**USE CASE ANALYSIS**

Based on the content of the research paper, here are insights regarding your concerns about suggesting exits, updating stocks, and allowing users to provide parameters for portfolio creation:

### 1. Suggesting Exits and Updating Stocks

* **Alignment with Research**: The research paper primarily focuses on using LSTM neural networks for predicting stock prices and optimizing portfolios based on those predictions. While it does not explicitly cover exit strategies, the underlying principles of price prediction and portfolio optimization can be adapted to suggest exits based on predicted price movements.
* **Feasibility**: You can implement a feature that suggests exits based on predicted declines in stock prices. This would involve using the same predictive models discussed in the paper to forecast future stock performance. Therefore, while not directly covered, it is feasible to extend the research's findings to include exit strategies.

### 2. Updating Stocks

* **Alignment with Research**: The paper discusses portfolio optimization based on predicted returns, which implies that updating stock holdings based on new predictions is a logical extension. The ability to update stocks aligns with the dynamic nature of portfolio management emphasized in the research.
* **Feasibility**: Implementing stock updates is feasible and aligns well with the research's focus on continuous optimization based on new data.

### 3. User Parameters for Portfolio Creation

* **Alignment with Research**: Allowing users to specify parameters such as the amount to invest and expected return percentages is a practical feature. The research paper discusses mean-variance optimization (MPT), which can incorporate user-defined constraints and objectives.
* **Feasibility**: This feature is feasible and can be implemented by adapting the optimization algorithms discussed in the paper. You would need to ensure that the optimization process can handle user-defined parameters effectively.

### Conclusion

* **Exit Suggestions**: While not explicitly covered in the research, you can feasibly implement this feature by leveraging the predictive models for stock prices.
* **Stock Updates**: This is directly in line with the research and is feasible to implement.
* **User Parameters for Portfolio Creation**: This is also feasible and aligns with the principles of portfolio optimization discussed in the paper.

In summary, you can create a system that includes suggestions for exits and stock updates, as well as allows users to input investment parameters, all while remaining aligned with the research paper's findings. You may need to adapt some methodologies or algorithms to fit these additional functionalities, but they are certainly achievable within the framework established by the research.

**How the project may look from user and developer prospective**

Combining these techniques offers a strong foundation for optimizing stock portfolios based on past data and future predictions. Here's a breakdown of how you can approach it, especially from the **backend (FastAPI), frontend (Streamlit), database**, and **ML/DL model integration** perspectives.

**Backend Architecture (FastAPI)**

FastAPI will handle key components like data fetching, model serving, and user interactions.

**1. LSTM & GRU for Predictive Modeling**

* **Purpose**: These models will predict future stock prices based on historical NEPSE data (open, close, high, low, traded quantity, traded amount, etc.).
* **Backend Requirement**:
  + Use **FastAPI** to expose endpoints for running LSTM/GRU predictions.
  + Models can be trained offline with historical data, and predictions served via REST API when users request portfolio optimization.
  + Store models locally or in a cloud storage (S3, GCS) for real-time inference.
  + You can use **PyTorch** or **TensorFlow** to implement LSTM/GRU models.

**2. Temporal Fusion Transformer (TFT)**

* **Purpose**: TFT can be integrated for more advanced stock price predictions that take into account time-series dynamics, multiple covariates, and future uncertainties.
* **Backend Requirement**:
  + A separate API endpoint for TFT can handle predictions. Depending on the complexity of your application, you could use **TFT** alongside **LSTM/GRU** to compare results.
  + TFT models typically require more advanced tuning and could be trained using libraries like **PyTorch Forecasting** or **TensorFlow**.

**3. Real-Time Data Integration**

* **Purpose**: FastAPI will fetch live NEPSE stock data and historical data using web scraping or external APIs.
* **Backend Requirement**:
  + Use FastAPI to create a scheduler (e.g., using **Celery** or **APScheduler**) that fetches stock data periodically and updates the database.
  + Real-time data can be processed and served to the frontend through REST APIs for real-time predictions and updates.

**4. User Preferences and Constraints**

* **Backend Requirement**:
  + Expose endpoints to accept user input for risk tolerance, investment horizon, and constraints (sector limits, stock limits).
  + Use **SQLAlchemy** or another ORM to store user preferences in a database.

**Frontend (Streamlit)**

**1. Portfolio Customization Features**

* **Customizable Risk Tolerance Levels**:
  + Streamlit can provide sliders or input fields where users can adjust risk tolerance levels. The input values will be sent to FastAPI, which will use these to adjust the weight allocations in the optimization model.
* **Investment Horizon Selection**:
  + Streamlit can offer dropdowns or date range selectors to let users choose their investment horizon. This data will be sent to FastAPI to calculate portfolio returns for different timeframes (short, medium, or long-term).

**2. Scenario Analysis**

* **Purpose**: Users will be able to see how their portfolio performs in different market conditions (bullish, bearish).
* **Frontend Implementation**:
  + Streamlit can provide buttons or dropdowns to select scenarios.
  + Once selected, the user input will trigger FastAPI to run scenario analysis by simulating future market conditions using the prediction models.

**3. Rebalancing Strategies**

* **Frontend Implementation**:
  + Allow users to view portfolio rebalance suggestions based on predicted returns and volatility.
  + Streamlit will display the suggested rebalances using dynamic tables or visualizations.

**4. Performance Tracking and Reporting**

* **Frontend Implementation**:
  + Streamlit will display performance reports comparing the user's portfolio to benchmarks like the NEPSE Index or equally weighted portfolios.
  + Visualize the portfolio performance using line charts, bar graphs, or heatmaps (using libraries like **Altair**, **Plotly**, or **Matplotlib**).

**5. User-Defined Constraints**

* **Frontend Implementation**:
  + Streamlit can offer checkboxes or sliders to set constraints (e.g., sector limits, minimum/maximum stock allocation).
  + These constraints will be passed to FastAPI for integration with the portfolio optimization model (mean-variance optimization).

**Database and Data Storage**

* **Historical and Real-Time Stock Data**:
  + Store past NEPSE data and real-time stock prices in a database (e.g., **PostgreSQL** or **MongoDB**).
  + FastAPI will fetch historical data from external sources and periodically update the database with real-time data.
* **User Data**:
  + **SQLAlchemy** can be used in FastAPI to manage user profiles, portfolio preferences, and saved portfolios.
  + Store user login details securely with hashed passwords using **bcrypt**.
* **Model Results Storage**:
  + Store LSTM/GRU predictions and TFT results in the database for performance tracking and optimization.
  + Results can also be cached using **Redis** for quicker access.

**Machine Learning / Deep Learning Module Integration**

**1. LSTM/GRU Model**

* **Model Training**:
  + Train LSTM or GRU models using historical stock data (open, close, high, low, traded amount, etc.).
  + Use libraries like **PyTorch**, **TensorFlow**, or **Keras** for model development.
  + Store trained models for serving via FastAPI.
* **Model Serving**:
  + FastAPI serves real-time predictions when users request portfolio optimization or scenario analysis.

**2. Temporal Fusion Transformer (TFT)**

* **Model Training**:
  + Train TFT models for more advanced stock predictions, especially when considering multiple covariates like stock volume, external factors, and sentiment analysis.
* **Model Serving**:
  + Like LSTM/GRU, the TFT model will be exposed via FastAPI endpoints for real-time predictions.

**3. Portfolio Optimization with ML Models**

* **Backend (FastAPI)**:
  + Use predicted returns and volatility from LSTM/GRU or TFT to optimize the user's portfolio.
  + Mean-variance optimization can be applied using libraries like **SciPy** or custom implementations.
  + Serve optimized portfolio allocations via API endpoints.

**4. Sentiment Analysis Integration**

* **Purpose**: Enhance predictions by adding sentiment analysis on stock-related news.
* **Backend (FastAPI)**:
  + FastAPI can integrate sentiment analysis models (e.g., using **VADER**, **TextBlob**, or a fine-tuned Transformer model) to analyze stock sentiment from news articles.
  + Sentiment analysis results can be passed as additional features to the TFT model.

**Research and Refinement Features**

**1. Feedback Loop for Model Improvement**

* As users interact with the system and provide feedback on portfolio performance, you can store this feedback for continuous learning and model improvement.

**2. Incorporating Additional Factors**

* In the future, you can integrate more factors into the prediction models, such as macroeconomic indicators, sentiment analysis, and other financial data.

**Conclusion**

Yes, you can absolutely combine the **LSTM/GRU** models for price prediction and **TFT** for more sophisticated time-series modeling. Your **portfolio optimization system** can be built with **Streamlit** and **FastAPI**, with each component handling a specific function. Streamlit will handle the user interface, allowing users to interact with features like risk tolerance, scenario analysis, and rebalancing. FastAPI will serve as the backend, managing real-time data, machine learning models, and the optimization logic. This stack is flexible enough to integrate more advanced ML/DL models and additional user-driven features as your project evolves

Got it! Below is a proposed **flowchart pipeline** for your **Portfolio Optimization Website** based on **NEPSE stock data** and your use of **LSTM**, **GRU**, and potentially **Temporal Fusion Transformers (TFT)** models. This pipeline will take into account the key features and requirements you provided for your Final Year Project (FYP).

**Portfolio Optimization Website Pipeline**

**Input:**

1. **User Inputs**:
   * **Risk Tolerance Level** (e.g., Low, Medium, High).
   * **Investment Horizon** (e.g., Short-term, Medium-term, Long-term).
   * **Constraints** (e.g., Sector limits, specific stock exclusions).
   * **Initial Investment Amount**.
2. **Stock Data**:
   * Historical stock data (columns: published\_date, open, high, low, close, per\_change, traded\_quantity, traded\_amount, status).
   * **Top 100 NEPSE stocks**.
   * **Real-time stock data** from APIs or databases.

**Process:**

1. **Data Preprocessing**:
   * **Data Cleaning**:
     + Handle missing values.
     + Remove outliers.
     + Normalize/Standardize relevant features (e.g., traded\_quantity, traded\_amount).
   * **Feature Engineering**:
     + Generate additional features such as moving averages, volatility indicators, etc.
   * **Train-Test Split**:
     + Split data into training and testing datasets based on time intervals.
2. **Model Selection**:
   * **LSTM/GRU/TFT Models**:
     + Train multiple models (e.g., **LSTM** and **GRU**) on historical data to predict future stock prices or returns.
     + **Temporal Fusion Transformer (TFT)** could also be introduced to incorporate richer temporal patterns and additional covariates (e.g., stock sector data, macroeconomic indicators).
   * **Training**:
     + Train models using time-series forecasting techniques.
     + Evaluate models based on error metrics (e.g., MSE, RMSE, MAE).
     + Fine-tune models to optimize accuracy.
3. **Portfolio Optimization**:
   * **Risk-Return Calculation**:
     + Based on predicted stock returns from models, calculate the expected **return** and **risk (volatility)** of each stock.
   * **Markowitz Modern Portfolio Theory (MPT)**:
     + Optimize the portfolio by adjusting the weights of stocks based on user-defined constraints and **risk tolerance**.
   * **Efficient Frontier**:
     + Use the risk-return trade-off to plot the efficient frontier and suggest an optimal portfolio with maximum return for a given risk level.
4. **Rebalancing Strategies**:
   * Incorporate strategies to **rebalance the portfolio** periodically based on updated predictions.
   * Use model-predicted data to suggest **when to buy/sell** stocks based on the current portfolio performance.
5. **Scenario Analysis**:
   * **Bullish/Bearish Market Simulations**:
     + Simulate how the portfolio would perform under different market scenarios (e.g., Bullish/Bearish conditions).
   * Use LSTM/GRU predictions in combination with **historical market conditions** to predict future market trends.

**Output:**

1. **Optimized Portfolio**:
   * Show the final portfolio allocation (stock weights) based on user inputs, predicted returns, and risk.
   * Display **expected return**, **volatility (risk)**, and **Sharpe ratio** for the portfolio.
   * Compare with benchmarks (e.g., NEPSE Index, equally weighted portfolios).
2. **Visualization**:
   * **Risk-Return Graph** (Efficient Frontier).
   * **Portfolio Composition** (e.g., pie chart showing stock weights).
   * **Scenario Analysis Results** (e.g., graph comparing portfolio performance in bullish vs. bearish markets).
3. **Performance Tracking and Reporting**:
   * Provide users with tools to track **portfolio performance** over time.
   * Generate detailed reports comparing the portfolio to benchmarks like the NEPSE Index.

**Feedback Loop and Future Model Improvement:**

* Allow users to **give feedback** based on their experience and the actual performance of their portfolios.
* **Improve the model** periodically based on user feedback and evolving market conditions.

**Additional Features:**

* **User-Defined Constraints**: Allow users to set constraints (e.g., limit exposure to certain sectors).
* **Real-Time Updates**: Fetch live data to keep the model updated and adapt to current market trends.
* **Sentiment Analysis** (future feature): Use NLP models to incorporate news-based sentiment analysis of stocks.

**Flowchart Representation**

Here is the flowchart of the Portfolio Optimization Pipeline:

**Step 1: Input**

* User Inputs: Risk tolerance, investment horizon, etc.
* Stock Data: Historical NEPSE stock data, real-time data.

**Step 2: Data Preprocessing**

* Clean and preprocess historical stock data.
* Engineer new features (volatility, moving averages, etc.).

**Step 3: Model Selection**

* Train LSTM/GRU/TFT models to predict stock returns.
* Evaluate and fine-tune models using performance metrics.

**Step 4: Portfolio Optimization**

* Calculate expected return and risk.
* Optimize using Markowitz Modern Portfolio Theory (MPT).
* Visualize Efficient Frontier.

**Step 5: Rebalancing Strategies**

* Use predicted data to implement rebalancing strategies (buy/sell suggestions).

**Step 6: Scenario Analysis**

* Simulate portfolio performance under different market conditions (bullish/bearish).

**Step 7: Output**

* Display optimized portfolio allocation (stock weights, expected return, risk).
* Track portfolio performance and generate reports.

**Step 8: Feedback Loop**

* Incorporate user feedback and continuously improve model performance.

**Backend Requirements:**

1. **Data Sources**:
   * API integration for real-time NEPSE data.
   * Database for storing historical data, user preferences, and portfolio performance.
2. **Models**:
   * LSTM/GRU and TFT models implemented using TensorFlow/PyTorch.
   * Portfolio optimization algorithms using scikit-learn or specialized libraries like cvxpy.
3. **Backend Framework**:
   * Use **FastAPI** for serving the ML models and handling user input/output.
   * Integrate database management (e.g., PostgreSQL) for storing stock data, portfolio performance, and user feedback.

**Frontend (User Interface):**

1. **Frontend Framework**:
   * Use **Streamlit** for building a user-friendly interface where users can:
     + Input risk tolerance, investment horizon, etc.
     + View optimized portfolio and scenario analysis results.
     + Track performance over time.
   * Provide real-time graphs and visualizations of the optimized portfolio.

**Database:**

1. **Historical Stock Data**:
   * Store NEPSE historical stock data (open, close, high, low, traded quantity).
   * Real-time data integration via APIs.
2. **User Preferences**:
   * Store user-specific preferences such as risk tolerance, constraints, and investment horizon.
3. **Model Predictions**:
   * Store model predictions and portfolio performance metrics for comparison with benchmarks (NEPSE index).

By integrating LSTM, GRU, and possibly TFT models into this pipeline, your **Portfolio Optimization Website** will offer users dynamic stock recommendations, risk management tools, and personalized investment strategies based on cutting-edge time-series forecasting methods.

**Use of tensorflow for the project**

When focusing specifically on your final year project (FYP) related to portfolio optimization, here are tailored reasons for choosing TensorFlow over PyTorch:

**1. Model Deployment and Scalability**

* + **TensorFlow Serving**: TensorFlow provides robust tools for deploying machine learning models in production environments. If your portfolio optimization model needs to be integrated into a web application or a financial service, TensorFlow Serving allows for easy deployment and management of models at scale.
  + **Scalability**: TensorFlow is designed to handle large datasets and complex models efficiently, which is crucial for portfolio optimization tasks that may involve extensive historical data and multiple assets.

**2. Integration with Financial Libraries**

* + TensorFlow can be easily integrated with various financial libraries and tools that are commonly used in portfolio optimization, such as **QuantLib** or **Zipline**. This can enhance your ability to perform backtesting and risk analysis alongside your model.

**3. Performance Optimization**

* + TensorFlow's static computation graph (in TensorFlow 1.x) allows for optimizations that can improve performance, which is beneficial when training models on large datasets typical in financial applications. Even in TensorFlow 2.x, the performance optimizations are significant, especially for complex models used in portfolio optimization.

**4. Support for Reinforcement Learning**

* + If your portfolio optimization approach involves reinforcement learning (e.g., using algorithms like Proximal Policy Optimization), TensorFlow has strong support for these techniques through libraries like **TF-Agents**. This can be particularly useful if you are exploring dynamic portfolio strategies that adapt to market conditions.

**5. Comprehensive Documentation and Resources**

* + TensorFlow has extensive documentation and a wealth of resources specifically related to financial applications and portfolio optimization. This can be beneficial for understanding best practices and finding relevant examples that align with your project.

**6. Keras for Rapid Prototyping**

* + TensorFlow includes Keras as its high-level API, which allows for rapid prototyping of models. This can be particularly useful in the early stages of your project when you are experimenting with different architectures for predicting stock returns or optimizing portfolios.

**7. Community and Industry Adoption**

* + TensorFlow's widespread adoption in the finance industry means that there are many case studies and examples of successful implementations of portfolio optimization using TensorFlow. This can provide you with insights and inspiration for your own project.

**8. TensorFlow Probability**

* + TensorFlow Probability is a library that extends TensorFlow to support probabilistic reasoning and statistical analysis. This can be particularly useful in portfolio optimization, where understanding the uncertainty and risk associated with asset returns is crucial.

**Conclusion**

In the context of your FYP on portfolio optimization, TensorFlow offers advantages in terms of deployment capabilities, performance optimization, integration with financial tools, and support for advanced techniques like reinforcement learning. These factors make it a strong choice for developing a robust and scalable portfolio optimization model.

i was thinking maybe built lstm model from scrap and if possible finetune tft, is it feasible from tensorflow and wheather is it good idea or my lack of knowledge

Building an LSTM model from scratch and fine-tuning a Temporal Fusion Transformer (TFT) in TensorFlow is indeed feasible, and it can be a good idea depending on your goals and current knowledge level. Here are some considerations to help you decide:

**Feasibility**

* + **Building LSTM from Scratch**:
    - **Learning Opportunity**: Implementing an LSTM from scratch can provide a deep understanding of how LSTMs work, including the underlying mathematics and architecture. This knowledge can be beneficial for debugging and optimizing your models later.
    - **TensorFlow Support**: TensorFlow provides the necessary tools and functions to build LSTM networks. You can use the tf.keras API to create custom layers and models, which simplifies the process.
  + **Fine-tuning TFT**:
    - **Pre-trained Models**: If you can find a pre-trained TFT model or a well-documented implementation, fine-tuning it can save you time and effort. You can adapt the model to your specific dataset and requirements, which is often more efficient than training from scratch.
    - **Complexity**: TFTs are more complex than LSTMs, as they incorporate attention mechanisms and other advanced features. Fine-tuning requires a good understanding of the model architecture and how to adjust hyperparameters effectively.

**Considerations**

* + **Knowledge and Skills**:
    - If you have a solid understanding of deep learning concepts and are comfortable with TensorFlow, building an LSTM from scratch can be a rewarding experience. However, if you're still learning, it might be more efficient to start with existing implementations and gradually build your knowledge.
    - Fine-tuning a TFT may require familiarity with attention mechanisms and the specific architecture of the TFT. If you're not yet comfortable with these concepts, consider studying them first.
  + **Time and Resources**:
    - Building models from scratch can be time-consuming, especially if you're also learning. Ensure you have enough time to complete your project while balancing the learning curve.
    - Fine-tuning a pre-trained model can be quicker and may yield good results without the need for extensive training.
  + **Project Goals**:
    - If your goal is to demonstrate your understanding of deep learning and model building, creating an LSTM from scratch is a great way to showcase your skills.
    - If your primary goal is to achieve the best performance in portfolio optimization, fine-tuning a TFT or using a well-established model may be more effective.

**Conclusion**

In summary, both building an LSTM from scratch and fine-tuning a TFT are feasible in TensorFlow. If you have the time and desire to learn, building an LSTM can deepen your understanding of deep learning. However, if your focus is on achieving strong results in your portfolio optimization project, fine-tuning a TFT may be a more efficient approach. You could also consider a hybrid approach: start with a pre-trained TFT to establish a baseline, then experiment with building your own LSTM to compare results and gain insights.

**Agile for the FYP**

Agile is often considered a highly collaborative and iterative framework, typically suited for teams working together to rapidly deliver value through small, frequent releases. However, it can still be adapted and beneficial for solo projects.

Here's a breakdown of **Agile** methodology and how it can be applied to your portfolio optimization website project using LSTM and GRU, even if you're the only one working on it:

**Agile Methodology Overview**

Agile is centered around breaking down the project into smaller, manageable chunks called **iterations** or **sprints**. Each sprint typically lasts 1-4 weeks, and by the end of each sprint, a working feature or module of the project is completed. Agile emphasizes:

* **Iterative development:** Build in small, incremental cycles.
* **Frequent feedback:** Continuously improve the project based on feedback.
* **Flexibility:** Adjust the project scope or direction based on ongoing discoveries.

**Benefits of Agile for a Solo Developer:**

Even as a solo developer, Agile can help in structuring your work, keeping you focused, and ensuring steady progress. Here’s why:

1. **Continuous Progress:** Breaking tasks into smaller parts helps avoid procrastination and burnout.
2. **Adaptability:** You can quickly shift focus if you realize something is off or if you discover better techniques during the process.
3. **Self-accountability:** Planning sprints and setting goals help you stay on track.

**How to Apply Agile to Your Project**

1. **Backlog Creation:**
   * List all the features and tasks you need to complete for the website, such as:
     + Data scraping and cleaning.
     + Building and training LSTM and GRU models.
     + Designing the website interface.
     + Integrating the models into the website.
     + Creating user authentication and storing user preferences.
     + Testing and deployment.
2. **Sprint Planning:**
   * Divide the work into 1-2 week sprints.
   * Each sprint should have clear goals (e.g., "Implement LSTM model" or "Design front-end for the dashboard").
   * Prioritize tasks: Focus first on setting up your data pipeline, then on the core machine learning models (LSTM/GRU), and later on front-end development.
3. **Daily or Weekly Progress Checks:**
   * Since you're the only one working, you can do personal "stand-ups" by reviewing what you did yesterday, what you plan to do today, and what obstacles might be in your way.
4. **Sprint Reviews:**
   * At the end of each sprint, review what you've achieved. Reflect on what went well and what could be improved for the next sprint. This also helps you adapt if new issues or requirements arise.
5. **Sprint Retrospectives:**
   * After completing a sprint, analyze what worked and what didn’t. Since you're working alone, it’s more about self-reflection to optimize your workflow.
6. **Final Integration and Testing:**
   * Agile allows continuous integration, meaning that after each sprint, you can check how everything fits together (data handling, models, UI, etc.). This reduces the risk of leaving too much work for the end.

**Example of Sprints for Your Project:**

* **Sprint 1: Data Collection & Preprocessing**
  + Scrape data from NEPSE and prepare it for use.
  + Clean and format the data to be fed into the models.
* **Sprint 2: Develop LSTM Model**
  + Build and train an LSTM model for stock price prediction.
  + Test the model’s accuracy and performance.
* **Sprint 3: Develop GRU Model**
  + Build and train a GRU model and compare it to the LSTM.
* **Sprint 4: Frontend Design (Streamlit UI)**
  + Create the user interface for the portfolio optimization website using Streamlit.
* **Sprint 5: Model Integration & Backend Setup**
  + Integrate LSTM and GRU models into the backend using FastAPI.
  + Set up user authentication and database to store user preferences using MySQL.
* **Sprint 6: Testing & Debugging**
  + Test the website, ensure predictions are accurate, and fix any bugs.
* **Sprint 7: Final Deployment**
  + Deploy the website using cloud services and ensure it's accessible to users.

**Is Agile Still the Best for Solo Development?**

Agile can still be useful for your solo project because it brings structure and helps avoid the common pitfalls of solo work, such as losing focus or not delivering on time. However, if you find Agile too formal or cumbersome for your workflow, you can adapt it to be more lightweight, focusing only on the iterative nature and frequent reviews.

**Alternatives:**

If Agile feels too heavy for a solo project, you could also look into **Kanban**, which is a simpler, visual-based method. With Kanban, you maintain a visual board (like Trello or Jira) where tasks move from “To Do” → “In Progress” → “Done.” This way, you can keep track of your work without the full sprint structure.

In conclusion, Agile can definitely work for you as a solo developer and is great for ensuring progress on a complex project like portfolio optimization with machine learning models. The methodology’s iterative approach helps break the project into manageable steps, so you can maintain focus, regularly assess progress, and adapt quickly.

# Methodology

This project will be developed using the Agile methodology. Although Agile is traditionally used in team-based development, it will be adapted for this solo project to ensure systematic progress, flexibility, and continuous improvement throughout the development process.

Agile emphasizes iterative development, where tasks are broken down into manageable units called sprints. Each sprint will focus on achieving specific project goals, ensuring regular review and adaptation to meet changing needs or overcome unforeseen challenges. The project will be completed in several phases, with feedback incorporated at the end of each sprint.

The major phases of the project are outlined below:

## Project Phases and Sprints

The project will be divided into multiple sprints, each focusing on a specific aspect of the portfolio optimization website using LSTM and GRU models. The following sprints have been identified:

**Sprint 1**: Data Collection and Preprocessing

Scrape stock market data for NEPSE Top 100 companies from GitHub and other websites such as NEPSE Alpha, if necessary.

Clean and preprocess the data, including removing inconsistencies and formatting it for use in model training.

**Sprint 2:** Model Development - LSTM

Develop and train an LSTM (Long Short-Term Memory) model for stock price prediction.

Evaluate the model’s performance by testing it on historical stock price data.

Optimize the model based on initial results.

**Sprint 3**: Model Development - GRU

Build and train a GRU (Gated Recurrent Unit) model as an alternative to LSTM.

Compare its performance to the LSTM model in terms of accuracy and efficiency.

Perform hyperparameter tuning for both models to improve prediction performance.

**Sprint 4:** Frontend Design

Use Streamlit to design an intuitive and user-friendly interface where users can interact with the models.

Display model predictions and allow users to customize portfolio optimization inputs.

Design the user experience (UX) to cater to both casual users and financial analysts.

**Sprint 5**: Backend Development & Integration

Develop the backend using FastAPI to handle requests, pass data between the frontend and machine learning models, and manage user interactions.

Implement MySQL as the database to store user information, preferences, and historical data.

Integrate both the LSTM and GRU models with the frontend interface, ensuring real-time updates for stock price forecasts and portfolio optimization.

**Sprint 6:** Testing and Debugging

Conduct extensive testing on both the frontend and backend systems.

Perform unit tests on the LSTM and GRU models, ensuring accurate stock predictions.

Identify and fix any bugs that arise during integration and interaction testing.

**Sprint 7:** Final Deployment

Deploy the website to a cloud platform for public use, leveraging Google Colab for model training and computation.

Ensure that the platform is accessible on multiple devices, with user authentication and data security in place.