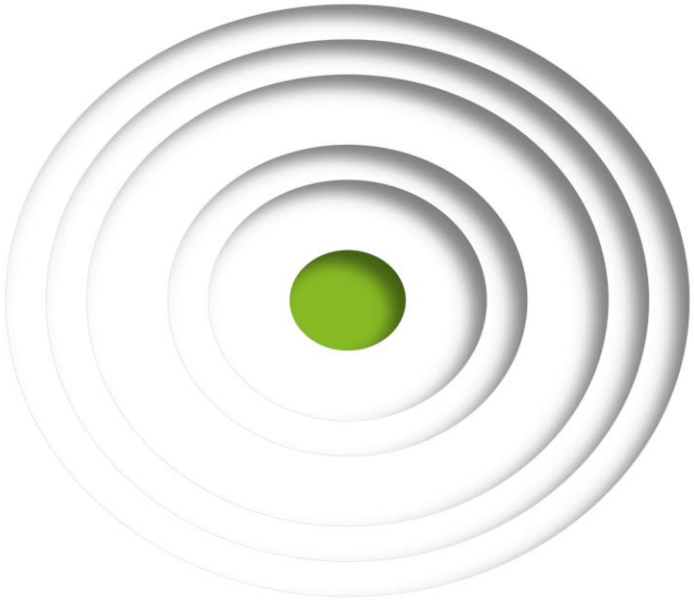
Deloitte.



Agentic Workforce Transformation - Rocksteady

Mariam ElAwadly

Introduction

AI-Powered Software Development: Enhancing the SDLC with PairD & GitHub Copilot

PairD

PairD (pronounced “paired”) is Deloitte’s very own Generative AI (GenAI) platform, designed with your feedback in mind.

Do’s ✓

- Be Clear & Specific : The more detailed your Prompt, the better the response.
- Provide Context : If your question builds on previous knowledge or a specific scenario, include that information in your prompt.
- Specify a role : By assigning a role in your prompts, you provide additional context that shapes how the reply is generated for example : As an expert in the Agile framework, Generate the user stories for this project.
- Iterate on Prompts :Refinement is Essential, If the first response doesn’t meet your needs, refine your prompt and try again. Iteration can help hone in on the exact information you’re seeking.
- Specify the Format of the Response : whether you want the code in multiple python files or in one notebook.

Don’ts ✗

- Use Vague Prompts: Vague questions lead to vague answers, and nobody wants that.
- Don’t Overlook Important Details.

Pros

- Built-in prompts for different use cases (e.g., Agile Storyteller, Coder, Explainer, PowerPoint Creator).
- Allows file uploads (PDFs, images, etc.).
- Users can save custom prompts for future use.
- Useful for critical thinking & ideation, providing recommendations.

Cons

- Output can sometimes be truncated. Solution: Type “Continue” to resume generation.
- Information may be outdated or incorrect (e.g., deprecated libraries or functions).

GitHub Copilot

GitHub Copilot is an AI coding assistant Integrated in your IDE that helps you write code faster and with less effort.

Do's ✓

- Use Copilot for Code Autocompletion: Copilot is good at completing functions and code snippets.
- Use Copilot for code enhancement: It can suggest optimized versions of your existing code.
- Use Copilot for Inline Documentation & Comments
- Copilot can suggest fixes for errors and optimize inefficient code.

Don'ts ❌

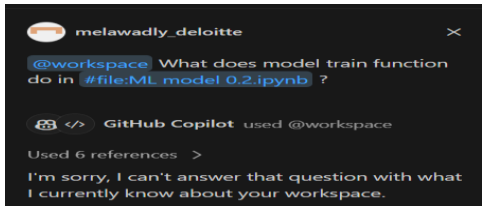
- Don't Rely on Copilot for Full Code Generation: Copilot works best as an assistant, not as a primary coder.

Pros

- you can submit prompts to Copilot in VS Code in several ways, depending on your workflow.
  - Chat view: have an AI assistant on the side to help with your questions and to provide code suggestions (Ctrl+Alt+I)
  - Inline Chat: start an inline chat conversation directly from the editor or integrated terminal to get suggestions in-place (Ctrl+I)
  - Quick Chat: ask a quick question and get back into what you're doing (Ctrl+Shift+Alt+L)
- There are several built-in chat participants, and extensions can also contribute chat participants. Use @ in the chat input field.
- Slash (eg. /fix) commands provide a shortcut to specific instructions to avoid that you have to write complex prompts.

Cons

- It still unaware of the context of the project despite using the chat participants and can't access the files.



Use Case: AI-Generated SDLC for an Emotion Classification ML

Overview

Using AI tools to fully generate the SDLC of a small project, this case study demonstrates how PairD and GitHub Copilot can streamline the development of an emotion classification machine learning model. From requirements gathering to code implementation and testing, AI accelerates each phase, reducing manual effort and improving efficiency. This use case provides a practical example of AI-assisted development in action.

Implementation Workflow

1. AI-Generated Requirements Document
2. AI-Created User Stories & Task Breakdown
3. AI-Generated System Architecture Diagram
4. AI-Generated Codebase
5. AI-Automated Unit Tests & Validation

1. Requirements Document Generation

PairD was used to generate the requirements document.

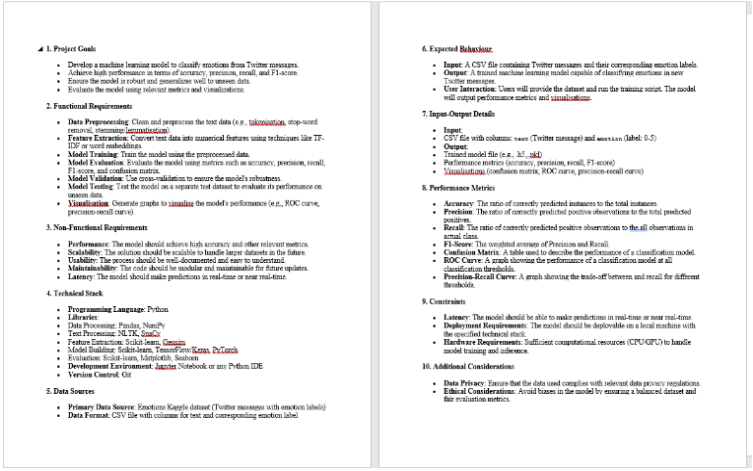
Prompt

I want to develop a ML emotion classification model with test, train and validation. The model will use the emotions Kaggle dataset (Each entry in this dataset consists of a text segment representing a Twitter message and a corresponding label indicating the predominant emotion conveyed. The emotions are classified into six categories: sadness (0), joy (1), love (2), anger (3), fear (4), and surprise (5).) The model's performance should be evaluated using the relevant metrics & graphs (It should be of high performance). Note that this project will be on my local computer. Start by Generating a requirements document that includes: project goals, functional and non-functional requirements, technical stack , data sources, expected behavior, input-output details, performance metrics and any constraints (e.g., latency, deployment requirements) Feel free to add what you see relevant.

A well-structured prompt acts as a blueprint for AI-generated content. It ensures accuracy, relevance, and completeness while reducing back-and-forth iterations. When using AI tools in software development, crafting precise prompts is essential for maximizing productivity and achieving high-quality results. **Unlike a prompt like "Generate a requirements document for a ML emotion classification model."**

The Generated Document

It generated a thorough requirements document that outlines the key aspects of developing a high-performance emotion classification model using the Kaggle dataset. It includes project goals, functional and non-functional requirements, technical stack, data sources, expected behaviour, input-output details, performance metrics, and constraints.

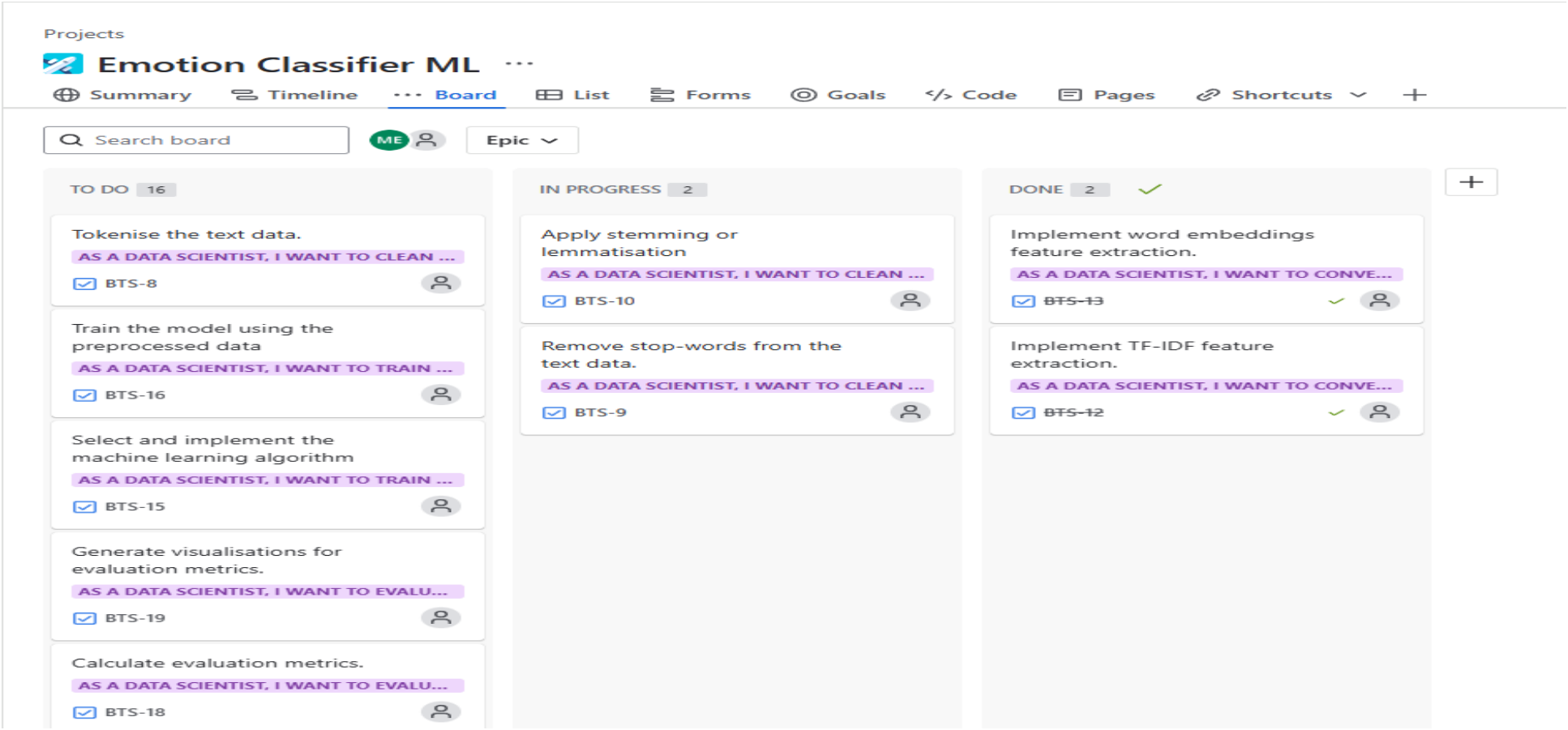


1. **Project Goals**
   - Develop a machine learning model to classify emotions from Twitter messages.
   - Achieve high performance in terms of accuracy, precision, recall, and F1-score.
   - Ensure the model is robust and generalizes well to unseen data.
   - Evaluate the model using relevant metrics and visualizations.
2. **Functional Requirements**
   - **Data Preprocessing**: Clean and preprocess the text data (e.g., tokenization, stop-word removal, stemming/lemmatization).
   - **Feature Extraction**: Convert text data into numerical features using techniques like TF-IDF or word embeddings.
   - **Model Training**: Train the model using the preprocessed data.
   - **Model Evaluation**: Evaluate the model using metrics such as accuracy, precision, recall, F1-score, and confusion matrix.
   - **Model Validation**: Use cross-validation to ensure the model's robustness.
   - **Model Testing**: Test the model on a separate test dataset to evaluate its performance on unseen data.
   - **Visualization**: Generate graphs to visualize the model's performance (e.g., ROC curve, precision-recall curve).
3. **Non-Functional Requirements**
   - **Performance**: The model should achieve high accuracy and other relevant metrics.
   - **Scalability**: The solution should be scalable to handle larger datasets in the future.
   - **Usability**: The process should be well-documented and easy to understand.
   - **Maintainability**: The code should be modular and maintainable for future updates.
   - **Latency**: The model should make predictions in real-time or near real-time.
4. **Technical Stack**
   - **Programming Language**: Python
   - **Libraries**:
   - Data Processing: Pandas, NumPy
   - Text Processing: NLTK, SpaCy
   - Feature Extraction: Scikit-learn, Gensim
   - Model Building: Scikit-learn, TensorFlow/Keras, PyTorch
   - Evaluation: Scikit-learn, Matplotlib, Seaborn
   - **Development Environment**: Jupyter Notebook or any Python IDE
   - **Version Control**: Git
5. **Data Sources**
   - **Primary Data Source**: Emotions Kaggle dataset (Twitter messages with emotion labels)
   - **Data Format**: CSV file with columns for text and corresponding emotion label
6. **Expected Behaviour**
   - **Input**: A CSV file containing Twitter messages and their corresponding emotion labels.
   - **Output**: A trained machine learning model capable of classifying emotions in new Twitter messages.
   - **User Interaction**: Users will provide the dataset and run the training script. The model will output performance metrics and visualizations.
7. **Input-Output Details**
   - **Input**:
   - CSV file with columns: `text` (Twitter message) and `emotion` (label: 0-5)
   - **Output**:
   - Trained model file (e.g., `.h5`, `.pkl`)
   - Performance metrics (accuracy, precision, recall, F1-score)
   - Visualizations (confusion matrix, ROC curve, precision-recall curve)
8. **Performance Metrics**
   - **Accuracy**: The ratio of correctly predicted instances to the total instances.
   - **Precision**: The ratio of correctly predicted positive observations to the total predicted positives.
   - **Recall**: The ratio of correctly predicted positive observations to the all observations in actual class.
   - **F1-Score**: The weighted average of Precision and Recall.
   - **Confusion Matrix**: A table used to describe the performance of a classification model.
   - **ROC Curve**: A graph showing the performance of a classification model at all classification thresholds.
   - **Precision-Recall Curve**: A graph showing the trade-off between and recall for different thresholds.
9. **Constraints**
   - **Latency**: The model should be able to make predictions in real-time or near real-time.
   - **Deployment Requirements**: The model should be deployable on a local machine with the specified technical stack.
   - **Hardware Requirements**: Sufficient computational resources (CPU/GPU) to handle model training and inference.
10. **Additional Considerations**
    - **Data Privacy**: Ensure that the data used complies with relevant data privacy regulations.
    - **Ethical Considerations**: Avoid biases in the model by ensuring a balanced dataset and fair evaluation metrics.

2. User Stories and Tasks Generation

The Agile Story Assistant prompt in PairD was used. In addition to providing the generated requirements document.

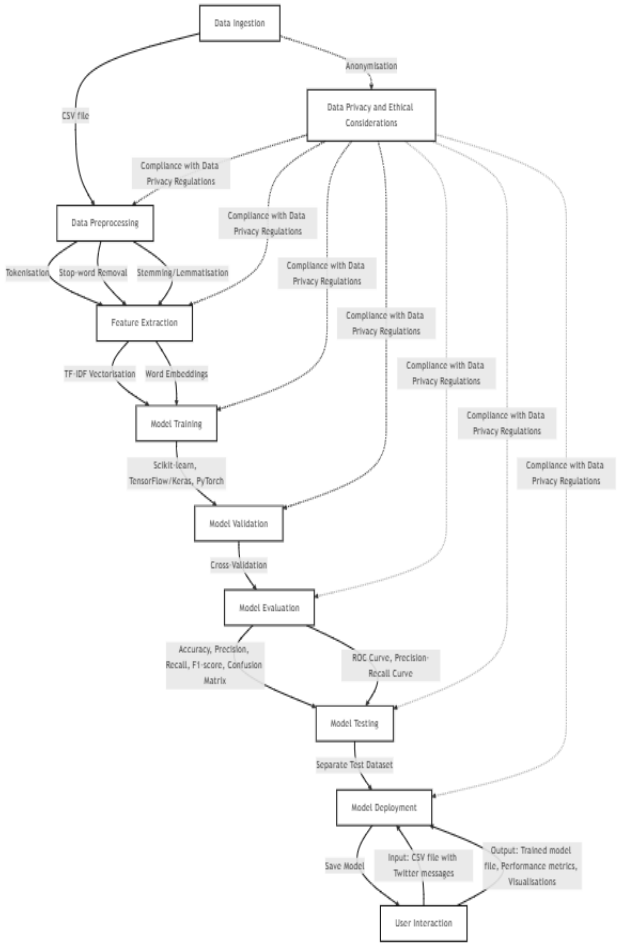
The Generated output



3. System Architecture Diagram

PairD was used to generate the system architecture diagram with the help of the MermaidViz Helper Prompt and by giving it the requirements document as well as the tasks. However, it insisted on generating a process diagram of the requirements.

In an attempt to get the desired outcome, A detailed description of the used components and steps of the system was provided, and this prompt was generated by PairD.



Prompt

" Create an architecture diagram that visually represents the components and workflow of the emotion classification model for Twitter messages. Components to Include: Data Ingestion: Source: Emotions Kaggle dataset (CSV file with Twitter messages and emotion labels) Data Format: CSV file with columns for text and emotion Data Preprocessing: Tokenisation Stop-word Removal Stemming/Lemmatisation Feature Extraction: TF-IDF Vectorisation Word Embeddings Model Training: Machine Learning Algorithms (e.g., Scikit-learn, TensorFlow/Keras, PyTorch) Training Process Model Validation: Cross-Validation Model Evaluation: Metrics: Accuracy, Precision, Recall, F1-score, Confusion Matrix Visualisations: ROC Curve, Precision-Recall Curve Model Testing: Separate Test Dataset Model Deployment: Save Model (e.g., .h5, .pkl) Deployment Environment: Local Machine with specified technical stack User Interaction: Input: CSV file with Twitter messages Output: Trained model file, Performance metrics, Visualisations Data Privacy and Ethical Considerations: Anonymisation Compliance with Data Privacy Regulations Steps to Create the Diagram: Identify the main components and their relationships. Use standard symbols and notation for each component (e.g., databases, processing units, models). Connect the components to show the data flow and interactions. Label each component and connection clearly. Include annotations for important details (e.g., data formats, evaluation metrics)."

4. Code Base Generation & Edits & Documentation

Code Generation

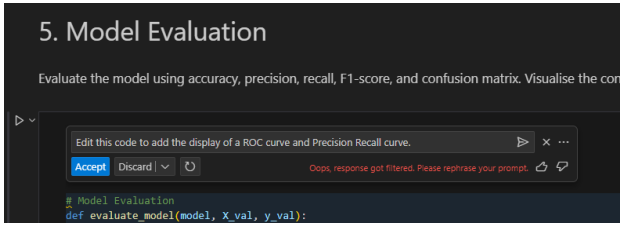
Code was Generated Using PairD as well by giving the previously generated System Architecture Diagram. In addition to this prompt

"Based on this architecture diagram, I want to develop a ML emotion classification model with train, test and validation. The model will use the emotions Kaggle dataset (Each entry in this dataset consists of a text segment representing a Twitter message and a corresponding label indicating the predominant emotion conveyed. The emotions are classified into six categories: sadness (0), joy (1), love (2), anger (3), fear (4), and surprise (5).) The model's performance should be evaluated using the relevant metrics & graphs (It should be of high performance). Note that this project will be on my local computer. Show me the project structure and the code. Be very thorough and walk me through the whole process."

The generated code was a Classical Machine Learning Model (Logistic Regression). That followed the System architecture proposed. It processes textual data, extracts meaningful features, trains a classification model, and evaluates its performance. The model achieved an accuracy of 89%.

Code edits

Some code edits were made to the code using GitHub Copilot to add the code to generate an ROC Curve and a Precision Recall Curve. However, it didn't perform well using the Inline chat feature.



So, Copilot chat was used to edit the code snippet.

Code Refactoring

Code was generated in multiple files structure, PairD was used to refactor the code without losing functionality into being in one Ipynb notebook. Copilot was tried at first However, it wasn't aware of the files content, so adding to the PairD chat was more feasible.

Key Takeaways

PairD vs. GitHub Copilot: Which Was More Useful?

PairD was the primary and more effective tool for this project, as it provided significant assistance in:

- Document Generation (Requirements, User Stories, Tasks)
- Code & Diagrams Generation (Functional Code, Architecture)
- Overall Workflow Automation

GitHub Copilot had limited usefulness because:

- It could not access workspace files directly.
- It required manual code snippets to provide assistance.

It was only useful for:

- Code Refactoring (suggesting optimizations & improvements)
- Error Fixing (identifying and resolving issues)
- Code Documentation (generating docstrings & inline comments)

Observation: The commercial version of Copilot is likely more powerful, offering enhanced capabilities and better project-wide assistance.

AI Tools Reduced SDLC Implementation Time

Using AI tools	Without AI tools
Total project time: ~ 3-4 hours This included: • Iterating on different prompts • Copying & pasting results into files • Training the model	Estimated time: 8+ hours This would have required manual effort for: • Writing requirements documents • Creating user stories & tasks • Designing architecture diagrams • Implementing the functional code

Conclusion

- PairD was more effective in generating documents and code, making it the better choice for automating early development stages.
- Copilot was only useful for refining existing code through error fixing, refactoring, and documentation.
- AI tools significantly accelerate the software development lifecycle (SDLC). However, current AI assistants still require manual intervention to provide context and integrate code effectively.