
   * **Action:** The query is sent from the UI to the API via a POST request to /api/generate.
2. **Step 2: Request Processing**
   * **API Action:** The API processes the query, retrieves relevant information from the vector database, and passes the data to Allam for response generation.
3. **Step 3: Response Generation and Formatting**
   * **API Action:** Allam generates a detailed response about "محمد بن سعود," including his contributions to the Saudi state, and the API formats this response.
4. **Step 4: Response Delivery**
   * **API Action:** The API sends the generated response back to the UI, which displays it to the user in a readable format.

**Sample API Response:**

{

"name": "محمد بن سعود",

"role": "مؤسس الدولة السعودية الأولى",

"description": "محمد بن سعود ولد في الدرعية عام 1697 وكان له دور كبير في تأسيس الدولة السعودية الأولى، حيث وضع أسس الحكم في الجزيرة العربية.",

"historical\_significance": "حكم الدرعية وأسهم في بناء المملكة من خلال توحيد القبائل وتعزيز الأمن."

}

**Benefits of Using an API in "Heritage - إرث":**

* **Scalability:** The API allows for easy scaling of the project, adding new functionalities and endpoints as needed.
* **Integration:** APIs enable seamless integration of various components such as Allam, vector databases, and user interfaces, ensuring smooth data flow.
* **Security and Management:** APIs can be secured and managed with authentication, logging, and rate limiting to ensure robust and reliable interactions.

**References:**

* **FastAPI Documentation:** FastAPI
* **Flask Documentation:** Flask

## **User Interface (UI)**

User Interface (UI) is the interactive front-end of the "Heritage - إرث" project, allowing users to engage directly with the character simulation and explore historical data in a visually appealing and user-friendly manner. A well-designed UI is crucial for enhancing user experience, making the complex functionalities of the project accessible and intuitive. It serves as the point of interaction between the user and the backend processes, such as retrieving data, generating responses, and simulating character dialogues.

**Purpose:**

* The main purpose of the UI is to present the outputs generated by models like Allam in a structured, engaging, and easily navigable format. It enables users to interact with the system through inputs such as questions, commands, or selections, and receive responses that are clear, informative, and contextually relevant.

**How It Works:**

1. **Design and Layout:**
   * **Process:** The UI design involves creating a layout that is visually appealing and logically organized, guiding users through the various functionalities of the simulation.
   * **Elements:**
     + **Input Fields:** Allow users to enter queries or interact with simulated characters.
     + **Output Display:** Presents generated responses, retrieved data, or character dialogues in a clear format.
     + **Navigation Buttons:** Facilitate easy movement between different sections of the interface, such as historical overviews, character profiles, and interactive simulations.
   * **Example Layout:** A clean, minimalist interface with sections for input, interactive character responses, and additional historical context about "محمد بن سعود."
2. **Interaction Workflow:**
   * **Process:** The UI captures user inputs, sends these inputs to the backend API, receives the processed data or responses, and displays them to the user in a coherent format.
   * **Example Interaction:**
     + A user types, "أخبرني عن محمد بن سعود"، in the input field.
     + The UI sends this query to the API, which processes it with Allam and retrieves the relevant data.
     + The response is displayed in a dedicated area, with options for further exploration or follow-up questions.
3. **Dynamic Content Updates:**
   * **Process:** The UI dynamically updates based on user interactions, providing a responsive experience that feels natural and engaging. This includes updating character dialogues, historical timelines, or visual elements such as maps or images.
   * **Example:** If the user asks about the battles of "محمد بن سعود," the UI might display a map highlighting key locations mentioned in the response.

**Tools for Building the User Interface:**

1. **Streamlit:**
   * **Description:** Streamlit is an easy-to-use Python library for creating interactive UIs and dashboards. It is highly suitable for machine learning projects and allows for rapid prototyping and deployment.
   * **Application in "Heritage - إرث":** Streamlit can be used to build the entire UI, integrating input fields, interactive displays, and visualizations in a straightforward way.
   * **Features:** Supports dynamic updates, embedding of multimedia content, and integration with APIs for real-time data processing.
   * **Example Use Case:** Streamlit displays character profiles, handles user inputs, and visually represents Allam’s generated responses, providing a seamless user experience.
2. **Gradio:**
   * **Description:** Gradio is another Python library that allows for quick deployment of interactive UIs for machine learning models. It is designed to make models accessible and interactive, with easy-to-create input and output interfaces.
   * **Application:** Ideal for creating a lightweight UI that can handle simple inputs and display outputs directly, including audio, text, or visual formats.
   * **Features:** Drag-and-drop interface building, support for multiple input types, and integration with backend models.
   * **Example Use Case:** Gradio can be used to create a quick, user-friendly interface where users can engage with "Heritage - إرث" without complex setup.
3. **Custom Web Development (HTML/CSS/JavaScript):**
   * **Description:** For more complex or highly customized UIs, traditional web development techniques using HTML, CSS, and JavaScript can be used. This allows for greater control over design elements, responsiveness, and interactivity.
   * **Application:** Suitable for creating a highly polished UI with advanced animations, custom layouts, and integrated multimedia elements.
   * **Example Use Case:** A fully customized web application that offers interactive storytelling, detailed character simulations, and integrated multimedia content.

**Example UI Workflow in "Heritage - إرث":**

1. **Step 1: User Input**
   * **Scenario:** A user wants to learn more about the historical significance of "محمد بن سعود."
   * **Action:** The user types their question into the input field on the UI.
2. **Step 2: API Interaction**
   * **UI Action:** The UI sends the query to the API, which retrieves data and generates a response using Allam.
3. **Step 3: Display Output**
   * **UI Action:** The UI formats and displays the response in a clear, organized manner, highlighting key points, dates, or locations relevant to "محمد بن سعود."
4. **Step 4: Further Interaction**
   * **UI Action:** The user can click on highlighted elements (e.g., dates or places) to explore additional information or trigger related content.

**Sample UI Layout:**

* **Header:** Title and brief introduction to "Heritage - إرث."
* **Input Section:** Field for typing questions or prompts.
* **Response Display:** Area where responses from Allam are shown.
* **Character Profile:** Visual and text-based profile of the current character (e.g., "محمد بن سعود").
* **Additional Features:** Links to maps, timelines, or further reading.

**Benefits of an Effective UI in "Heritage - إرث":**

* **User Engagement:** A well-designed UI keeps users engaged, allowing them to explore historical content in an interactive and enjoyable way.
* **Accessibility:** Simplifies complex backend processes, making the information accessible to users of all backgrounds.
* **Enhanced Learning Experience:** Provides a visually appealing and educational interface that enriches the user’s understanding of Arabic heritage.

**References:**

* **Streamlit Documentation:** Streamlit
* **Gradio Documentation:** Gradio
* **Web Development Resources:** MDN Web Docs for HTML, CSS, and JavaScript.

## **Conclusion**

"Heritage - إرث" is an innovative project that blends advanced technologies with rich Arabic cultural heritage to create an immersive and interactive experience. By integrating OCR, text processing, advanced models like Allam, embedding techniques, vector databases, prompt engineering, Retrieval-Augmented Generation (RAG), APIs, and a user-friendly interface, "Heritage - إرث" successfully brings historical figures and events to life.

The project’s workflow leverages the power of cutting-edge language models specifically tuned for Arabic, ensuring high accuracy and contextual relevance in character simulations. The use of RAG enhances the authenticity of responses, grounding the generated content in factual and up-to-date information retrieved from a comprehensive knowledge base. This approach not only makes the simulations engaging but also deeply informative, providing users with a valuable educational tool that honors and preserves the Arabic language and its historical significance.

The thoughtfully designed user interface serves as the bridge between complex backend processes and the end-user, making the interaction seamless, intuitive, and enjoyable. Whether through exploring the life of "محمد بن سعود" or engaging with other historical personalities, the UI ensures that users can easily navigate through content, ask questions, and receive meaningful, context-rich responses.

"Heritage - إرث" represents a fusion of tradition and technology, offering a dynamic way to explore Arabic history and culture. It stands as a testament to the importance of preserving linguistic and cultural heritage in the digital age, providing a platform that not only educates but also inspires deeper appreciation for the rich legacy of the Arabic-speaking world. Through continuous refinement and expansion, "Heritage - إرث" aims to remain a pioneering tool in the field of Arabic historical simulations, contributing to the broader mission of cultural preservation and education.