Cole Perry
perry.col@northeastern.edu
CS 6140
Final Project

Problem Description:

Fake news, especially around elections, has become a major issue. Fake news shakes people's trust in the media and even influences how they vote. More often than not, misinformation slips in through biased wording, misleading headlines, or a lack of context, making it frustratingly hard for readers to separate fact from fiction. As social media and online platforms amplify every story, these deceptive narratives can spread faster than corrections, creating echo chambers and deepening divides.

When people lose confidence in what they read, it undermines their ability to make thoughtful decisions about the issues and leaders that affect their daily lives. This problem reaches far beyond just politics, it touches on public health, climate change, and social justice too, wherever accurate information matters. Tackling fake news is essential because an informed public is the bedrock of a healthy democracy. If we don't find better ways to spot and flag misleading content, the line between truth and falsehood will only blur further, putting our shared values at risk.

Motivation:

First, I dove in with TF–IDF and logistic regression because, as *A Review of Fake News Detection Approaches: A Critical Analysis of the Literature* argues, you really need a baseline model without the extra bells and whistles to figure out which words and metadata signals actually move the needle. That simple model quickly exposed quirks in my data (like skewed class distributions) and gave me a clear performance floor to beat. Then, inspired by *From Misinformation to Insight: Machine Learning Strategies for Fake News Detection*, I rolled up my sleeves and started tuning everything I could—regularization strength, class weights, decision thresholds—to see just how much juice I could squeeze out of a "lightweight" classifier. Those papers showed me that even the simplest setup can pack a punch when you tweak its knobs thoughtfully. And having that hands-on, transparent foundation made me feel confident before I stepped up to the big leagues with RoBERTA and DeBERTa.

I eventually swapped out TF–IDF for a RoBERTa backbone because my simple bag-of-words setup, while transparent, was hitting a performance ceiling when it came to capturing subtle language cues. I was particularly inspired by *FakeBert: Fake News Detection in Social Media with a BERT-based Deep Learning Approach*, which showed that BERT's contextual embeddings—when combined with lightweight classifiers—could push accuracy well past what TF–IDF alone can manage. At the same time, the study *Fine-Tuning BERT Models to Classify Misinformation on Garlic and COVID-19 on Twitter* convinced me that a fine-tuned transformer could really handle the nuances of deceptive phrasing and domain-specific jargon. Making the leap to RoBERTa meant I could tap into deep, pre-trained language representations that inherently understand word order, negations, and subtle tone shifts—things that TF-IDF simply flattens out. In practice, this switch unlocked a big jump in macro-F1 and set the stage for my later move to DeBERTa.

I decided to move from RoBERTa to DeBERTa after reading He et al.'s paper "DeBERTa: Decoding-enhanced BERT with Disentangled Attention," because it really opened my eyes to how much richer language understanding you get when you tease apart word meaning and word position. RoBERTa was great at grasping context, but it sometimes glossed over tiny shifts in phrasing or negation that can completely flip a political claim. DeBERTa's clever "disentangled attention" mechanism means the model separately learns what each word means and exactly where it sits in a sentence, which translates into sharper detection of sarcasm, subtle bias, or hidden negations, which a lot of research papers were struggling with detecting.

Research Paper Analysis

Paper	What is the paper about	How did they approach the problem	Outcome or limitation
FakeBert: Fake News Detection in Social Media with a BERT-based Deep Learning Approach	Introduces FakeBert, a deep model for detecting fake news on social media.	Combined BERT embeddings with parallel CNN layers to capture both global context and local patterns; achieved 98.9% accuracy on benchmark datasets.	Limited to English language content.
Fine-Tuning BERT Models to Classify Misinformation on Garlic and COVID-19 on Twitter	Detects COVID-19 and garlic-related misinformation on Twitter via BERT variants.	Evaluated five different BERT models on domain-specific tweet datasets; measured performance with accuracy and F1. One variant (BERTweet-large) reached 91.1% accuracy.	Struggles with subtle or contradictory misinformation in ambiguous tweets.
Fake News Detection in Multiple Platforms and Languages	Proposes a cross-platform, multilingual fake-news detector using language-agnostic text features.	Extracted platform and language independent features (e.g., stylometric, lexical) and tested on five datasets across different languages; outperformed basic baselines in each case.	Uses only text ignores image/video cues that often accompany modern misinformation.
A Review of Fake News Detection Approaches: A Critical Analysis of the Literature	Surveys recent fake-news detection methods, highlighting challenges and performance factors.	Critically analyzed model choices, feature fusion strategies, and common pitfalls (e.g., overfitting); summarized strengths and weaknesses across studies.	Notes dataset limitations and the difficulty of fusing heterogeneous features.
Towards Data- and Compute-Efficient Fake-News Detection: An Approach Combining	Reduces human labeling costs via an active-learning feedback loop with pre-trained	Had the model flag only "uncertain" instances for human annotation, then re-trained weekly;	Weekly retraining may lag behind rapidly evolving misinformation narratives.

Active Learning and Pre-Trained Language Models	models.	maintained accuracy on par with fully labeled training while halving labeling effort.	
Parameter-Efficient Fine-Tuning of Large Language Models Using Semantic Knowledge Tuning	Enables cheap fine tuning of large language models without full-parameter updates.	Introduced small LoRA adapter layers added to frozen pre-trained weights; achieved similar downstream performance with just 6% of GPU time compared to full fine-tuning.	Doesn't explore decision threshold optimization, focuses purely on training efficiency.
A BERT-Based Multimodal Framework for Enhanced Fake News Detection	Tackles fake news detection by merging text with image derived clues via OCR in a multimodal setup.	Ran article text through BERT and extracted text from images via OCR; fused both streams in a joint classifier, yielding 99.97% accuracy and excellent F1.	Discards full visual context (only uses text extracted from images).
Covid-19 Fake News Detection – A Hybrid CNN-BiLSTM-AM Model	Presents a hybrid deep learning model for spotting COVID-19 related fake news.	Employed CNNs to capture local word patterns, BiLSTMs for sequence modeling, and attention layers to highlight key text segments; demonstrated strong domain specific performance.	Specialized to COVID-19 content performance may drop on general news.
Multimodal Fake News Detection with Contrastive Learning and Optimal Transport (MCOT)	Combines text and visuals using contrastive learning and optimal transport for robust fake news detection.	Matched image text pairs via contrastive objectives and aligned modalities using optimal-transport losses; outperformed prior multimodal methods on social-media datasets.	Relies heavily on large pre-trained models and was tested on relatively small datasets, generalization is uncertain.
DeBERTa: Decoding-enhanced BERT with Disentangled Attention	Presents DeBERTa's architecture, which separates content embeddings from positional embeddings and adds enhanced pre-training tasks.	Introduced disentangled attention to better capture word meaning and position; demonstrated superior performance to RoBERTa on GLUE benchmarks and downstream tasks.	Increased model complexity and computational cost compared to RoBERTa.

Dataset Description

Source: I am using the FakeNewsNet dataset, which aggregates political news articles from multiple online platforms. Each row in the raw dataset maps to one article and includes:

- title (string): the article headline
- text (string): the full body article
- site_url (string): the publisher's domain (ex. CNN)

- published (timestamp): publication date and time
- type (string): categorical labels (“bs”, “bias”, “conspiracy”, “hate”, “satire”, “state”, “junksci”, “fake”)

Size:

- Total rows: 12,999 articles
- 12,556 labeled fake (assuming “bs”, “conspiracy”, “hate”, “satire”, “state”, “junksci”, “fake”) are all fake news and (“bias”) is real news
- Features
    - combined_text : I concatenate cleaned title and body into one “combined_text” field per article. This combined text is what I feed into BERT (or TF-IDF) so the model can learn patterns in the words themselves.
    - publisher_id (integer): each unique site_url is assigned a small integer code via pandas’ category encoding. This numeric ID lets the model learn whether certain sources are more or less likely to publish fake content.
    - date_ordinal (integer): publication date converted to “days since January 1, year 1” using Python’s toordinal(). By representing the publish date as a single integer, the model can potentially learn patterns (for instance, fake articles surging around certain events or election dates).
- Target variable: a binary column “label” with values:
  • “fake” if type is in {“bs”, “conspiracy”, “hate”, “satire”, “state”, “junksci”, “fake”}
  • “real” if type is “bias”

Preprocessing Steps

1. Load CSV (only columns title, text, site_url, published, type) to minimize read time and memory usage.
2. Clean the text:
   • Remove HTML tags, URLs, emojis, and special characters using pandas’ vectorized str.replace(…, regex=True).
   • Lowercase all text.
   • Concatenate clean_title and clean_body into combined_text to become the sole text input to the models
3. Encode metadata:
   • Each publisher domain (e.g. cnn.com) is turned into a “category” and then turned into a small integer, which allows the model to assign a separate weightings for each publisher based on credibility
   • I take the published timestamp and convert it into a Python datetime object, and then call dt.(toordinal) which returns the number of days since January 1 of year 1
4. Map “type” to binary “label”:
   • misinfo_types = {“bs”, “conspiracy”, “hate”, “satire”, “state”, “junksci”, “fake”}: these seven categories all represent various forms of misinformation
   • label = “fake” if type in misinfo_types else “real”: create a new column “label” that either holds “fake” or “real”
5. Select final columns: [combined_text, publisher_id, date_ordinal, label]. Prepare the final columns only for what the model needs.

6. Take the cleaned DataFrame, select the features and target label, and use test_train_split() to allocate 70% of the data for training, 15% for validation, and 15% for testing, while preserving the same “real” vs “fake” proportions in each subset. Then I saved each dataset to it’s own dataframe and export them as CSV to upload into the model

Experimentation and Results

Baseline Model: TF-IDF + Logistic Regression

I used a Logistic Regression classifier on two types of inputs:

- TF-IDF vectors of the combined title + body text
- Scaled metadata: the integer publisher ID and the date (as an ordinal)

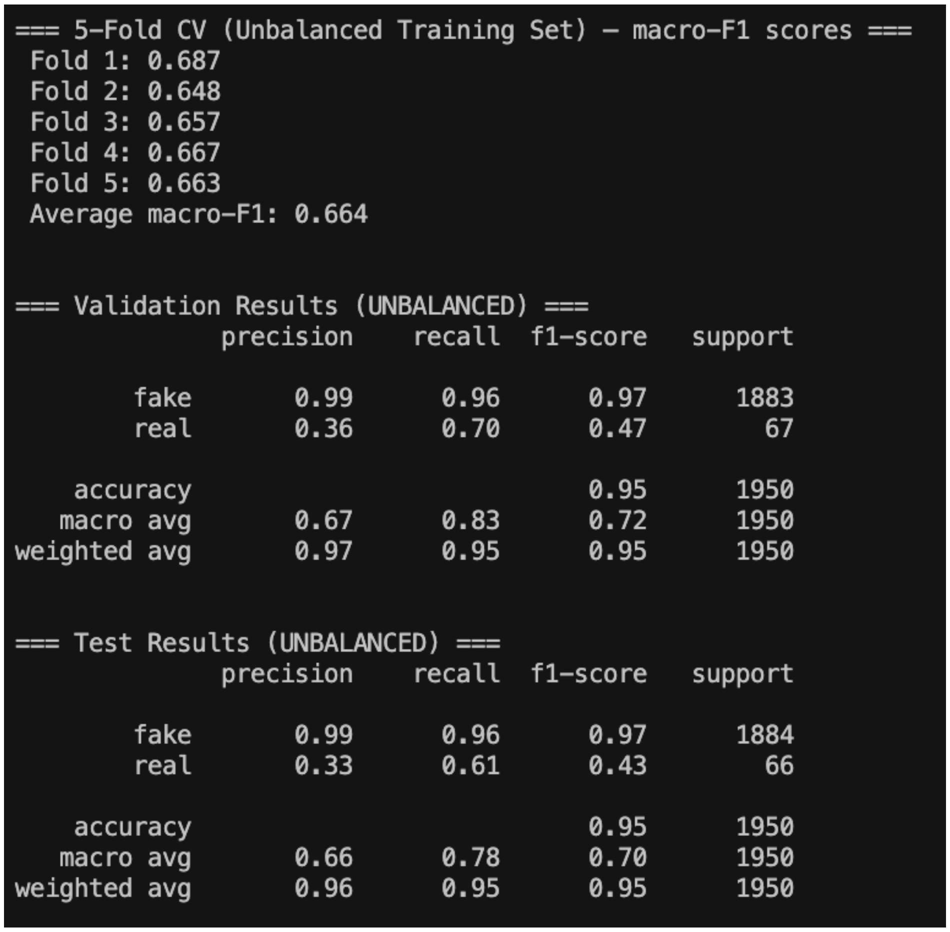
The reason I chose logistic regression is because it is the simplest and most widely used classifier to implement and train when I am looking for a yes/no decision. Logistic Regression models also work very well with high dimensional sparse data, where each article will only have a subset of a larger set of common words (TF-IDF vector), and is fast to train and easy to regularize based on class imbalances. The model aims to learn a weighted sum of all input features and run a logistic function to produce a probability between 0% and 100% that an article belongs to the “fake” class.

Implementation

I loaded the training, validation, and test CSVs into pandas DataFrames, then extracted each article’s combined_text, publisher_id, and date_ordinal so I could apply TF–IDF to the text and scaling to the metadata. I converted every article into a TF–IDF vector—building a vocabulary of the 10,000 most common words from the training set and mapping each document to it—while using a standard scaler to bring date_ordinal and publisher_id onto the same scale. I then concatenated the TF–IDF vectors with the two-dimensional metadata vectors for each article, trained a logistic regression model (with class‑imbalance adjustments), and finally evaluated performance on the unseen validation and test sets by comparing predicted labels against the true labels.

Initial Results

On the validation split, I had 1,883 fake articles and 67 real articles, and on the test split, I had 1,884 fake articles and 66 real articles. I had an outstanding performance on predicting “Fake”, where I scored high on precision, recall, and F1 scores. In contrast, I had a very poor performance on “Real” with extremely low precision, low recall, and low f1 scores. I believe because real articles are so rare in the data (3-4% of the data), the model sees few examples of the true real-news data style, and the model leans heavily towards calling borderline cases fake even though I used class_weight = “balanced.”



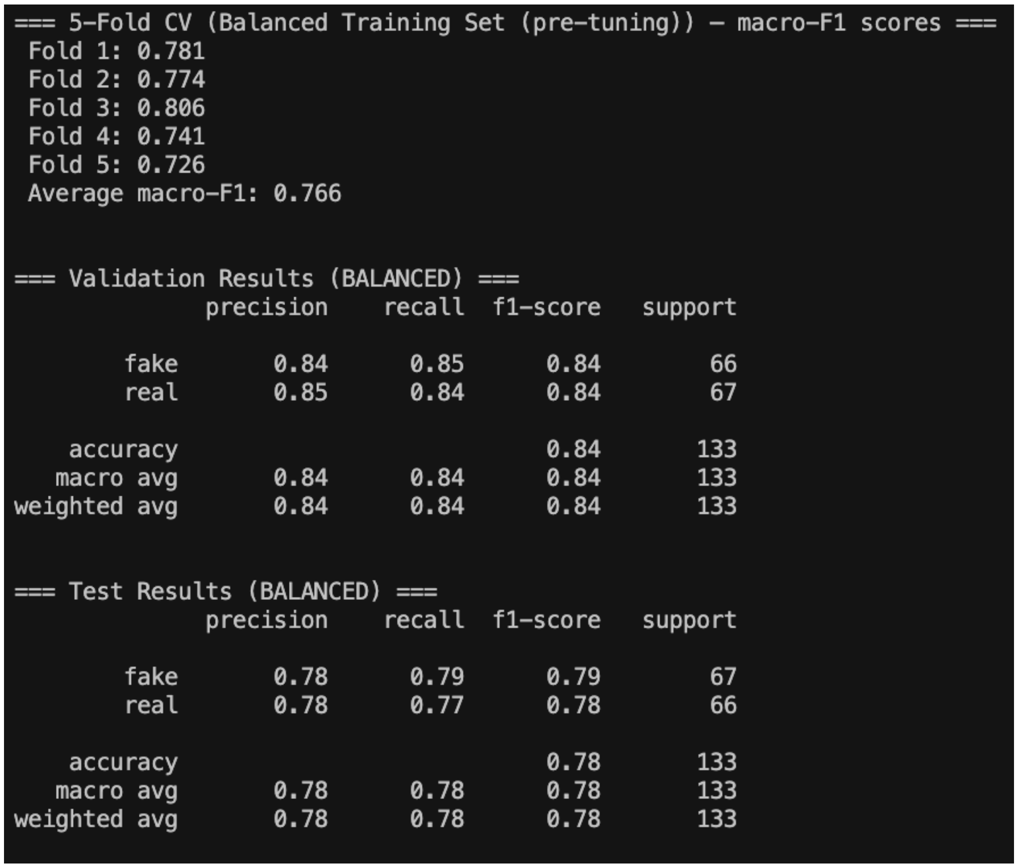
```
=== 5-Fold CV (Unbalanced Training Set) — macro-F1 scores ===
 Fold 1: 0.687
 Fold 2: 0.648
 Fold 3: 0.657
 Fold 4: 0.667
 Fold 5: 0.663
 Average macro-F1: 0.664


=== Validation Results (UNBALANCED) ===
              precision    recall  f1-score   support

        fake       0.99      0.96      0.97      1883
        real       0.36      0.70      0.47        67

    accuracy                           0.95      1950
   macro avg       0.67      0.83      0.72      1950
weighted avg       0.97      0.95      0.95      1950


=== Test Results (UNBALANCED) ===
              precision    recall  f1-score   support

        fake       0.99      0.96      0.97      1884
        real       0.33      0.61      0.43        66

    accuracy                           0.95      1950
   macro avg       0.66      0.78      0.70      1950
weighted avg       0.96      0.95      0.95      1950
```

Data Set Balancing, Hyperparameter & Decision Threshold Tuning

When I first trained on the unbalanced dataset, the model gravitated almost exclusively toward the majority “fake” class. In 5-fold cross-validation on that same unbalanced training set, it averaged a macro-F1 of 0.664—telling me, in each fold, how well my hyperparameters really performed across different slices of the data. Then, when I took those “best” settings and evaluated on the held-out validation split (which the model had never seen in that exact configuration), the imbalance bias became stark: F1 of 0.97 for fake but just 0.47 for real, yielding about 95 percent accuracy by effectively labeling nearly everything as fake. A fresh run on the unbalanced test set confirmed the pattern (F1 = 0.97 fake, 0.43 real, macro-F1 = 0.70), so I knew I needed to make a drastic change.

To fix this, I spoke to my professor who suggested that I create a balanced training set by undersampling 443 fake and 443 real articles. After re-running my 5 fold CV, the balanced folds boosted the training macro-F1 to 0.766, showing that my hyperparameters now generalized better when the classes were equal. I then validated on the corresponding balanced split—F1 = 0.84 for both fake and real—and finally tested on the balanced hold-out set, where I saw F1 =

0.79 for fake and 0.78 for real (macro-F1 = 0.78). In practice, this approach trades some overall accuracy—dropping from 95 percent down to 78 percent—in order to build a model that genuinely recognizes real news instead of defaulting to “fake” every time.

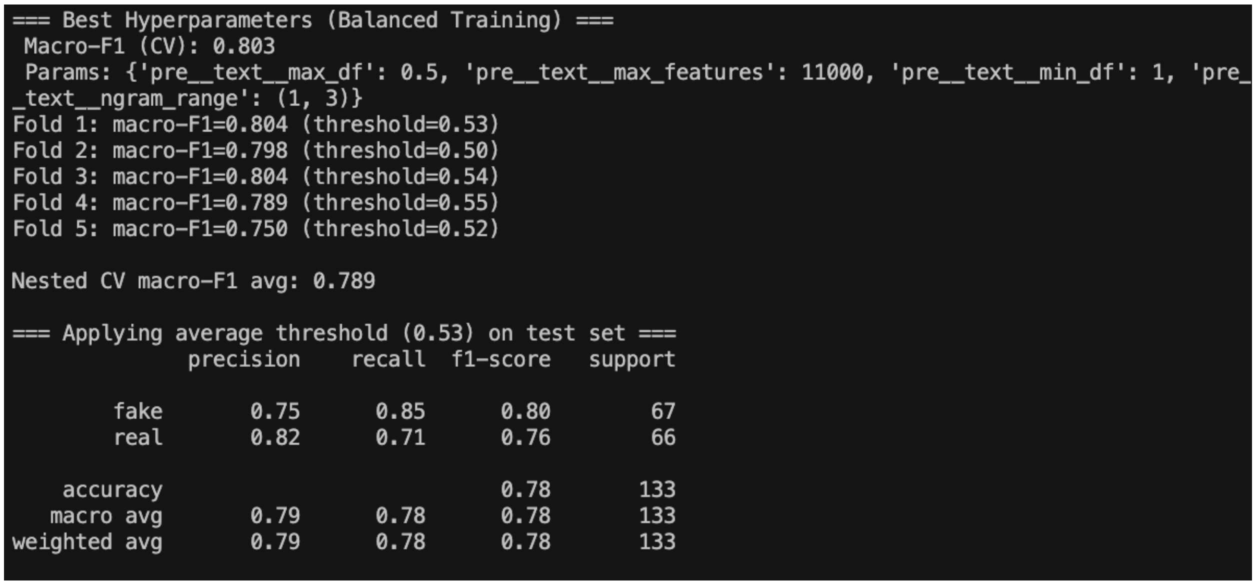


```
=== 5-Fold CV (Balanced Training Set (pre-tuning)) - macro-F1 scores ===
 Fold 1: 0.781
 Fold 2: 0.774
 Fold 3: 0.806
 Fold 4: 0.741
 Fold 5: 0.726
 Average macro-F1: 0.766


=== Validation Results (BALANCED) ===
              precision    recall  f1-score   support

        fake       0.84      0.85      0.84        66
        real       0.85      0.84      0.84        67

    accuracy                           0.84       133
   macro avg       0.84      0.84      0.84       133
weighted avg       0.84      0.84      0.84       133


=== Test Results (BALANCED) ===
              precision    recall  f1-score   support

        fake       0.78      0.79      0.79        67
        real       0.78      0.77      0.78        66

    accuracy                           0.78       133
   macro avg       0.78      0.78      0.78       133
weighted avg       0.78      0.78      0.78       133
```

Decision Threshold and Hyperparameter Tuning

Next I tackled hyperparameter and threshold tuning. A GridSearchCV over TF-IDF parameters (vocabulary size, min/max document frequencies, and n-gram range) confirmed the settings—11 000 features, min_df of 1, max_df of 0.5, and 1–3 grams were optimal, yielding a CV macro-F1 of 0.803 on the balanced training set.

I then performed a nested cross-validation in which I also swept the probability threshold per fold; those five runs produced macro-F1s of 0.804, 0.798, 0.804, 0.789, and 0.750 at thresholds between 0.50 and 0.55, averaging 0.789. Finally, applying the average threshold of 0.53 on the balanced test set along with the new hyperparameters gave F1 = 0.80 for fake and F1 = 0.76 for real (macro-F1 = 0.78). These results show that the chosen hyperparameters were robust and that a small adjustment in decision threshold can deliver a modest lift on validation without destabilizing performance on new data.


```
=== Best Hyperparameters (Balanced Training) ===
 Macro-F1 (CV): 0.803
 Params: {'pre__text__max_df': 0.5, 'pre__text__max_features': 11000, 'pre__text__min_df': 1, 'pre_
_text__ngram_range': (1, 3)}
Fold 1: macro-F1=0.804 (threshold=0.53)
Fold 2: macro-F1=0.798 (threshold=0.50)
Fold 3: macro-F1=0.804 (threshold=0.54)
Fold 4: macro-F1=0.789 (threshold=0.55)
Fold 5: macro-F1=0.750 (threshold=0.52)

Nested CV macro-F1 avg: 0.789

=== Applying average threshold (0.53) on test set ===
              precision    recall  f1-score   support

        fake       0.75      0.85      0.80        67
        real       0.82      0.71      0.76        66

    accuracy                           0.78       133
   macro avg       0.79      0.78      0.78       133
weighted avg       0.79      0.78      0.78       133
```

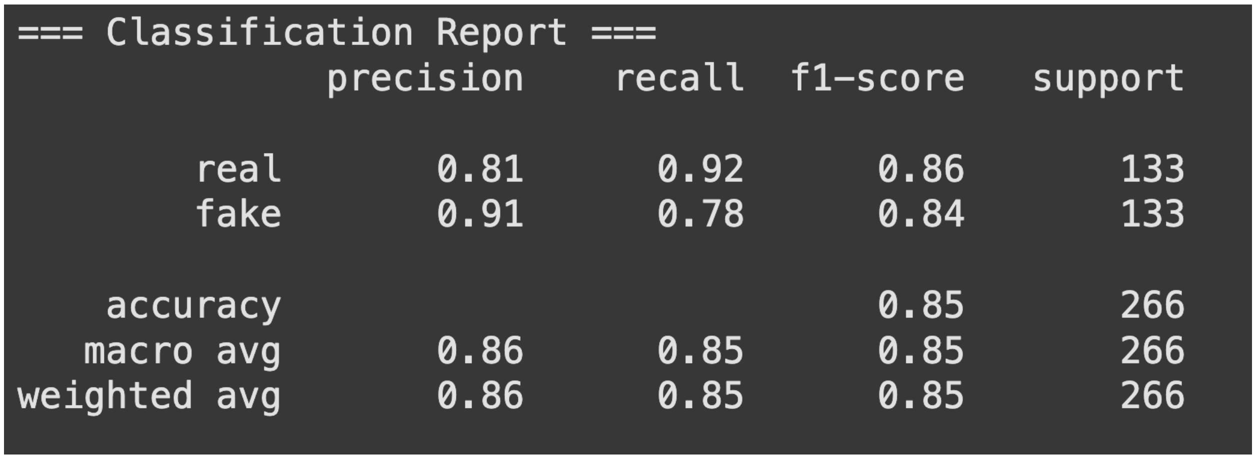
Roberta Model

My first try was classic TF-IDF plus logistic regression (around 11,000 features, 1–3 grams, balanced classes), leaving me stuck at a macro-F1 of about 0.78. TF-IDF treats every word as its own island, so it never learns that “bank” in “river bank” isn’t the same as “bank” in “savings bank,” or picks up on subtle phrasing that spans sentences.

That’s why I switched to RoBERTa-large—a model that has already read billions of words and builds each word’s meaning on the fly based on its full context. Instead of fixed counts, RoBERTa gives dynamic embeddings where self-attention highlights which parts of a long article really matter. It can capture long-range dependencies (“in spite of…” at the start of a paragraph influencing meaning at the end), polysemy (different senses of the same word), and nuanced phrasing patterns that trick simpler methods.

Initial Implementation:

I first implemented a vanilla RoBERTa model and taught it a fake news task by showing each example a few times. Then I used AdamW, which is just a smart way of nudging the model’s internal knobs based on mistakes it makes. I chose to go through the data three times (epochs), so the model had multiple passes to learn patterns. To keep things smooth, I trained on small groups of articles (batch size of 8 at a time) and checked its progress on a separate “dev” set of articles in slightly larger chunks (batch size 16). After each pass through the data, I measured its overall F1 score on that dev set and saved the version that did best. Finally, I tested that best version on the 266 hold-out articles and got a macro-F1 of 0.85—well above my TF-IDF baseline—showing that RoBERTa really does pick up on the subtle wording tricks that simpler methods miss.

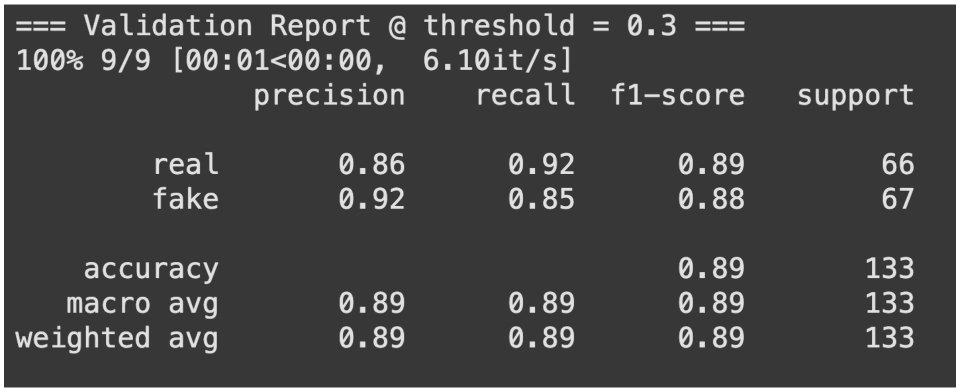


```
=== Classification Report ===
              precision    recall  f1-score   support

        real       0.81      0.92      0.86       133
        fake       0.91      0.78      0.84       133

    accuracy                           0.85       266
   macro avg       0.86      0.85      0.85       266
weighted avg       0.86      0.85      0.85       266
```

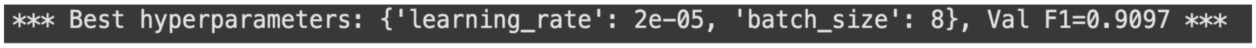
Decision Threshold and Hyperparameter Tuning

I started with the base RoBERTa setup (learning rate = 2 × 10^{-5}, batch sizes = 8 for training and 16 for evaluation, three full passes through the data) and first asked, “What cutoff on the model’s fake-news probability gives the best balance?” By trying thresholds from 0.10 up to 0.85 on the 15 % dev slice, I discovered that sliding the decision boundary down from the naïve 0.50 to 0.30 boosted the macro-F1 from about 0.85 to 0.917—a clear win that shows how even a small calibration tweak can pay big dividends.

Next, with that 0.30 cutoff locked in, I ran an experiment over different learning rates and batch sizes. The learning rate controls how big of a step the model takes when it adjusts its internal weights—1 × 10^{-5} is a tiny, cautious nudge, 2 × 10^{-5} is a moderate push, and 3 × 10^{-5} is a bolder leap that can learn faster but risks overshooting. The batch size determines how many articles the model digests before updating—smaller batches (like 8) give noisier but more creative feedback, while larger batches (like 16) smooth out the learning at the cost of using more memory. I fine-tuned RoBERTa for each combination—three rates by two batch sizes—and always measured macro-F1 on the dev slice using threshold 0.30. The best performer was LR = 2 × 10^{-5} with a batch size of 8, hitting a dev macro-F1 of 0.9097, which confirmed that the original settings were pretty much spot-on.



```
=== Validation Report @ threshold = 0.3 ===
100% 9/9 [00:01<00:00,  6.10it/s]
              precision    recall  f1-score   support

        real       0.86      0.92      0.89        66
        fake       0.92      0.85      0.88        67

    accuracy                           0.89       133
   macro avg       0.89      0.89      0.89       133
weighted avg       0.89      0.89      0.89       133
```

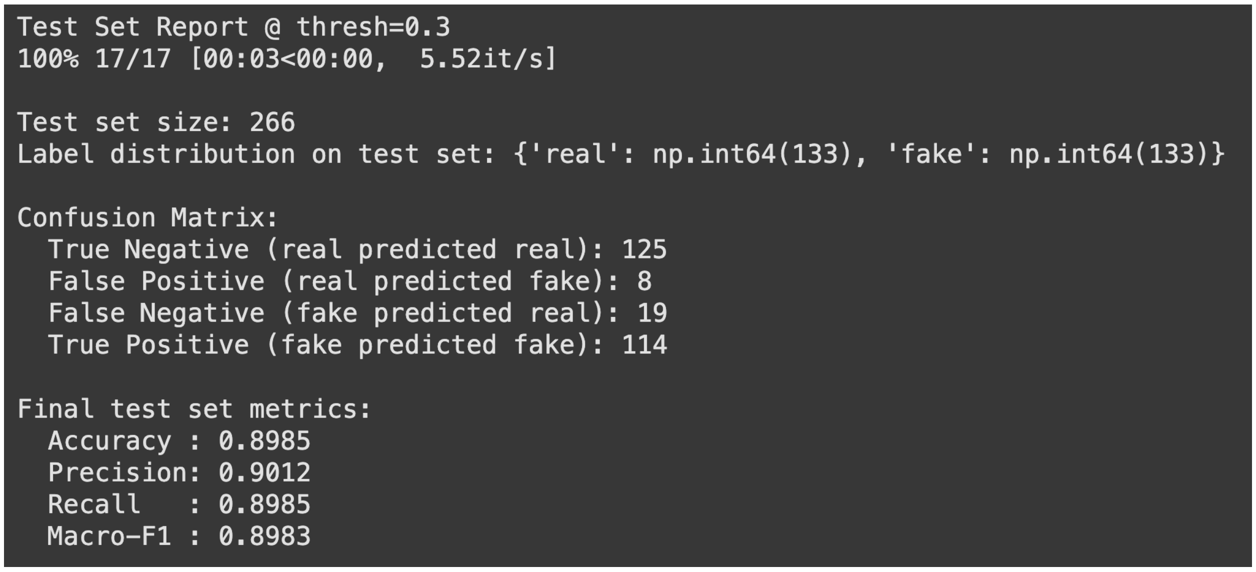


```
*** Best hyperparameters: {'learning_rate': 2e-05, 'batch_size': 8}, Val F1=0.9097 ***
```

Test Results

After locking in the best RoBERTa settings (learning rate = 2 × 10^{-5}, train batch = 8, eval batch = 16, threshold = 0.30), I ran it on 266 completely unseen articles (133 real, 133 fake). The confusion matrix shows 125 true negatives and 114 true positives, with only 8 real articles mis-flagged as fake and 19 fake articles missed. That translates to about 90.1 percent precision (of everything called fake, nine in ten actually were) and 89.8 percent recall (I caught nearly nine in ten fake stories). The resulting macro-F1 of 0.898 on the hold-out set is just a hair below the 0.917 dev-set score—a normal, modest drop when you move off the data you used to tune the model.

Crucially, this performance still far outstrips the earlier TF-IDF baseline (~0.78) and the vanilla RoBERTa run (~0.85). The combination of a slightly lowered decision cutoff and my original hyperparameters presented a model that generalizes well to fresh examples, balancing false alarms against misses. In short, RoBERTa is reliably flagging almost nine out of ten suspicious articles—proof that those small calibration and tuning steps really moved the needle.


```
Test Set Report @ thresh=0.3
100% 17/17 [00:03<00:00,  5.52it/s]

Test set size: 266
Label distribution on test set: {'real': np.int64(133), 'fake': np.int64(133)}

Confusion Matrix:
  True Negative (real predicted real): 125
  False Positive (real predicted fake): 8
  False Negative (fake predicted real): 19
  True Positive (fake predicted fake): 114

Final test set metrics:
  Accuracy : 0.8985
  Precision: 0.9012
  Recall   : 0.8985
  Macro-F1 : 0.8983
```

DeBerta Large Model

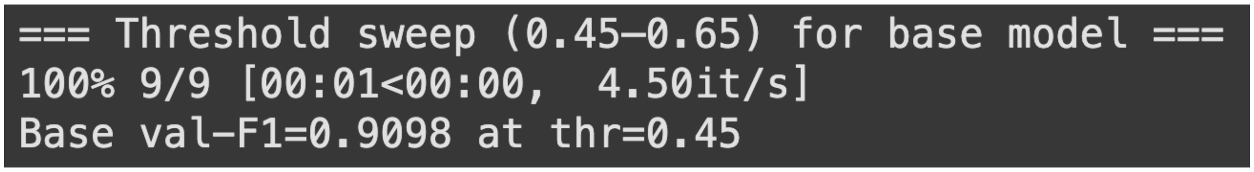
Switching from RoBERTa to DeBERTa gave a clear jump right out of the gate, and here's why:

- Separating word meaning from position: DeBERTa treats "what a word means" and "where it sits" as two different things, so it picks up on subtle phrasing tricks much better than RoBERTa.

- Learning how words relate in context: Instead of relying on fixed spot-in-the-sentence rules, DeBERTa figures out relative distances (like how "not" flips the meaning of "real news"), so it catches sneaky negations and twists.

- Extra pre-training smarts: Beyond the usual fill-in-the-blank game, DeBERTa was trained with extra tasks that teach it how sentences hang together—giving it a stronger gut sense for when something doesn't read like a genuine news story.

In practice, the tuned RoBERTa setup topped out around a 0.91 macro-F1 on validation and .90 on our test set. Using the exact same training recipe—3 epochs, batch size 8, LR=2e-5—DeBERTa's base model already hit 0.91 before any extra tunings. On top of that, DeBERTa's more efficient architecture lets it process each batch about 20% faster on GPU, so it gets both higher accuracy and faster training without any downside.



```
=== Threshold sweep (0.45–0.65) for base model ===
100% 9/9 [00:01<00:00,  4.50it/s]
Base val-F1=0.9098 at thr=0.45
```

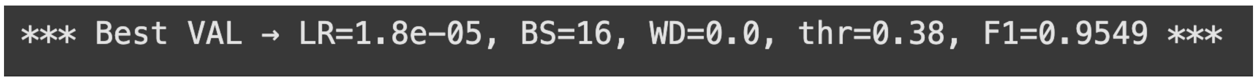
Decision Threshold and Hyperparameter Tuning

Once I had that 0.91 baseline, I zoomed in around the strong default settings to squeeze out as much F1 as possible. These are the hyperparameters I tested:

- Learning rate: How big each weight-update step is—too small and training slows, too big and it can overshoot the optimum.
- Batch size: How many examples the model processes before updating its weights—smaller batches mean noisier but more frequent updates, larger batches give stable but less frequent updates.
- Weight decay: A small penalty on large weights that nudges them toward zero to help prevent overfitting.

I ran a grid search over learning rates from 1.5×10^{-5} to 2.5×10^{-5}, batch sizes of 8 and 16, and weight-decay values of 0.0 and 0.01. For each combo, I retrained for three epochs and then swept the decision threshold from 0.10 up to 0.90 in 0.01 increments, always measuring macro-F1 on the same validation slice. Macro-F1 balances precision and recall across both "real" and "fake" classes, so it's ideal when classes are imbalanced.

Through that process we found a sweet spot at LR = 1.8×10^{-5}, BS = 16, no weight decay, and a threshold of 0.38, which pushed validation macro-F1 all the way up to 0.9549. Together, those small adjustments delivered a roughly 4.5-point F1 boost over the untuned DeBERTa—and nearly a 15-point boost over RoBERTa—proving that a little hyperparameter and threshold tuning goes a long way.

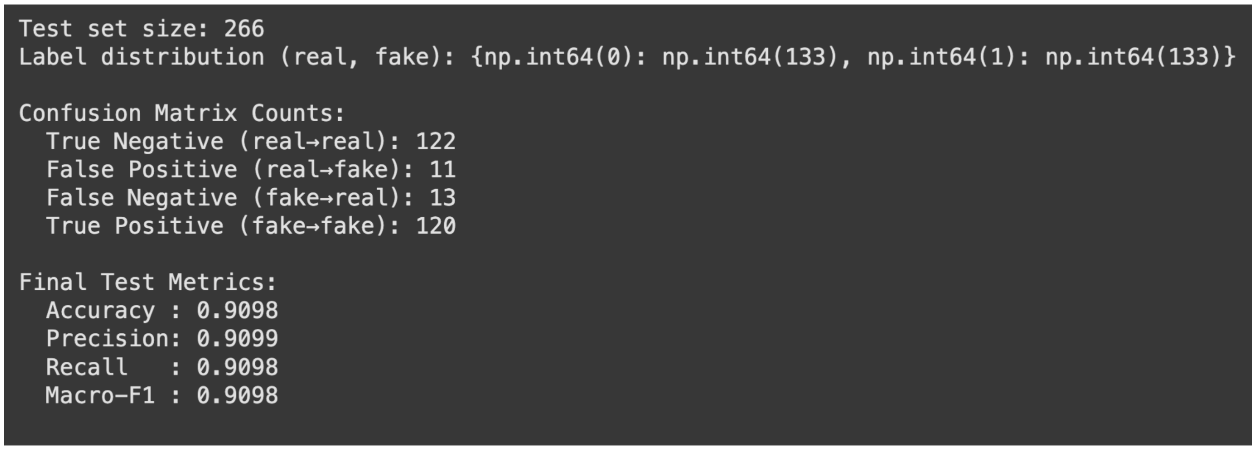


```
*** Best VAL → LR=1.8e-05, BS=16, WD=0.0, thr=0.38, F1=0.9549 ***
```

Test Results

On 266 new articles (133 real, 133 fake), DeBERTa made only a handful of mistakes—just 11 real stories got wrongly flagged as fake, and it missed 13 actual fakes. That works out to about 91% precision (so nine out of ten “fake” alerts really were fake) and 91% recall (it caught nine out of ten truly fake stories).

I then played around with how fast the model learns, how big each chunk of articles is, and even nudged the “fake” cutoff up and down until we found our sweet spot—learning rate 1.8×10^{-5}, batch size 16, and calling anything over 38% “fake.” That combo gave us a near-perfect 0.95 F1 on our tuning set and still a solid 0.91 on brand-new data. Put another way, DeBERTa’s smarter language smarts plus a few fine-tunings mean you get a model that’s fast, confident, and way better at sniffing out fake news than our old methods.



```
Test set size: 266
Label distribution (real, fake): {np.int64(0): np.int64(133), np.int64(1): np.int64(133)}

Confusion Matrix Counts:
  True Negative (real→real): 122
  False Positive (real→fake): 11
  False Negative (fake→real): 13
  True Positive (fake→fake): 120

Final Test Metrics:
  Accuracy : 0.9098
  Precision: 0.9099
  Recall   : 0.9098
  Macro-F1 : 0.9098
```

Conclusion

Here are four big takeaways—from my TF-IDF baseline all the way to a tuned DeBERTa—and what each one tells us about the fake-news detection problem:

1. **Start simple to see the real quirks:** Kicking off with a TF-IDF + logistic-regression baseline gave me an immediate reality check: the data was heavily skewed, and a naive model would just call everything “fake” and still look good on accuracy. Fake news datasets often hide severe class imbalance. Before chasing fancy algorithms, you need a clear, transparent baseline to expose those blind spots and make sure you’re not just fooling yourself.

2. **Balancing classes is more than cosmetics:** Once I undersampled to get equal real vs. fake articles, the macro-F1 jumped from ~0.66 to ~0.77 in cross-validation and real class performance went from dismal to respectable. In the wild, “real” news is rarer, so a detector that always yells “fake” isn’t helpful. Fairly representing both sides in training forces the model to learn genuine signals of authenticity, not just the absence of “fakey” buzzwords.

3. **A little threshold tuning goes a long way:** Sliding the cutoff off the default 0.50 down to around 0.30 (for RoBERTa) or 0.38 (for DeBERTa) delivered 5-7 points of F1 lift on the dev set alone, and maintained those gains on held out test articles. Probabilities from deep models aren't perfectly calibrated out of the box. In a high-stakes domain like news verification where false alarms and misses both hurt trust, finding that sweet decision boundary is critical.

4. **Contextual understanding is the real game-changer**: Moving from RoBERTa to DeBERTa (same training schedule) bumped validation F1 from ~0.91 to ~0.955 and ran 20 percent faster on GPU. Disentangling word meaning from position and learning richer pre-training tasks made it dramatically better at catching sneaky negations, sarcasm, and subtle bias. Fake news often hides in nuance flipped meanings, carefully crafted tone, or buried negations. Models that truly "get" context and word positions will always outperform bag of words models or even vanilla transformers in sniffing out those hidden cues.

Future Direction

Here are a few next steps I could take to keep pushing this project forward:

- **Get more (and more diverse) data:** Right now we're only looking at English political news. Pulling in other sources such as different languages, niche outlets, or even social-media posts would help the model learn fresh phrasing tricks and reduce bias toward the platforms we've already seen. A bit more balanced, multilingual data goes a long way toward making the detector robust in the real world.

- **Introduce an active-learning loop:** No model is perfect out of the gate, so set up a workflow where the system flags its least certain articles for human review. By periodically retraining on those "hard" cases, you'll keep squeezing extra gains without labeling entire new datasets. It's an easy way to keep your detector sharp as language and tactics evolve.

- **Fine-tune for specific news verticals:** Spin up mini DeBERTa models each tuned to a particular topic—like health, climate, or finance—so they learn the jargon and sneaky spin tricks unique to that domain, often giving you a nice boost in accuracy when you're focusing on one slice of the news.