Hidden Secrets Detection using Hybrid Multi-Task Transformers

1. Abstract

Objective: Accidental leakage of secrets such as API keys, access tokens, and credentials in source code repositories remains a major security risk. Existing tools like Gitleaks and TruffleHog rely primarily on pattern matching and entropy-based heuristics, which makes them brittle under obfuscation and context manipulation.

Key Innovations: This project presents a hybrid multi-task learning approach for secret detection that jointly performs binary classification, secret span localization, and entropy regression using a CodeBERT encoder. The model is trained on a carefully constructed dataset combining real-world leaks, synthetic positives, and hard negatives, with explicit correction of span annotation biases.

Core Results: Extensive evaluation demonstrates that the proposed approach achieves a recall of 83.9% under adversarial conditions, a 23% improvement over regex-based scanners, while maintaining competitive precision. The results highlight the importance of contextual understanding over surface-level string patterns for reliable secret detection in modern codebases.

2. Dataset Construction

The construction of a balanced and uniform dataset is the major highlight of this project. This dataset was constructed with particular attention to bias, coverage, and adversarial robustness.

2.1 Real Positive Samples

- Extracted from public GitHub repositories using Gitleaks and TruffleHog
- Identified using Gitleaks and TruffleHog
- Filtered to remove false positives and balance the number of heuristic samples

- Broad categories like "Parsehub" or "Yelp" were mapped to a high-entropy generic-api-key rule to maintain a focused training set while ensuring coverage.

2.2 Real Negative Samples

- 50% : Legitimate code snippets without secrets
- 50% : Contextually similar to positives
- Includes identifiers, hashes, and configuration values

2.3 Synthetic Positives

Secrets were programmatically injected into code templates using:

- Encoding transformations (base64, hex, URL encoding)
- Structural transformations (string concatenation, reversal, noise injection)
- Positional variations (environment variables, indirection, comments)

2.4 Hard Negatives

High-entropy but non-secret strings (injected into contextually non-secret templates) such as:

- UUIDs
- Hashes
- Request IDs
- Checksums

These were crucial for preventing entropy-based shortcuts.

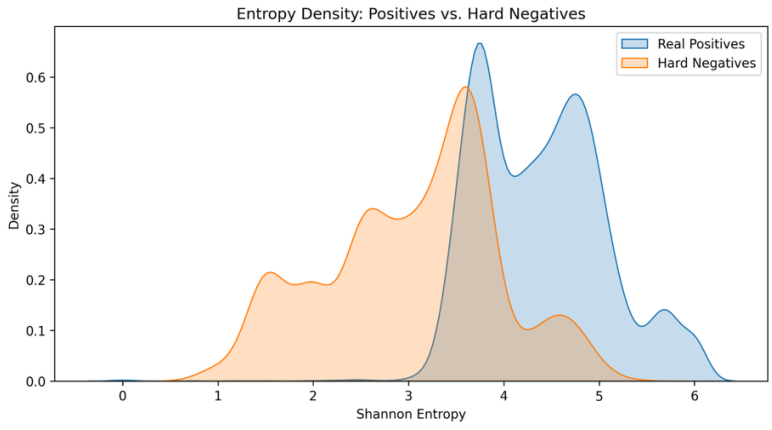


Fig - Overlapping entropy distributions of real secrets and hard negatives, illustrating why entropy alone is insufficient for detection.

2.5 Span Bias Correction

Initial analysis revealed a heavy bias toward secret span start = 0. To correct this:

- Suitable prefix padding was applied
- Structural wrappers were introduced
- Span distributions were rebalanced

This prevented the model from learning positional shortcuts.

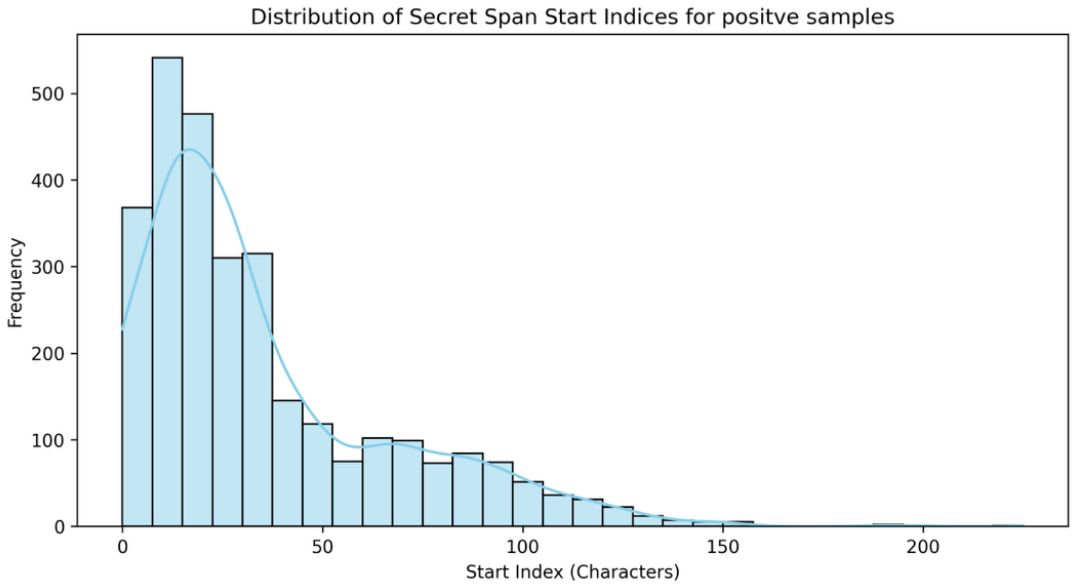


Fig - Uniform distribution of secret span start positions across the sequence length after bias correction and prefix padding

2.6 Dataset statistic summary

- Frequency Distribution:

Total Samples	8496
Positive Samples	4723 (55.59%)
Negative Samples	3773 (44.41%)

- Source distribution:

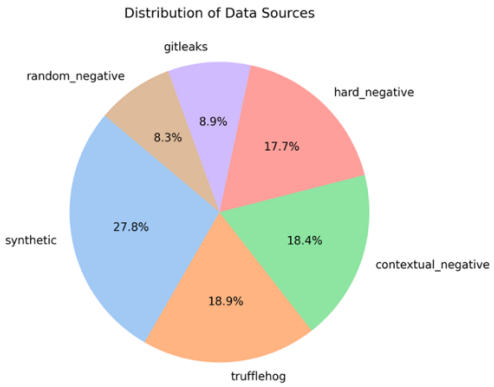


Fig - Distribution of data sources across the 8,496 samples, highlighting the balance between synthetic and real-world leaks

- Train-Validation-Test split was done as **70-15-15** (percentage)

3. Model Architecture

3.1 Encoder

The model uses **CodeBERT**, a transformer pretrained on large-scale source code corpora, as the shared encoder (as its pre-training on 6.4M functions provides a superior semantic understanding of variable assignments compared to standard BERT)

3.2 Task Heads

- Classification Head: Binary classification of secret presence
- Span Heads: Token-level start and end prediction
- Entropy Head: Regression head predicting entropy

3.3 Multi-Task Loss

The total loss is defined as:

$$\text{Loss} = c_1 \cdot \text{BCE} + c_2 \cdot \text{MSE} + c_3 \cdot \text{Loss}_{\text{span}}$$

where, BCE : Binary Cross Entropy

MSE : Mean Squared Error

$\text{Loss}_{\text{span}}$: Span Loss

c_1, c_2, c_3 : Training weights

For warmup epochs, I used -

c_1 = 1.0, c_2 = 0.3, c_3 = 0.1

and for finetune-epochs, I used -

c_1 = 1.0, c_2 = 0.5, c_3 = 0.2

The increase in weight forces the model to focus on the secondary signals (localization and randomness) only after the encoder has stabilized on basic binary classification.

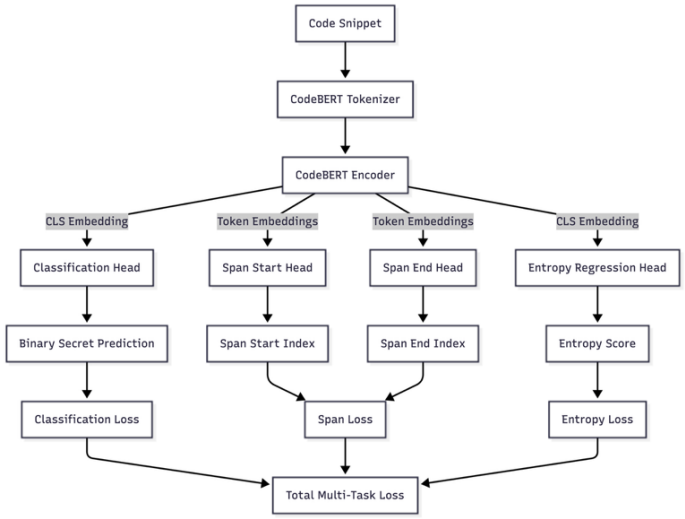


Fig - Architecture diagram for the hybrid multi-task transformer

4. Training the Model

4.1 Training strategy

 # Stage 1: Warm-up

- Encoder frozen
- Task heads trained
- Prevents early catastrophic forgetting

 # Stage 2: Fine-tuning

- Full model unfrozen
- Lower learning rate
- Allows deep semantic adaptation

 # Span Masking

Span loss is computed only for samples with valid span annotations, avoiding noise from partial labels.

4.2 Training results

After 2 warmup and 5 finetune epochs -

- Train Loss: 0.0371 (classification: 0.0035, span: 0.0062, entropy: 0.1523)
- Val Loss: 0.0415 (classification: 0.0199, span: 0.0063, entropy: 0.0925)
- Val Metrics: Acc=0.9969, P=0.9986, R=0.9958, F1=0.9972
- Test Loss: 0.0652 (classification: 0.0373, span: 0.0175, entropy: 0.0958)
- Test Metrics: Acc=0.9945, P=0.9972, R=0.9929, F1=0.9951

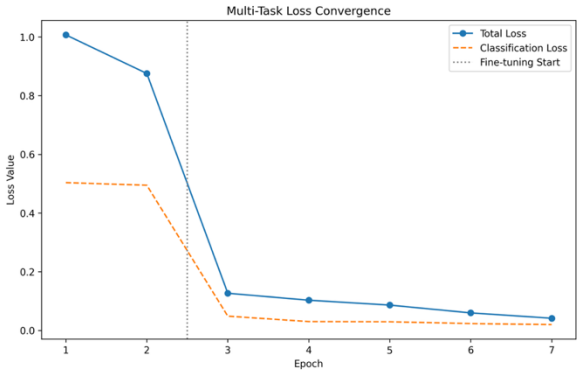


Fig - Convergence of total and classification loss across warmup and fine-tuning stages

5. Baseline Model

A baseline classifier using the same CodeBERT encoder but only binary classification was trained for comparison. This establishes the value of span localization and entropy supervision.

5.1 Training Results

After 2 warmup and 1 finetune epochs -

- Train Loss: 0.0599
- Val Loss: 0.0124
- Val Metrics: Acc=0.9969, P=0.9972, R=0.9972, F1=0.9972
- Test Loss: 0.0160
- Test Metrics: Acc=0.9961, P=0.9972, R=0.9958, F1=0.9965

By looking at the training results, both the models seems to be similar but later during adversarial testing, we will prove how the hybrid model is better than the baseline model.

6. Evaluation Methodology

6.1 Standard Evaluation

Metrics: Accuracy, Precision, Recall, F1-score, MAE (for entropy) and exact match (for span)

6.2 Adversarial Evaluation

To evaluate robustness, adversarial datasets were constructed using **546** adversarial samples :

- Seen obfuscations (used during training)
- Unseen obfuscations (never seen during training)

7. CLI Scanner

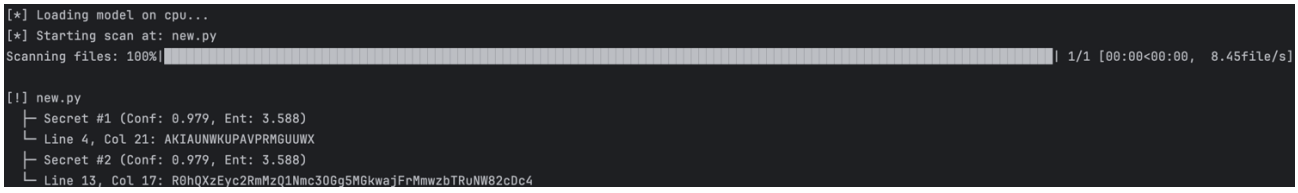
A CLI scanner was implemented to:

- Scan files or directories
- Output detected secrets with spans
- Basic command structure: (detailed usage given in README.md)

```
python scan_secrets.py <path> --model <checkpoint_path> [options]
```

"The system utilizes a transformer-based span head to identify semantically relevant 'danger zones' containing potential secrets. These predicted regions act as semantic anchors for a hybrid extraction engine, which performs targeted refinement using deterministic regex logic. This approach combines deep-learning contextual intelligence with precise string recovery, ensuring robust detection in complex, obfuscated, or multi-language environments."

Sample CLI output:



8. Results and Conclusion

8.1 Results of the adversarial benchmarking

Model	Precision	Recall	F1 Score
Hybrid Multi-Task model	**0.812**	**0.839**	**0.825**
Baseline-Classifier	**0.845**	**0.681**	**0.755**
Gitleaks (Regex)	**1.000**	**0.538**	**0.700**

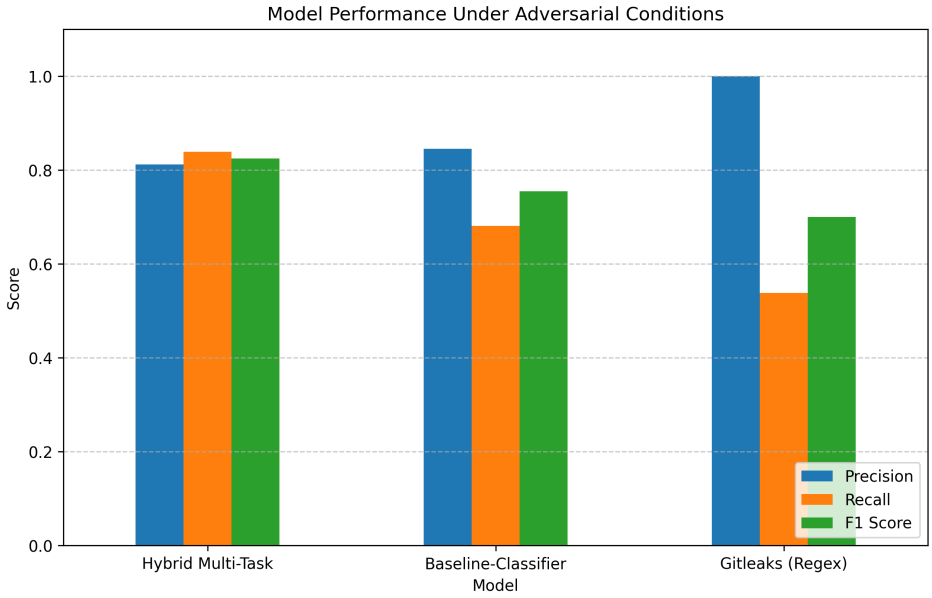


Fig - Visual representation of comparative performance of the Hybrid Multi-Task model against the baseline and Gitleaks under adversarial obfuscation

8.1 Additional results of the hybrid model

Task	Metric	Value
Span Localization	Exact Match Accuracy (in %)	**90.63**
Span Samples Evaluated	Count	**288**
Entropy Regression	MAE	**0.714**

8.3 Conclusion

1. **Best Recall-Precision Balance Under Adversarial Conditions :**

   While Gitleaks achieves perfect precision, it misses nearly half of the real secrets under adversarial transformations.
   The hybrid model maintains high recall (83.9%) while keeping strong precision (81.2%), resulting in the highest F1 score overall.
   It is important to note that **Recall** is a really important factor of comparison here as higher number of false positives will lead to more secret exposure.

2. **Multi-Task Learning Enables Robust Generalization :**
   The baseline classifier struggles under adversarial obfuscations, with recall dropping to 68.1%. In contrast, the hybrid model benefits from:
   - Span localization supervision
   - Entropy regression as an auxiliary signal

   This forces the shared encoder to learn **semantic context**, not just surface patterns.

3. **The hybrid model (unlike regex-based tools and simple classifiers) :**
   - Pinpoints exact secret locations (90% exact span match)
   - Estimates entropy, enabling prioritization of high-risk leaks

"The proposed hybrid multi-task model consistently outperforms both a strong classification baseline and state-of-the-art regex-based tools under adversarial conditions, achieving the best balance between recall and precision while additionally providing accurate secret localization and entropy estimation."

Appendix

- Real Positive Sample :

```
access_key: AKIAWFWEKJPSAOH5YHFT
```

- Synthetic Positive Sample (concatenation-split) :

```
config["secret"] = ""Kfwvp07YsGJhh" + "EvGYQmL7zFYe9M""
```

- Hard Negative Sample :

```
build: checksum: "9d597d8183c7b9f2943c493070bf6177"
```

- Contextual Negative Sample :

```
op.execute("DROP TYPE element_type;")\n    op.drop_table('naics_structures')
```

- Random Negative Sample :

```
let high = m.locals.add(ValType::I64);\n
```

- Adversarial Positive Sample (Unseen - try_except_fallback) :

```
ACCESS_TOKEN = try:\n    key = load_key()\nexcept Exception:\n    key =
"gA5JblekSiUvyDCPZHEJEFDwyGiXmkfGi"
```

- Adversarial Negative Sample :

```
checksum = "9e107d9d372bb6826bd81d3542a419d6"
```