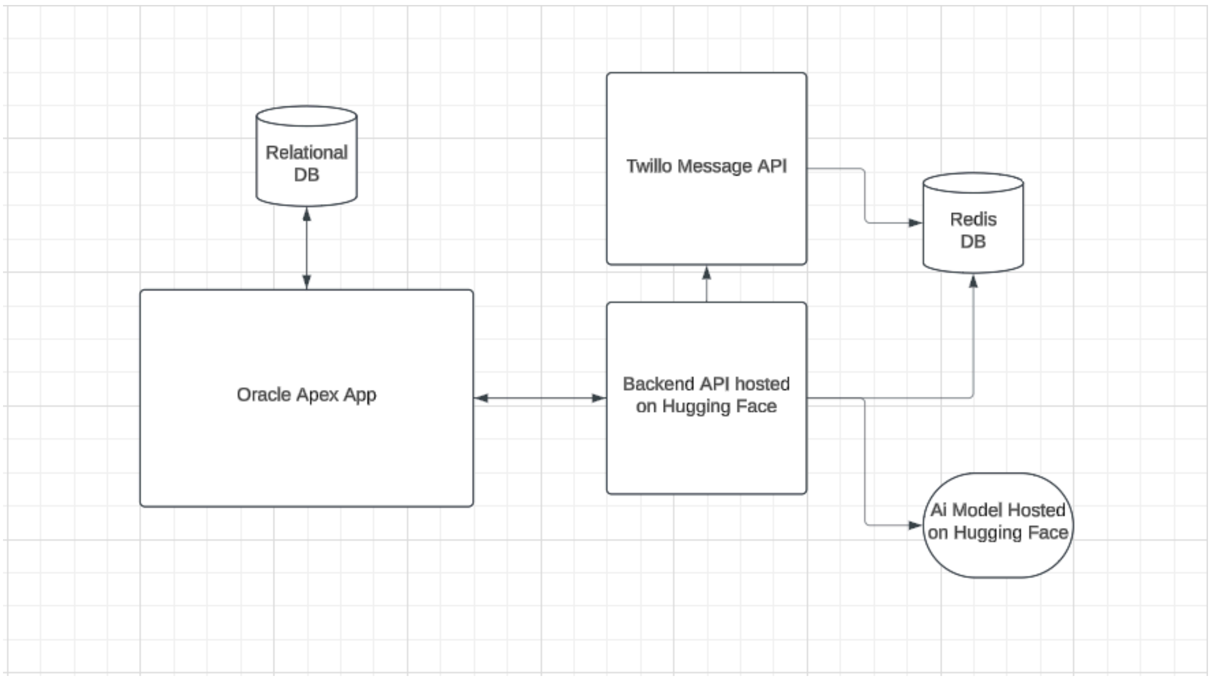
Caregiver – AI-Powered Assistance for Assisted Living Facilities

Caregiver is an innovative application designed to streamline and enhance the management of assisted living facilities. The primary goal of the project is to empower facility administrators by providing advanced data visualization, intelligent insights, and interactive resident engagement tools. By integrating state-of-the-art technologies, Caregiver aims to improve resident care and operational efficiency through the following key functionalities:

System Diagram:


Relational DB
Twillo Message API
Redis DB
Oracle Apex App
Backend API hosted on Hugging Face
Ai Model Hosted on Hugging Face

- **AI-Driven Medical Assistance:** Utilizes large language models (LLMs) via Hugging Face Transformers to answer complex medical queries and provide contextual, accurate information. This allows administrators to ask questions related to general health (e.g., treatment options for cancer, exercises for osteoporosis) as well as facility-specific inquiries (e.g., event details, resident health statistics).
- **Dual-Purpose Redis Database:** Acts as both a key-value store to maintain conversation history (ensuring that the chatbot interactions remain contextually aware) and as a vector database to support efficient retrieval of relevant data based on vector searches.
- **Oracle APEX Frontend:** The user interface is built on Oracle APEX, which is synchronized with a relational SQL database. This setup allows for dynamic data visualizations and an interactive dashboard that administrators can use to monitor resident health and facility operations.
- **Interactive Memory Game:** Provides an engaging cognitive exercise for residents. In this game, an LLM simulates a person from a specific year. Users ask questions to gather hints (e.g., inquiries about popular music or historical events), and then they attempt to guess the hidden year.

- **Twilio SMS Notifications:** Integrates with Twilio's API to send daily SMS notifications. These messages alert administrators to upcoming events and important updates at the facility, ensuring they stay informed in real time.

The GitHub repository is organized into five main subdirectories:

1. **backend-api**
2. **data**
3. **frontend-components**
4. **sql**
5. **vector-database-configurations**

Additionally, the root directory contains supportive configuration files (such as `.gitattributes`, `.gitignore`, and the README) that provide further context and setup details.

Backend API Directory

The **backend-api** directory is the backbone of the Caregiver application. It is hosted on a Hugging Face Spaces server and is responsible for managing all the interactions between the frontend interface, the AI models, and the databases. Here is a detailed breakdown of its components and functionalities:

1. **API and Database Integration:**
    - **Hugging Face Communication:**
      The backend loads a Hugging Face (HF) token that allows it to securely communicate with large language models (LLMs) hosted on the Hugging Face platform. This integration provides access to powerful models for processing natural language queries.
    - **Redis Connection:**
      The application establishes a connection with a Redis database, which serves two critical purposes:
        - **Conversation History Storage:** Maintains a log of past interactions, ensuring that the chatbot can provide contextually aware and coherent responses.
        - **Vector Database:** Supports efficient retrieval of data through vector searches, enabling the system to quickly find relevant information (such as event details and resident statistics) based on user queries.
2. **Medical Assistant Functionality (app.py):**
    - The `app.py` file is the central script within the backend API. It defines a function often referred to as the **Medical Assistant**. This function:
        - Receives and processes medical-related queries.
        - Uses the HF token and connected LLM (with models like TinyZero—uploaded but not deployed due to cost considerations—and Minstrel, a seven-billion-parameter model currently in use) to generate responses.

        - Incorporates both the current prompt and historical conversation context to provide detailed and precise answers, whether the question is about general health (e.g., exercises for osteoporosis) or specific facility information (e.g., event schedules, dental issues).

3. **Interactive Memory Game:**
    - The backend also supports an engaging memory game designed to stimulate cognitive function in residents. Key aspects include:
        - **Game Mechanics:**
          A game session is initiated in which the LLM is tasked with simulating a person from a specific year. Users interact with the game by asking questions (e.g., "Is jazz popular during that time?") to gather hints about the hidden year.
        - **User Engagement:**
          Based on the clues provided by the LLM, the user makes guesses regarding the year. This interactive game not only entertains but also serves as a mental exercise, providing a fun way to enhance memory.
4. **API Endpoints:**
    - The backend API exposes several endpoints to facilitate these functionalities:
        - **`/api/data`:**
          Handles POST requests for the medical chatbot, processing user queries and returning relevant medical or facility-specific information.
        - **`/api/game/start`:**
          Initiates a new memory game session by setting up the game environment and selecting a hidden year.
        - **`/api/game/question`:**
          Accepts questions related to the memory game, allowing users to interact with the simulated historical persona.
        - **`/api/game/guess`:**
          Processes user guesses for the hidden year, determining whether the correct year has been identified.
5. **Event Notifier (event_notifier.py):**
    - This script is designed to integrate with Twilio's API to send daily SMS notifications to facility administrators. Its functionality includes:
        - **Connecting to Redis:**
          The script establishes a connection with the Redis database.
        - **Retrieving Today's Events:**
          It performs a vector search (using a function called `getTodayEvents`) to fetch events scheduled for the current day.
        - **SMS Dispatch:**
          After gathering the relevant event information, the script sends an SMS to the administrator, ensuring they receive timely updates on facility events and other important notifications.

In summary, the backend-api directory is a robust and multifaceted component of the Caregiver project. It seamlessly integrates AI-driven medical assistance, interactive gaming, real-time notifications, and efficient data management to create a responsive and intelligent system that supports both administrators and residents in assisted living facilities.

Data Directory

The **data** directory contains files related to data analysis and visualization. This directory plays a crucial role in transforming raw data into meaningful insights, which are later used to generate SQL-based visualizations in the Oracle APEX frontend.

1. Healthy Aging.ipynb

- **File Name:** `Healthy Aging.ipynb`
- **Purpose:**
    - This Jupyter Notebook (`.ipynb`) file was created by one of the project partners and is dedicated to analyzing data related to healthy aging.
    - It contains Python scripts for **data visualization**, leveraging various libraries (such as Pandas, Matplotlib, and Seaborn) to generate graphs and charts based on the dataset.
    - The goal of this analysis is to extract useful insights related to resident health trends, such as **dental health statistics, disability information, mental distress trends, and aging-related conditions**.
    - These visualizations were later **converted into SQL statements** and integrated into the Oracle APEX frontend for dynamic data representation.

2. environment.yml

- **Purpose:**
    - This file defines the **Python environment configuration** required to run the `Healthy Aging.ipynb` file.
    - It specifies all dependencies, including libraries needed for data processing, visualization, and analysis.
    - This ensures that any team member or developer working on the project can **replicate the same environment** by setting up their Python environment using Conda.

Summary of the Data Directory

The **data directory** serves as the foundation for data-driven decision-making within the Caregiver application. It houses the `Healthy Aging.ipynb` file, which was used to generate insightful **visualizations** that were later transformed into SQL queries for the **Oracle APEX frontend**. Additionally, the `environment.yml` file ensures that the development environment remains consistent across different setups.

Frontend Components Directory

The **frontend-components** directory contains custom-built frontend elements that were integrated into the **Oracle APEX application**. These components improve usability and provide an interactive experience for administrators and residents. This directory has **three subdirectories**, each representing a key frontend feature:

1. Changing Colors Header

- **Purpose:**
  - A **visually engaging and animated header** that changes colors dynamically over time.
  - Designed to enhance the **aesthetic appeal** of the Oracle APEX interface while maintaining responsiveness for mobile devices.
- **Files in this subdirectory:**
  - `index.html`:
    - Defines the **HTML structure** of the animated header.
    - Contains the **title and layout** for the dynamic color effect.
  - `style.css`:
    - Implements **CSS animations** that make the background colors change smoothly over time.
    - Ensures the header is **responsive and visually appealing** across different devices.

2. Medical Chatbot

- **Purpose:**
  - The **Medical Chatbot** allows users (primarily administrators) to ask **medical-related questions** and receive AI-driven responses.
  - It is the main interface that communicates with the **backend API**, fetching **general health information and facility-specific data**.
- **Files in this subdirectory:**
  - `index.html`:
    - Creates the **chat interface**, where users can type questions and receive responses.
  - `index.js`:
    - Implements **JavaScript logic** to handle API requests and responses.
    - Sends user queries to the **backend API** and displays responses in real time.
  - `style.css`:
    - Defines the **styling and layout** for the chatbot.
    - Ensures a **clean and user-friendly design** with properly formatted chat bubbles.

3. Memory Game

- **Purpose:**

    - This component powers the **interactive memory game** designed to stimulate cognitive function in residents.
    - The game challenges players to **guess a historical year** based on clues provided by an **LLM (Large Language Model)**.
    - The AI role-plays as someone living in a specific year, and players must ask questions (e.g., “Was jazz popular then?”) to gather hints before making a guess.
- **Files in this subdirectory:**
    - `index.html`:
        - Defines the **structure** of the game interface.
        - Provides an input field for asking the LLM questions and a submit button for making guesses.
    - `index.js`:
        - Handles **game logic and user interactions**.
        - Sends **player questions** to the backend and retrieves AI-generated responses.
        - Processes user **guesses** to determine if they correctly identified the hidden year.
    - `style.css`:
        - Applies **styling** to the game interface.
        - Ensures **readability and responsiveness**, making the game visually appealing and easy to use.

Summary of the Frontend Components Directory

The **frontend-components** directory contains **three subdirectories**, each representing a unique UI element integrated into **Oracle APEX**:

1. **Changing Colors Header** – A dynamic, animated header that changes colors over time.
2. **Medical Chatbot** – The main interface for administrators to retrieve **medical information** and **facility-specific data** from the AI-powered assistant.
3. **Memory Game** – An interactive **historical guessing game** that engages residents by challenging them to identify a hidden year based on AI-generated responses.

These components ensure **an engaging, interactive, and user-friendly experience** for administrators and residents within the Caregiver application.

SQL Directory

The **SQL directory** contains structured queries used to generate **data visualizations and reports** within the **Oracle APEX application**. These SQL scripts retrieve and process health-related data from the **Healthy Aging dataset**, allowing facility administrators to visualize trends across different demographic factors such as **age, location, gender, and ethnicity**.

Each SQL file corresponds to a **specific page** in the Oracle APEX application, providing insightful **charts and tables** that help administrators monitor the well-being of residents.

SQL Pages and Their Visualizations

1. Dental Information Page

- **Purpose:**
  - This page presents **data on tooth loss due to decay or gum disease** in older adults.
  - The SQL queries retrieve **average percentages** of adults with five or fewer lost teeth across different demographic categories.
- **Visualizations:**
  - **Chart 1:** Average percentage of older adults with five or fewer lost teeth, **by age**.
  - **Chart 2:** Average percentage of older adults with five or fewer lost teeth, **by location**.
  - **Chart 3:** Average percentage of older adults with five or fewer lost teeth, **by gender**.
  - **Chart 4:** Average percentage of older adults with five or fewer lost teeth, **by ethnicity**.
- **SQL File:** `DentalInformationPage.sql`

2. Disability Information Page

- **Purpose:**
  - This page visualizes data on **self-reported disabilities** among older adults, covering sensory, mobility, mental, and emotional impairments.
- **Visualizations:**
  - **Chart 1:** Average percentage of older adults **with a disability, by age**.
  - **Chart 2:** Average percentage of older adults **with a disability, by location**.
  - **Chart 3:** Average percentage of older adults **with a disability, by gender**.
  - **Chart 4:** Average percentage of older adults **with a disability, by ethnicity**.
- **SQL File:** `DisabilityInformationPage.sql`

3. Homepage

- **Purpose:**
  - This page serves as the **main dashboard**, presenting multiple key health metrics in **table format** rather than visual charts.
- **Data Tables:**
  - **Table 1:** Percentage of **adult women** who have received a **mammogram** within the past two years.

    - **Table 2:** Percentage of older adults experiencing **frequent mental distress**.
    - **Table 3:** Percentage of older adults who have **reported a disability**.
    - **Table 4:** Percentage of older adults who report having lost **five or fewer teeth** due to decay or gum disease.
- **SQL File:** `HomePage.sql`

4. Mental Distress Page

- **Purpose:**
    - This page **tracks the prevalence of mental distress** among older adults, helping administrators monitor potential well-being concerns.
- **Visualizations:**
    - **Chart 1:** Average percentage of older adults experiencing **frequent mental distress, by age**.
    - **Chart 2:** Average percentage of older adults experiencing **frequent mental distress, by location**.
    - **Chart 3:** Average percentage of older adults experiencing **frequent mental distress, by gender**.
    - **Chart 4:** Average percentage of older adults experiencing **frequent mental distress, by ethnicity**.
- **SQL File:** `MentalDistressPage.sql`

SQL Query Breakdown

Below is a **summary of how the SQL queries are structured** for retrieving relevant data for each visualization.

Example Query for Mental Distress Data (Grouped by Age)

```
SELECT

    STRATIFICATION1 AS CATEGORY,

    AVG(DATA_VALUE) AS AVERAGE_DATA_VALUE

FROM HEALTHY_AGING

WHERE QUESTION = 'Percentage of older adults who are experiencing frequent mental distress'

AND DATA_VALUE IS NOT NULL

AND STRATIFICATION1 IS NOT NULL

GROUP BY STRATIFICATION1;
```

- This query retrieves the **average percentage** of older adults who experience **frequent mental distress**, grouped by **age categories**.
- The results are used to generate **Chart 1** on the **Mental Distress Page**.

Example Query for Disability Data (Grouped by Gender)

```
SELECT

    STRATIFICATION2 AS CATEGORY,

    AVG(DATA_VALUE) AS AVERAGE_DATA_VALUE

FROM HEALTHY_AGING

WHERE QUESTION = 'Percentage of older adults who report having a disability (includes
limitations related to sensory or mobility impairments or a physical, mental, or emotional
condition)'

AND DATA_VALUE IS NOT NULL

AND STRATIFICATION2 IS NOT NULL

AND STRATIFICATION2 IN ('Male', 'Female')

GROUP BY STRATIFICATION2;
```

- This query retrieves **disability statistics** for older adults, broken down by **gender**.
- The data is used in **Chart 3** on the **Disability Information Page**.

Example Query for Mammogram Screening Data (Homepage Table)

```
SELECT ID,

       STRATIFICATION1,

       LOCATIONDESC,

       DATA_VALUE

FROM HEALTHY_AGING

WHERE QUESTION = 'Percentage of older adult women who have received a mammogram within the
past 2 years'

AND DATA_VALUE IS NOT NULL;
```

- This query retrieves **mammogram screening rates** among **older women**, displaying them in **Table 1** on the **Homepage**.

Summary of the SQL Directory

The **SQL directory** plays a crucial role in **transforming raw health data into meaningful insights** by:

1. **Generating data visualizations** for **dental health, disabilities, and mental distress**.
2. **Presenting key statistics** on the **Homepage** in table format.
3. **Allowing facility administrators** to monitor resident health trends across **age, location, gender, and ethnicity**.

The SQL scripts were derived from **data visualizations created in Python (Jupyter Notebook)** and later **converted into SQL queries** for integration with **Oracle APEX**.

Vector Database Configuration Directory

The **vector_database_configuration_files** directory contains essential Python scripts for **setting up and testing vector search capabilities** within the **Redis database**. These scripts ensure that the backend API can efficiently retrieve relevant information using **vector embeddings**. This functionality enhances the AI-powered chatbot's ability to provide **accurate and contextually relevant responses** to user queries.

---

1. `init_redis.py` (Vector Index Creation & Data Initialization)

- **Purpose:**
    - This script is responsible for **creating and configuring the vector similarity index** within Redis.
    - It ensures that the backend API can **perform efficient searches** for event-related data using **vector embeddings**.
    - It loads event data from an **Excel spreadsheet** and encodes event descriptions into numerical vector representations.
    - The setup allows for **future expansion**, such as incorporating **medical data and additional facility-related insights**.
- **Key Functionalities:**
    - **Load Environment Variables:**
        - Reads configuration values (e.g., Redis credentials, file paths) from a `.env` file.
    - **Connect to Redis Database:**
        - Establishes a secure connection to Redis using **host, port, and authentication credentials**.
    - **Define the Vector Search Schema:**
        - Creates a Redis **vector search index** (`idx:events`) with the following key fields:
            - `name` → Event Name

            - `description` → Event Description
            - `date`, `start_time`, `end_time` → Event Schedule
            - `location` → Event Location
            - `organizer` → Event Organizer
            - `resident_participants` → List of Participants
            - `embedding` → **Vector representation** (for similarity searches)
    - **Generate Vector Embeddings:**
        - Uses **`sentence-transformers/all-MiniLM-L6-v2`** to encode event descriptions into **384-dimensional vector embeddings**.
    - **Store Events in Redis:**
        - Saves event details **along with vector embeddings** in Redis using **hash storage**.
    - **Test Vector Search Functionality:**
        - Runs a **test query** using a sample event name (e.g., "art workshop") to verify that the vector similarity search is **working correctly**.

Example Query:

```
def test_vector_search(test_embedding):

    results = redis_client.execute_command(

        'FT.SEARCH', 'idx:events', '* =>[KNN 3 @embedding $BLOB]',

        'PARAMS', '2', 'BLOB', test_embedding.astype(np.float32).tobytes(),

        'SORTBY', '__vector_score',

        'RETURN', '2', 'name', '__vector_score',

        'LIMIT', '0', '3'

    )
```

-
    - **Performs a nearest-neighbor search (KNN) on event embeddings.**
    - **Finds the top 3 most similar events** based on cosine similarity.

2. `debug_read_redis.py` (Vector Search Testing & Debugging)

- **Purpose:**
    1. This script **verifies the integrity** of the Redis vector search system by performing **test queries**.
    2. It ensures that events are **correctly stored** in the database and that similarity searches return **meaningful results**.
- **Key Functionalities:**
    1. **Connects to Redis** using the stored **environment variables**.
    2. **Performs a Simple Query Test:**

    - Runs a **basic FT.SEARCH** to list all indexed event records.

```
def simple_search():

    results = redis_client.execute_command("FT.SEARCH", "idx:events", "*")
```

3. **Executes a Vector Search Test:**
    - Encodes a **sample query ("art workshop")** into a vector embedding.
    - Runs a **KNN search** against the event embeddings stored in Redis.
    - Retrieves and sorts results based on **vector similarity scores**.

```
def vector_search_test(query_text):

    query_embedding = embed_model.encode(query_text)

    results = redis_client.execute_command(

        'FT.SEARCH', 'idx:events',

        "*=>[KNN 3 @embedding $BLOB AS vector_score]",

        'PARAMS', '2', 'BLOB', query_embedding.astype(np.float32).tobytes(),

        'DIALECT', '2',

        'SORTBY', 'vector_score',

        'RETURN', '2', 'name', 'vector_score',

        'LIMIT', '0', '3'

    )
```

Summary of the Vector Database Configuration Directory

The **vector_database_configuration_files** directory is crucial for enabling **efficient AI-powered searches** in the Caregiver application.

- `init_redis.py`:
    - **Creates the Redis vector index** (`idx:events`).
    - **Encodes and stores event data** as **vector embeddings**.
    - **Ensures AI-assisted search capabilities** for event retrieval.
- `debug_read_redis.py`:
    - **Tests the accuracy of Redis vector searches**.
    - **Ensures data integrity** and confirms that the system is **returning meaningful event matches**.

These scripts lay the foundation for **advanced AI-assisted facility management** by allowing Redis to **store and retrieve critical event data efficiently**. The vector search setup is also **scalable**, enabling future enhancements such as **medical document retrieval, resident history tracking, and AI-powered decision support**.

Hugging Face Model: TinyZero

As part of the **Caregiver project**, we uploaded a model called **TinyZero** to **Hugging Face**. While this model was not used in the **demo application** due to cost constraints, we firmly believe that if this application were deployed in a **production environment**, we would have utilized this model.

Why TinyZero?

- **Cost-Efficient GPU Compute:**
  - TinyZero requires significantly **lower computational resources**, making it an ideal candidate for **production deployment** where cost optimization is critical.
  - If we had used a more computationally expensive model like Minstrel (which we used for the hackathon), the deployment would require much higher **GPU power**, leading to **higher costs**.
- **Pre-trained for Reasoning & Search Abilities:**
  - TinyZero builds upon **DeepSeek R1 Zero**, leveraging **reinforcement learning (RL) techniques** to develop **self-verification and search abilities**.
  - This makes it an excellent fit for **assisting administrators** in retrieving relevant facility information **accurately and efficiently**.
- **Scalability & Adaptability:**
  - The model's lightweight architecture makes it **scalable** for production use.
  - It can be fine-tuned or integrated with **additional training data** to improve its **accuracy** and **domain-specific performance**.

Why Didn't We Deploy It in the Demo?

- Since **Hugging Face Spaces** charges for GPU hosting, deploying TinyZero during the **hackathon** would have required **additional costs** that were **not feasible** for the competition.
- Instead, we used **Minstrel (7B model)** as our AI assistant for the demo, but we still **uploaded TinyZero** to **Hugging Face** as a proof of concept.

TinyZero: Model Overview

TinyZero is a **reproduction of DeepSeek R1 Zero**, optimized for reasoning tasks like **countdown and multiplication problems**. It builds upon **veRL** (volcengine RL framework) and demonstrates the power of **reinforcement learning in AI reasoning tasks**.

- **Developed using RL-based techniques**
- **Optimized for self-verification and search tasks**
- **Designed to run efficiently on consumer-grade GPUs**

TinyZero Installation & Training (For Future Use)

While we did not fine-tune or modify TinyZero, the model can be **trained and deployed** using the following setup:

Installation

```
conda create -n zero python=3.9

pip install torch==2.4.0 --index-url https://download.pytorch.org/whl/cu121

pip3 install vllm==0.6.3

pip3 install ray

pip install -e .

pip3 install flash-attn --no-build-isolation

pip install wandb IPython matplotlib
```

Running Training (Single GPU)

For models **≤1.5B parameters**, training can be done on a single GPU.

```
export N_GPUS=1

export BASE_MODEL={path_to_your_model}

export DATA_DIR={path_to_your_dataset}

export ROLLOUT_TP_SIZE=1

export EXPERIMENT_NAME=countdown-qwen2.5-0.5b

export VLLM_ATTENTION_BACKEND=XFORMERS


bash ./scripts/train_tiny_zero.sh
```

For **larger models (3B+)**, multiple GPUs are required:

```
export N_GPUS=2

export BASE_MODEL={path_to_your_model}

export DATA_DIR={path_to_your_dataset}

export ROLLOUT_TP_SIZE=2

export EXPERIMENT_NAME=countdown-qwen2.5-3b

export VLLM_ATTENTION_BACKEND=XFORMERS


bash ./scripts/train_tiny_zero.sh
```

Future Considerations

If the **Caregiver application** is moved from a **hackathon project to a real-world production system**, we would consider **deploying TinyZero** to **reduce compute costs** while maintaining **strong AI reasoning capabilities**.

Advantages in Production Use:

✅ **Low-cost inference** compared to larger LLMs
✅ **Efficient search & retrieval** using AI reasoning
✅ **Better scalability** for integrating additional domain-specific data

Conclusion

Although we **did not deploy TinyZero** for this hackathon **due to budget constraints**, we firmly believe it would be **the ideal model** for **real-world deployment** in assisted living facilities. The fact that we uploaded it to Hugging Face demonstrates our forward-thinking approach to **cost-efficient AI deployment**.