Project: Email Spam/Ham Classification

---

1. Introduction

1. 1 Project Objective

The objective of this project is to develop an automated email spam detection system capable of accurately classifying emails as either spam or ham.

1.2 Project Goals

This project aims to:

- Build preprocessing pipeline handling messy email data
- Implement TF-IDF vectorization to transform text into numerical features
- Create a deployable solution for real-time spam filtering

2. Methodology

2.1 Data Processing Pipeline

The spam detection pipeline processes email text through ten systematic stages:

Stages 1: Data Loading and Preprocessing

The Transform class implements critical cleaning operations:

- Null Value Removal: Eliminates records with missing text
- Lowercase Conversion: Standardizes text (important since spam often uses aggressive capitalization like "URGENT")
- Numeric Removal: Strips numbers that vary widely between emails
- Whitespace Normalization: Removes excessive spacing spammers use to evade filters
- Punctuation Elimination: Focuses analysis on word content

Stage 2: Tokenization and Linguistic Processing

The Tokenized_sentence class applies NLP techniques:

- Word Tokenization: Segments email text into individual words, handling URLs and special characters common in spam.

- Stop Word Removal: Filters out common words (e.g., "the", "is") appearing in both spam and ham, retaining discriminative words like "free", "winner", "urgent".
- Lemmatization: Reduces words to base forms (e.g., "winning", "wins", "won" → "win"). This consolidates spam variants into single features, making patterns more detectable.

Stage 3: Label Encoding

The Label_column class transforms spam/ham labels into numerical format (0 and 1) required for machine learning algorithms.

Stage 4: TF-IDF Vectorization

The Tfidf class converts processed tokens into numerical vectors.

2.2 Deep Learning Architecture

The neural network employs a feed-forward architecture:

Input Layer: Accepts TF-IDF feature vectors

Hidden Layers:

- First dense layer with RELU activation learns complex word combinations
- Dropout layer prevents overfitting to specific spam templates
- Batch Normalization stabilizes training
- Second dense layer compresses learned representations

Output Layer: Single neuron with sigmoid activation produces spam probability

Training Configuration:

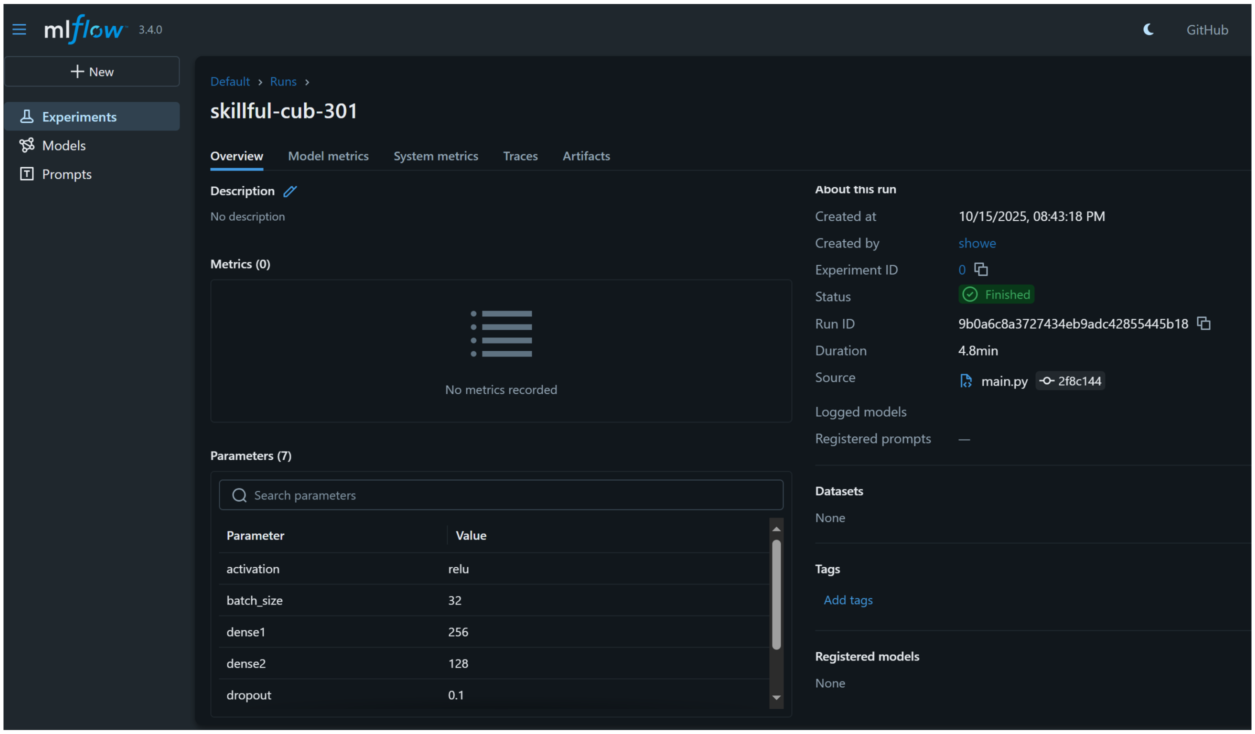
- Data Split: 70% training, 10% validation, 20% test
- Optimizer: Adam with adaptive learning rates
- Loss Function: Binary cross-entropy
- Early Stopping: Monitors validation loss to prevent overfitting

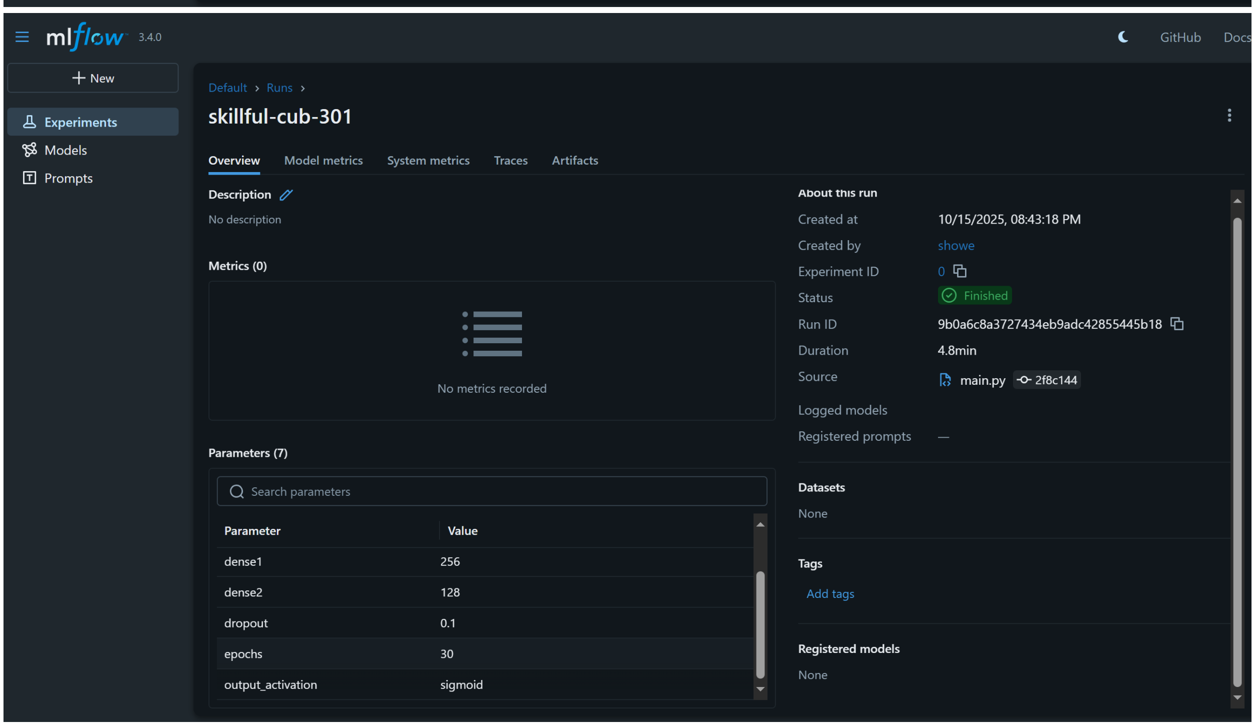
2.3 Hyperparameter Tuning with MLflow

The project implements comprehensive experiment tracking using MLflow, enabling systematic hyperparameter optimization and reproducible model development.

- Neural network architecture parameters (dense1, dense2 layer sizes, dropout rate)
- Activation
- epochs, batch_size

- Data splitting ratios





2.4 Evaluation Framework

Metrics Used:

- Accuracy: Overall correctness
- Precision: Reliability of spam flags (minimizes false positives)
- Recall: Spam catch rate (maximizes detection)
- F1-Score: Balances precision and recall

3. Personal Reflection

3.1 Identified Challenge: Class Imbalance

The most significant challenge in spam detection is class imbalance, where one class outnumbers the other. This creates model bias: Algorithms naturally predict the majority class more frequently. (With 90% ham emails, a model predicting all "ham" achieves 90% accuracy but catches zero spam.)

3.2 Proposed Adaptation Strategy

Oversampling the Minority Class:

Increase the number of instances in the minority class by duplicating existing samples or creating ones.

Undersampling the Majority Class:

Decrease the number of instances in the majority class by randomly removing them.

Sensitive Learning:

Change the algorithm to make it more sensitive to errors on the minority class.