User Guide  
Authorship Verification using Impostor Projections and Siamese Networks

**Purpose**  
This document is the User’s Guide / Operating Instructions for the capstone project. It focuses only on the nominal (successful) workflow of the system and is intended for end‑users who wish to run experiments and inspect past results.

# 1. Overview

The project implements an Impostor‑style Siamese Network built on fine‑tuned, domain‑specific BERT models to verify the authorship of short text chunks. The codebase is split into small, clean Python modules that are orchestrated by a single main.py controller. This lets you drive a complex end‑to‑end pipeline from either a command‑line menu or a headless Colab notebook.

# 2. System Capabilities – Nominal Flow

## 2.1 Fine‑Tuned BERT Artifacts

Every time the pipeline fine‑tunes a BERT model for an impostor‑author, it:  
1. saves the checkpoint under data.fine\_tuned\_bert\_model\_path (default saved\_trained\_models/berts/);  
2. pushes the same directory to the HuggingFace Hub repository configured in config.yaml. Our fine-tuned BERTs hub: <https://huggingface.co/ElyMK1/bert-shakespeare-english-mlm>

3. A BERT Tokenizer is also uploaded along with the MLM model. We use this fine-tuned tokenizer for preprocessing an impostor-author.

This process guarantees a local backup plus a single cloud source‑of‑truth, so you can later load the model with transformers.from\_pretrained() from any machine.

## 2.2 Interactive Menu (main.py)

python3 main.py

1 Create Dataset  
2 Run Model Procedure (train Siamese networks)  
3 Classification Procedure (signal generation)  
4 DTW Procedure (distance matrices between signals for each iteration– trained model)  
5 Isolation Forest Procedure (anomaly scores from each distance matrix)  
6 Clustering Procedure (final clusters of all anomaly scores vectors)  
7 Fine‑Tune All BERTs  
0 Exit  
  
Nominal path: 1 → 7 (Once only! Our HF repository already has all 31 fine-tuned BERTs on the listed hyperparameters down below) → 2 → 3 → 4 → 5 → 6 → 0. Each step requests confirmation—answer yes to proceed.

# 3. Prerequisites

\* Python 3.10 and the exact package versions in Pipfile.lock  
\* pipenv ≥ 2024.4.1 (pip install pipenv)  
\* CUDA 11.8 / cuDNN 8.1.1 on local GPUs (for code testing) or CUDA 12.5 on Colab (for real use).  
\* HuggingFace access token (write scope)  
\* Weights & Biases API key (if wandb.enabled: true – this key should generally not be disabled as our entire result-based procedures rely on its storage, unless locally testing).

# 4. Configuration File – config/config.yaml

The exact configuration file shipped with the repository. Modify values there (not the code) to redirect artifact locations, tune hyper‑parameters, toggle services and more.

## 4.1 Field‑by‑Field Explanation

# Field‑by‑Field Explanation

| **Section** | **Key** | **Description & Effect** |
| --- | --- | --- |
| **data** | organised\_data\_folder\_path | Root folder grouping raw & processed data. |
|  | shakespeare\_data\_source / impostors\_data\_source | Names of JSON files with canonical texts for each author pool. |
|  | shakespeare\_path / impostors\_path | Where raw plaintext files are stored before dataset creation. |
|  | all\_impostors\_data\_source | Aggregated impostor collection used during to fine-tune BERTs. |
|  | classify\_text\_data\_source | JSON file of unknown text used during inference. |
|  | fine\_tuned\_bert\_model\_path | Local directory saving per‑author fine‑tuned BERT checkpoints before they are uploaded to HuggingFace. |
|  | trained\_siamese\_path | Folder for Keras .h5 weights produced by Run Model Procedure. |
|  | signals\_folder\_name | Sub‑dir created under each model folder containing JSON arrays of numerical signals. |
|  | isolation\_forest.\* | Location & filenames for anomaly scores produced by Isolation Forest. |
|  | clustering\_output\_file | Final JSON with cluster labels after Clustering Procedure. |
|  | dtw.\* | Folder + file naming pattern for DTW distance matrices and related helper files. |
|  | pairs | JSON file containing all pairs by impostor-author names, used to train models and use them during the inference procedures.  It also defines the last iterations, allowing us to pick up where we left off during the last runtime. |
| **training** | load\_pretrained\_model | If true, skip training models for impostor pairs, if they exist. Otherwise, train models and upload the version as the latest tag in the artifacts registry. |
|  | optimizer.\* | Learning‑rate schedule and gradient‑clipping settings. |
|  | training\_batch\_size / epochs | Core loop batch size & number of epochs. |
|  | early\_stopping.\* | Keras early‑stopping callback parameters. |
|  | impostor\_chunk\_ratio | Multiplier for data augmentation of the number of chunks from each impostor during pre-processing. |
|  | test\_split | Validation dataset split ratio. |
| **model** | chunk\_to\_batch\_ratio / chunk\_size | How many chunks are in a batch, and how many tokens are in a chunk. |
|  | cnn.\* | 1‑D convolution filter bank capturing local n‑gram features. |
|  | bilstm.\* | Bi‑directional LSTM width and dropout for contextual fusion. |
|  | fc.\* | Width of final fully‑connected layer before computing pairwise distance. |
| **isolation\_forest** | number\_of\_trees / percentile\_threshold / anomaly\_score\_threshold | Hyper‑parameters governing anomaly detection. |
| **clustering** | algorithm | Currently supports k‑medoids (robust to outliers). |
|  | k-medoids.\* | Cluster count (n\_clusters) and seed (random\_state) for deterministic medoid selection. |
| **bert** | All keys | Control HuggingFace Trainer for domain fine‑tuning (sequence length, MLM probability, logging, checkpoint cadence). |
| **wandb** | All keys | Enable/disable W&B, set run names and artefact naming conventions. |

# 5. Monitoring, Visualising & Saving Experiments with Weights & Biases

Key URLs:

Dashboard (Runs): <https://wandb.ai/authorship-verification-siamese-network/siamese-authorship-verification>  
Artifacts: <https://wandb.ai/authorship-verification-siamese-network/siamese-authorship-verification/artifacts/>  
Table view: <https://wandb.ai/authorship-verification-siamese-network/siamese-authorship-verification/table?nw=nwuserwezkc21>

**Workflow:**  
1. Login once: wandb login $API\_KEY  
2. Run any pipeline stage; metrics & artefacts sync live.  
3. View Runs, Artifacts (trained Siamese .h5, fine‑tuned BERT checkpoints, DTW matrices, Signal plots, T-SNE Graph of Clusters, etc.), and Table for comparisons.  
4. Download: wandb artifacts get siamese-authorship-verification-branches-{impostor 1 name}\_{impostor 2 name}:latest.  
Or a specific tag. Though, generally the latest is the intended one for use.

# 6. Google Colab Notebook – Authorship\_Verification\_using\_Impostor\_Projections\_and\_Siamese\_Networks\_Model.ipynb

The notebook is a **bootstrapper**, not a research playground. It contains just five executable cells:

1. Reset & Python 3.10 setup – Cleans workspace, installs Python 3.10 via apt.  
2. CUDA 12.2 & cuDNN install – Adds NVIDIA repository, installs driver libs, verifies with nvcc. These installations take ~30 minutes, but are quite necessary to be able to have a compatible host system with the latest Tensorflow version (2.18.0), **as of July 2025 Google Colab natively supports CUDA 12.5 and Tensorflow 2.18.0, however we worked with CUDA 12.2, so skip these installations with caution**. Maintenance guide elaborates on this matter.

3. Clone repo & pip install – git clone and pip install of project requirements.  
4. Launch main.py in headless mode – python main.py --pipeline nominal (or interactively).