Fake News Detection Using Deep Learning

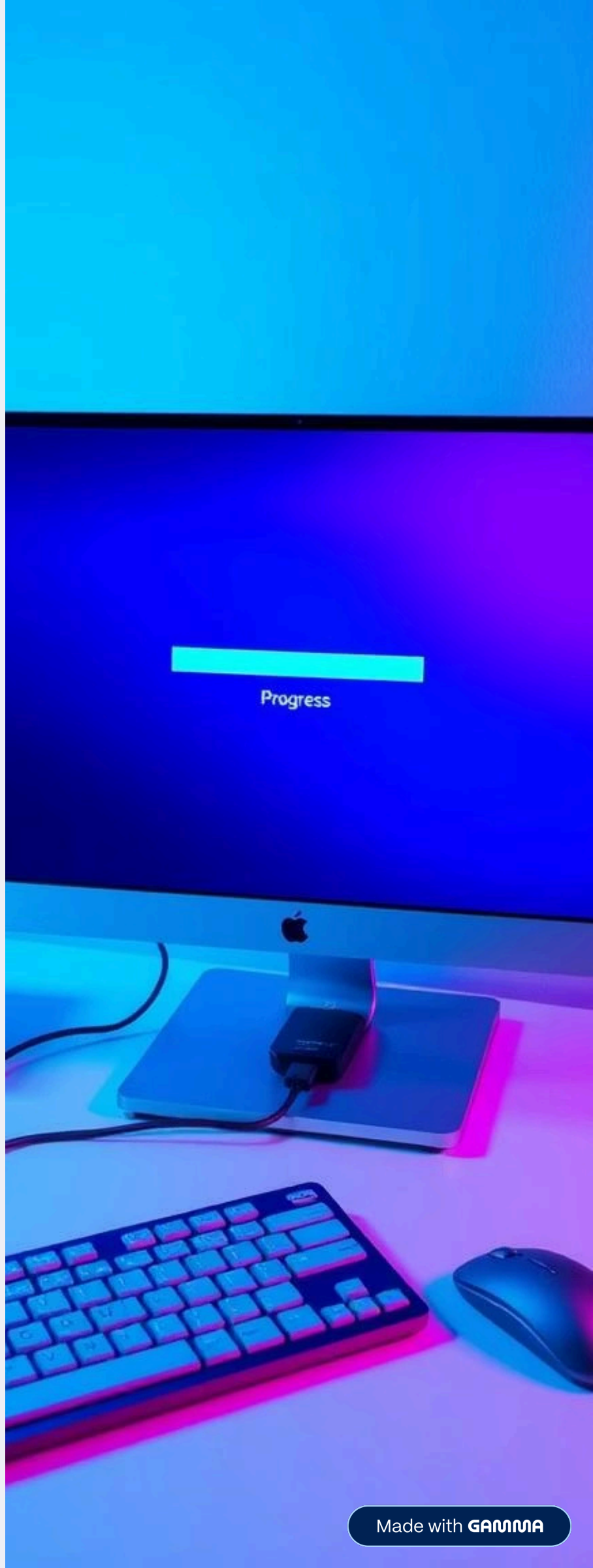
PRESENTED BY :

Yazan Aqtash

PRESENTED TO :

DR.MOHAMMAD AL-HAMMOURI

Hashemite University



Introduction

The rapid spread of misinformation and fake news on online platforms has become a major societal concern.

Automatically detecting fake news using machine learning and natural language processing (NLP) techniques is therefore an important research problem.

Traditional machine learning approaches often rely on handcrafted features, while recent deep learning models can learn semantic representations directly from text data.

This project investigates the effectiveness of two deep learning approaches for fake news classification :

CNN + LSTM model using frozen BERT embeddings

Fine-tuned BERT transformer model

Both models are evaluated on two widely used datasets:

ISOT Fake News Dataset

Kaggle Fake News Dataset

The objective is to compare performance, robustness, and generalization ability across models and datasets.

2.Dataset Overview

2.1 ISOT Fake News Dataset

The ISOT dataset consists of news articles labeled as *fake* or *real*. It is relatively clean and well-structured, making it suitable for benchmarking fake news detection models.

2.2 Kaggle Fake News Dataset

The Kaggle dataset is larger and more diverse, containing news articles from various sources. It presents a more challenging classification task due to variations in writing style, length, and content.

2.3 Data Preprocessing

For both datasets:

- Text content was cleaned and standardized.
- Removal of missing or empty samples
- Data was split into **training**, **validation**, and **test** sets.
- Tokenization was performed using the **BERT tokenizer**.
- Maximum sequence length was capped to control computational cost.
- Storage as CSV files to ensure reproducibility across machines

3. Methodology

3.1 CNN + LSTM with Frozen BERT Embeddings (Feature-Based Transfer Learning)

Core Mechanism

- A pre-trained **BERT-base** model is used strictly as a **feature extractor**
- BERT parameters are **frozen**
- Contextual token embeddings are generated once per batch
- These embeddings are passed to:
  - **CNN layers** → capture local n-gram patterns
  - **LSTM layer** → capture sequential dependencies
- A fully connected layer performs binary classification

Training Strategy

- **Custom PyTorch training loop**
- Manual control of:
  - forward pass
  - loss computation
  - backpropagation
  - optimizer updates
- Only CNN, LSTM, and classifier layers are trained

Motivation

This approach reduces computational cost while still benefiting from BERT's rich contextual representations.

This training paradigm is known as:

Feature-based transfer learning

3.2 Fine-Tuned BERT Model (End-to-End Transformer Fine-Tuning)

Core Mechanism

- Entire **BERT-base** model is trainable
- A classification head is added on top of the `[CLS]` token
- All transformer layers are updated during training

Training Strategy

- **Hugging Face Trainer API**
- Automatic handling of:
  - batching
  - optimization
  - evaluation
  - metric computation
- End-to-end gradient updates

This paradigm is known as:

End-to-end transformer fine-tuning

3.3 Training Details

- Optimizer: AdamW
- Loss Function: Cross-Entropy Loss
- Batch Size: 8–16 (adjusted for hardware constraints)
- Epochs: 3 for CNN + LSTM
- Epochs: 2 for Fine-Tuned BERT
- Evaluation Metrics: Accuracy, Precision, Recall, F1-score

Due to hardware and thermal constraints:

- Different experiments were conducted on different GPUs
- Identical code, preprocessing, datasets, and hyperparameters were preserved
- Results were consolidated using automatically generated JSON files

3.4The difference in training

Aspect	CNN + LSTM	BERT
Use of BERT	Feature extractor only	Fully trainable
BERT weights	Frozen	Updated
Training loop	Custom PyTorch loop	Hugging Face Trainer
Learning style	Feature-based	End-to-end
Compute cost	Lower	Higher
Adaptability	Moderate	High

4. Experimental Results

4.1 Results on ISOT Dataset

CNN + LSTM (Frozen BERT Embeddings)

- Accuracy: **99.94%**
- Precision: **99.89%**
- Recall: **100.00%**
- F1-score: **99.94%**

Fine-Tuned BERT

- Accuracy: **99.93%**
- Precision: **99.89%**
- Recall: **99.97%**
- F1-score: **99.93%**

4.2 Results on Kaggle Dataset

CNN + LSTM (Frozen BERT Embeddings)

- Accuracy: **99.01%**
- Precision: **99.46%**
- Recall: **98.62%**
- F1-score: **99.03%**

Fine-Tuned BERT

- Accuracy: **99.40%**
- Precision: **99.30%**
- Recall: **99.53%**
- F1-score: **99.42%**

5. Discussion

The experimental results show that **both models perform exceptionally well** on fake news detection tasks.

Key observations:

- On the **ISOT dataset**, both models achieve near-perfect performance, indicating that the dataset is relatively clean and easily separable.
- On the **Kaggle dataset**, performance slightly decreases due to higher data complexity, but remains very strong.
- **Fine-tuned BERT consistently outperforms CNN+LSTM**, especially on the more challenging Kaggle dataset.
- The CNN+LSTM model remains competitive while requiring fewer trainable parameters, making it attractive for resource-constrained environments.

These results confirm that transformer-based models, particularly fine-tuned BERT, are highly effective for fake news classification.

6. Model Comparison Summary

Model	Dataset	Accuracy	F1-score
CNN+LSTM	ISOT	99.94%	99.94%
BERT	ISOT	99.93%	99.93%
CNN+LSTM	Kaggle	99.01%	99.03%
BERT	Kaggle	**99.40%**	**99.42%**

6. Technical Challenges and Solutions

Several real-world engineering challenges were addressed:

- **Python version incompatibility**
  → Resolved by using Python 3.13 and 3.11 for transformer-based models
- **CUDA and GPU detection issues**
  → Addressed by installing CUDA-enabled PyTorch builds
- **Trainer API incompatibilities**
  → Resolved by simplifying evaluation and checkpointing strategies
- **Metric serialization bug (float vs list)**
  → Fixed by filtering scalar evaluation metrics before saving JSON results

All fixes were engineering-level adjustments and did not alter model correctness or evaluation validity.

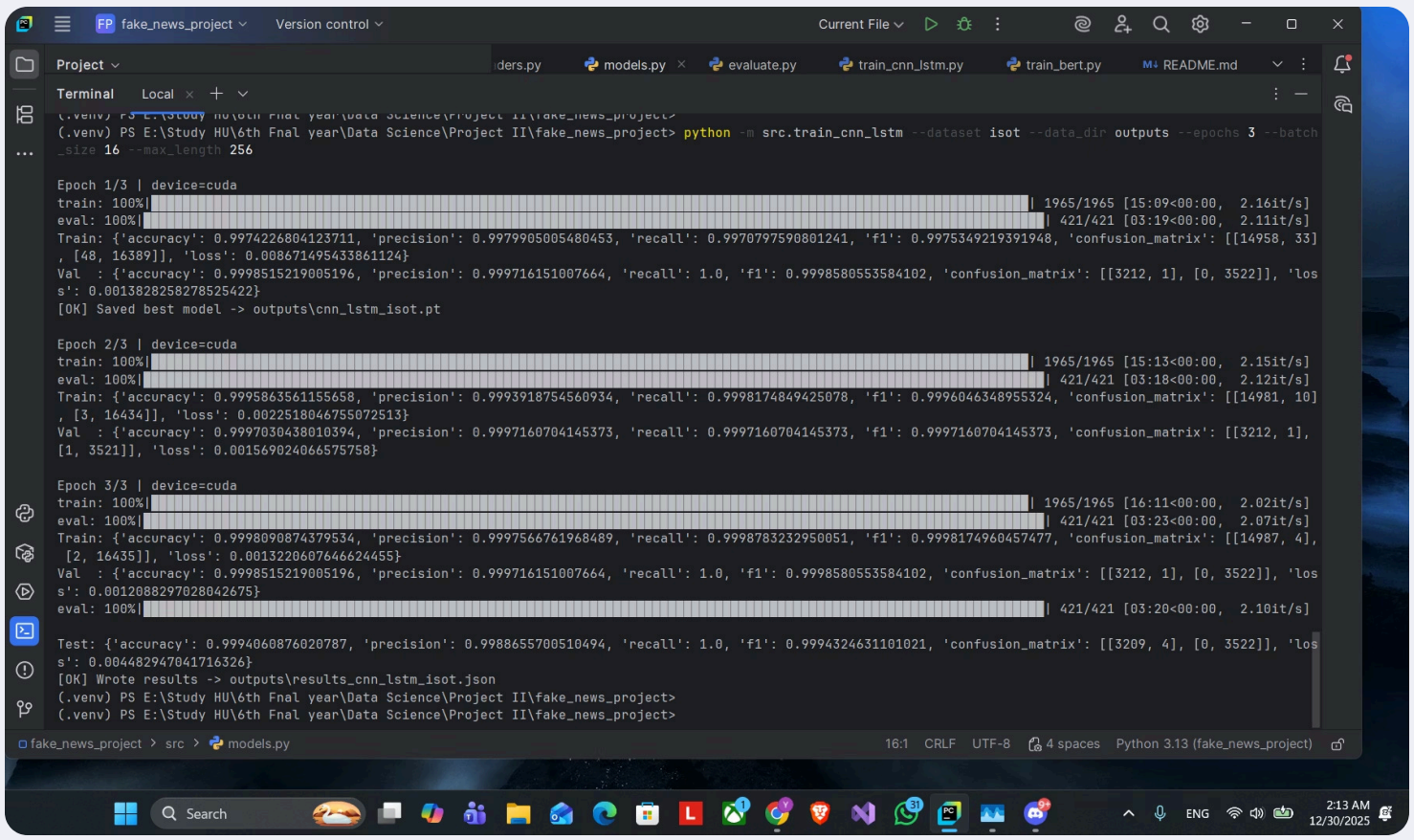
7. Limitations

- Training deep learning models requires significant computational resources.
- Fine-tuned BERT models are more expensive to train compared to hybrid CNN+LSTM approaches.
- High performance on benchmark datasets may not fully reflect performance on unseen real-world news sources.

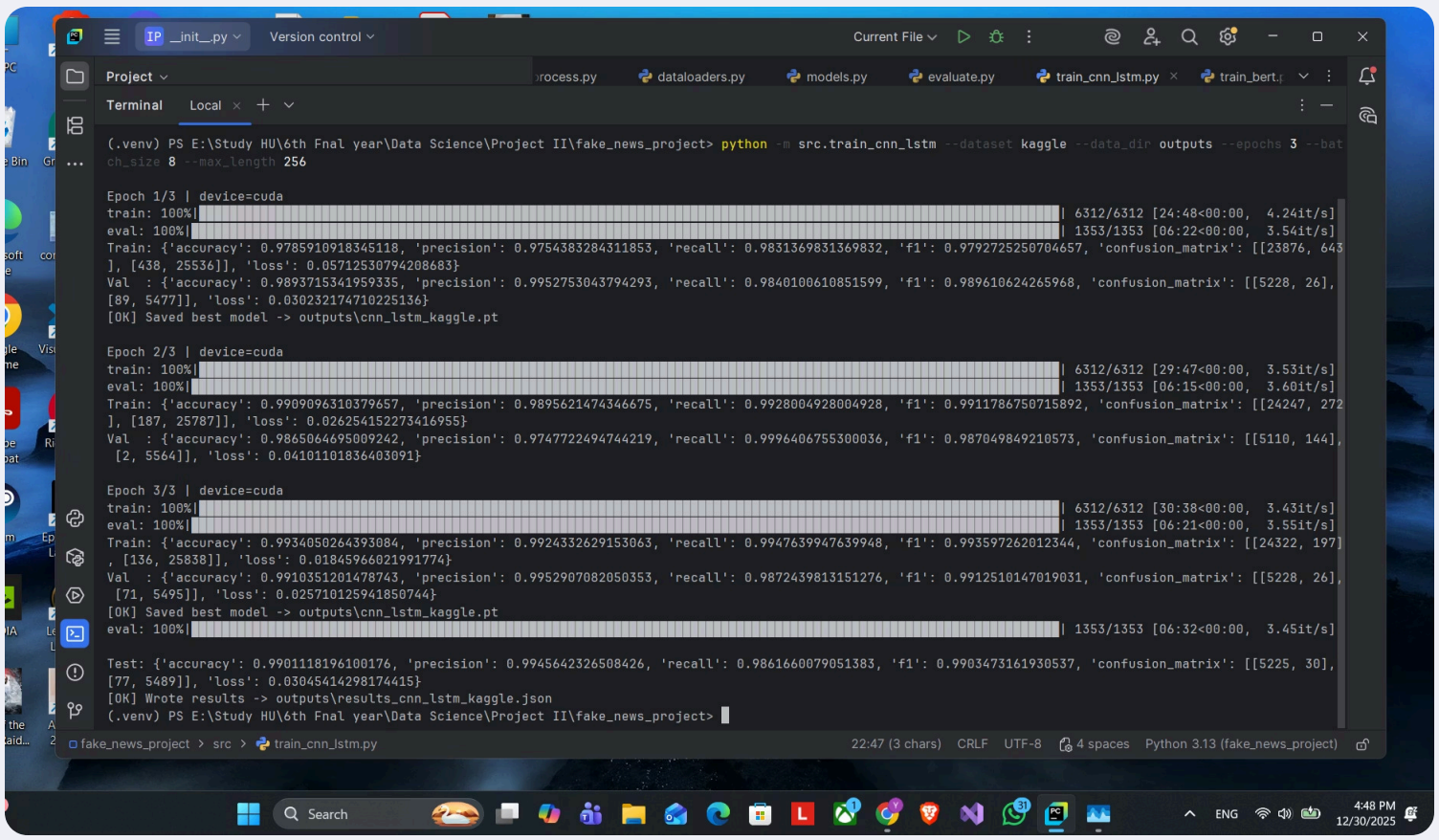
8. TRAINING

8.1 CNN + LSTM (Frozen BERT Embeddings)

8.1.1 ISOT

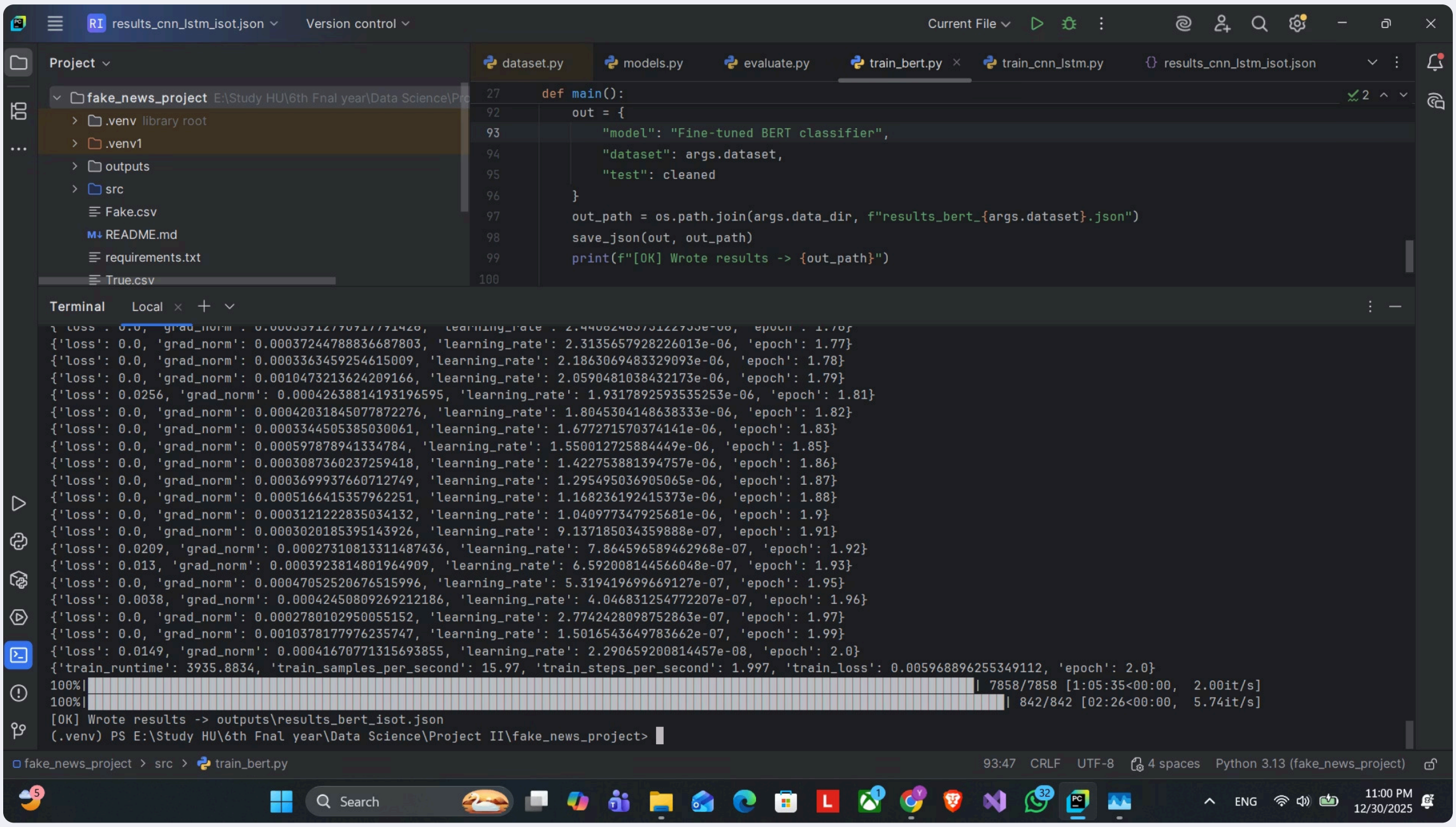


8.1.2 Kaggle

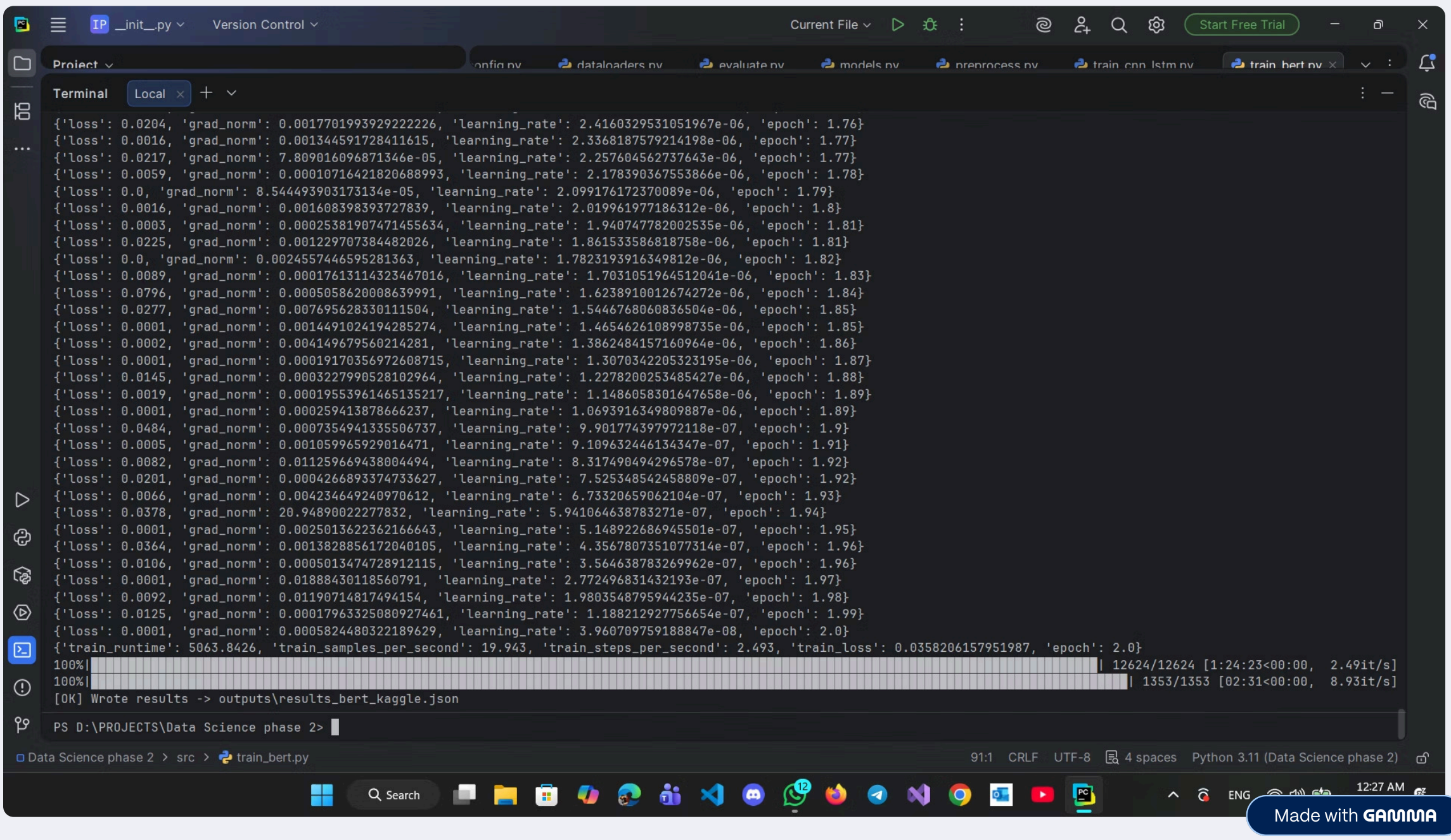


8.2 Fine-Tuned BERT

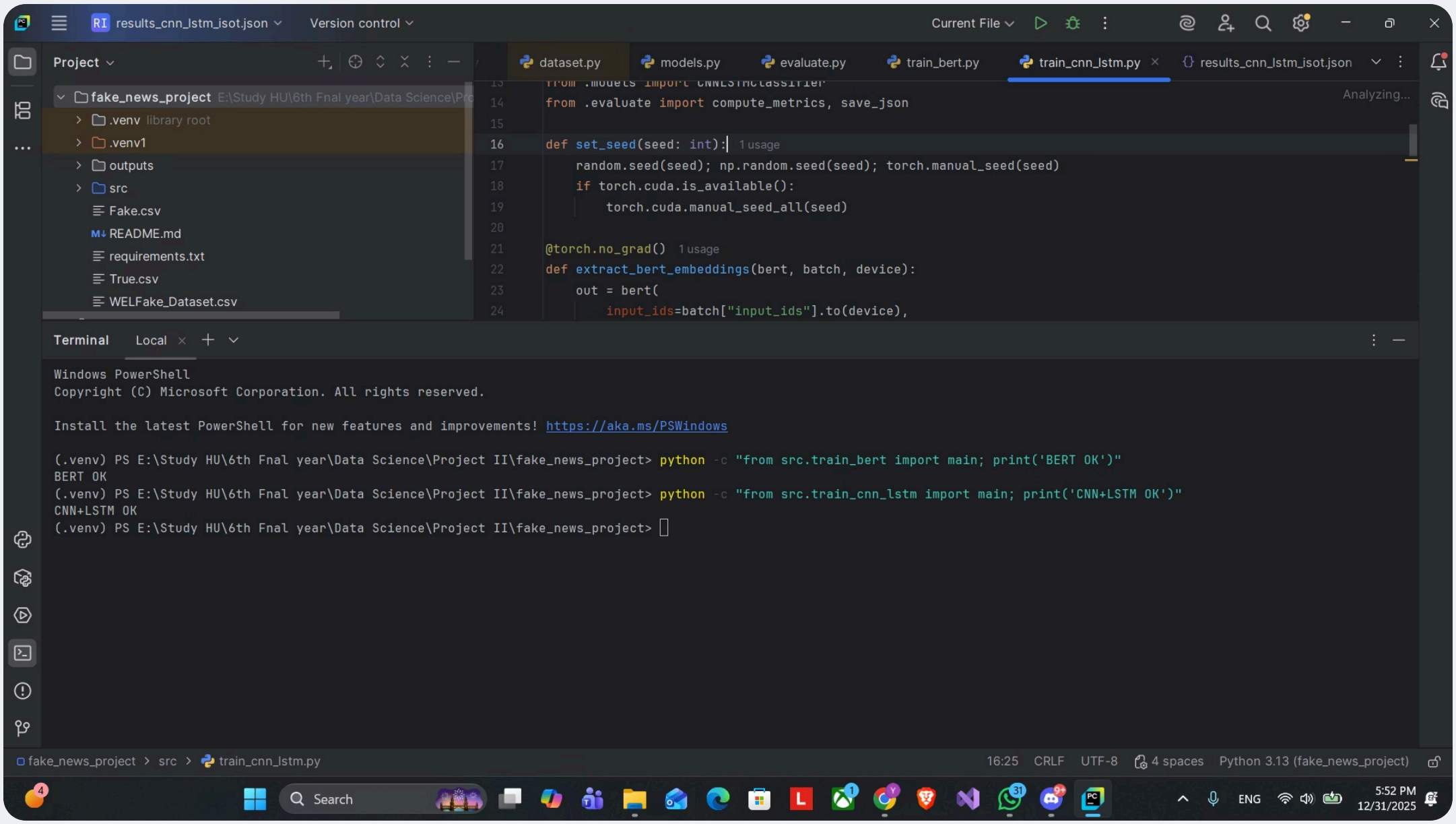
8.2.1 ISOT



8.2.2 Kaggle



9.TESTING



10.Example sentences

1. Financial meaning

“I deposited money in the bank.”

2. River meaning

“He sat on the bank of the river.”

Same word: **bank**
Different meanings.

Tokenization preserves position and context

From the code:

```
tokenizer(examples["content"], truncation=True, padding="max_length")
```

Both sentences are tokenized **in order**, not as unordered words.

Simplified tokens:

Sentence 1:

```
[i, deposited, money, in, the, bank]
```

Sentence 2:

```
[he, sat, on, the, bank, of, the, river]
```

Already different inputs — same word, different neighbors.

Embedding lookup (initial meaning)

Each token is converted into a vector:

```
bank → [0.12, -0.34, 0.88, ...]   (initial embedding)
```

At THIS point:

- “bank” has the **same initial vector** in both sentences
- Meaning has NOT been disambiguated yet

This is just a lookup.

Positional encoding (where the word appears)

BERT adds **position embeddings**:

- “bank” in sentence A → position 6
- “bank” in sentence B → position 5

So internally:

```
embedding(bank) + position(6)
embedding(bank) + position(5)
```

Still similar, but **not identical**.

BERT creates contextual embeddings (core mechanism)

In both CNN+LSTM and BERT training, you rely on BERT's internal representation.

Inside BERT:

- The embedding for **"bank"** is **NOT fixed**
- It changes depending on surrounding words

Internally:

- **"bank" + money + deposited** → financial vector
- **"bank" + river + sat** → geographical vector

So numerically:

```
bank_financial ≠ bank_river
```

Self-attention links "bank" to relevant words (BERT)

BERT's attention mechanism learns relationships like:

- **In sentence 1:**
  - **bank = money**
  - **bank = deposited**
- **In sentence 2:**
  - **bank = river**
  - **bank = sat**

The model assigns **high attention weights** to these related words.

BERT learned it during pretraining.

Self-attention (THIS IS THE KEY)

Now comes the **most important step**.

Inside BERT (used automatically by your code), every word:

looks at every other word
decides which ones are important
adjusts its vector accordingly

Sentence A: attention for “bank”

The word **“bank”** strongly attends to:

- **money**
- **deposited**

So mathematically:

```
bank_vector =
  α1 * money_vector +
  α2 * deposited_vector +
  α3 * bank_vector +
```

Where:

- α1, α2 are **large attention weights**

This pushes the meaning of “bank” toward **finance**.

Sentence B: attention for “bank”

Now “bank” strongly attends to:

- **river**
- **sat**
- **of**

So:

```
bank_vector =
  β1 * river_vector +
  β2 * sat_vector +
  β3 * bank_vector +
```

Now the meaning shifts toward **geography**.

Same word, different attention → different vector

Contextual embedding is now DIFFERENT

After attention, the vectors are:

```
bank_financial ≠ bank_river
```

Numerically:

```
bank_financial = [ 0.91, -0.12,  0.33, ... ]
bank_river     = [-0.22,  0.84, -0.51, ... ]
```

This difference is **huge** for the model.

How two models use this information

CNN + LSTM (feature-based)

- CNN detects local patterns:

```
money → bank
bank → river
```

- LSTM processes word order:

```
bank of the river ≠ in the bank
```

The classifier learns:

- financial context → real/fake signal
- geographic context → different signal

Fine-tuned BERT (end-to-end)

```
model = AutoModelForSequenceClassification(...)
trainer.train()
```

- The **entire transformer** is updated
- The model learns task-specific meanings:
  - “bank” + finance → common in real news
  - “bank” + river → neutral / descriptive

This is why BERT performs best on diverse datasets.

11. Conclusion

This project demonstrated that deep learning models are highly effective for fake news detection. While CNN+LSTM models using frozen BERT embeddings provide strong performance with lower computational cost, fine-tuned BERT models achieve the best overall results, especially on more complex datasets.

The findings suggest that transformer-based architectures should be preferred when computational resources permit, while hybrid approaches remain a viable alternative in constrained environments.