National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cybersecurity

Sai Raghu Ram Raghavapurapu
Student ID: 23380195

School of Computing
National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sai Raghu Ram Raghavapurapu
Student ID:	23380195
Programme:	MSc in Cybersecurity **Year:** 2025-2026
Module:	Research Practicum
Lecturer:	Jawad Salahuddin
Submission Due Date:	28-01-2026
Project Title:	Multi-Model Embedding Fusion for Phishing URL Detection: Combining GPT-3.5 Semantic Understanding with BERT Contextual Analysis for Enhanced Threat Classification
Word Count:	187 **Page Count:** 7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sai Raghu Ram Raghavapurapu
Date:	28-01-2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

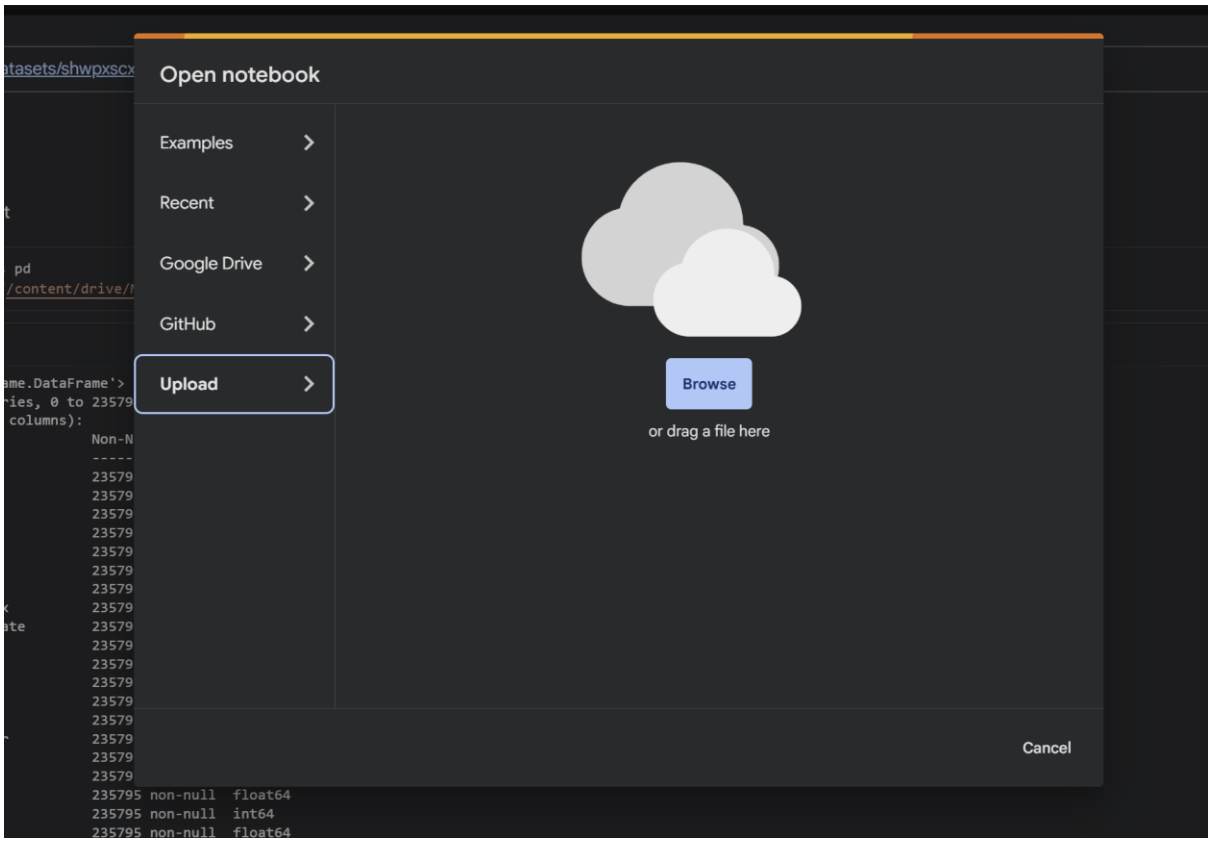
Configuration Manual

Sai Raghu Ram Raghavapurapu
23380195

1. Integrating Google Drive storage with Google Colab.

Step 1 : Launching the colab environment (navigating to google colab website)
Step 2: Sign in into the google account
Step 3:uploading the ipynb file /notebook
Step 4: Enabling the GPU Acceleration (in this change the runtime)
Step 5 : change the hardware accelerator to T4 GPU

Do the above steps and SAVE the selection and the google colab session is set

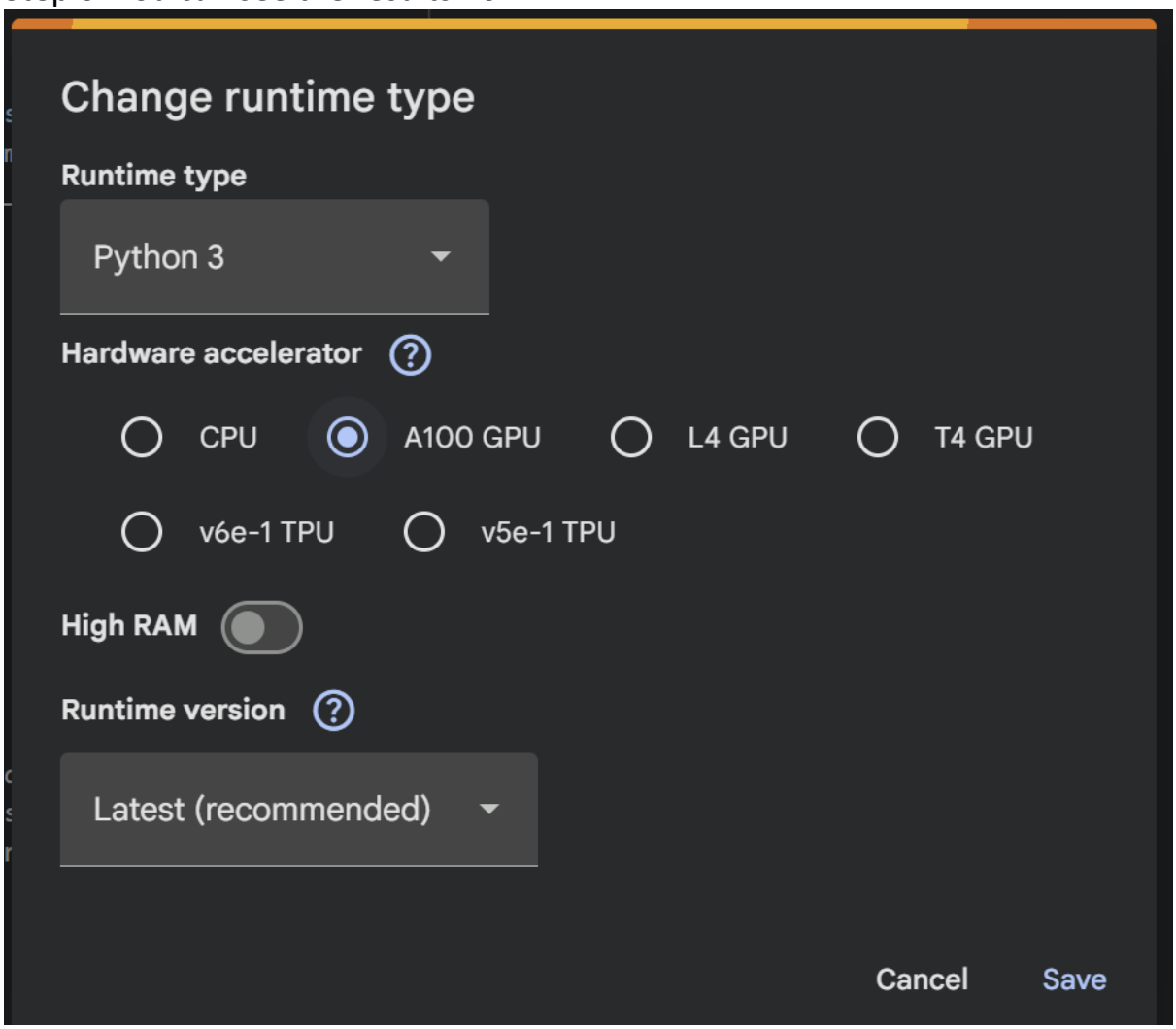


2. Bootstrapping the session

Step 1: open the google colab session
Step 2:Sign in to your Gmail
Step 3:start a new note book
Step 4 : Activate the GPU acceleration
Step 5:Create a folder on the website

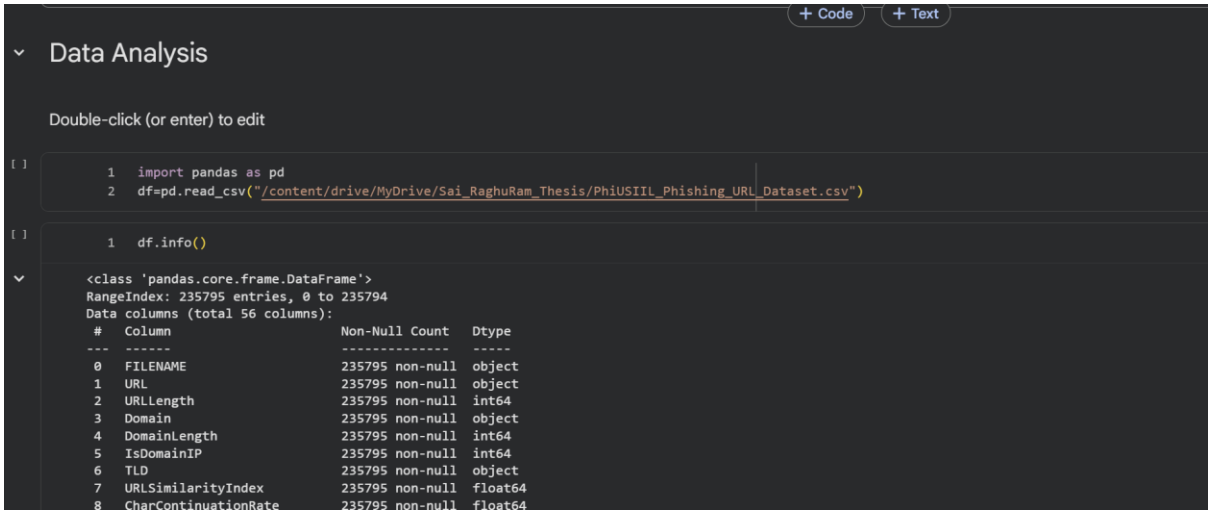
1. Upload the document

Step 6: You can see the results now



3. Fetching and loading the dataset

Step 1: :Loading the data set (PhiUSIIL_Phishing_URL_Dataset)(replace the path)



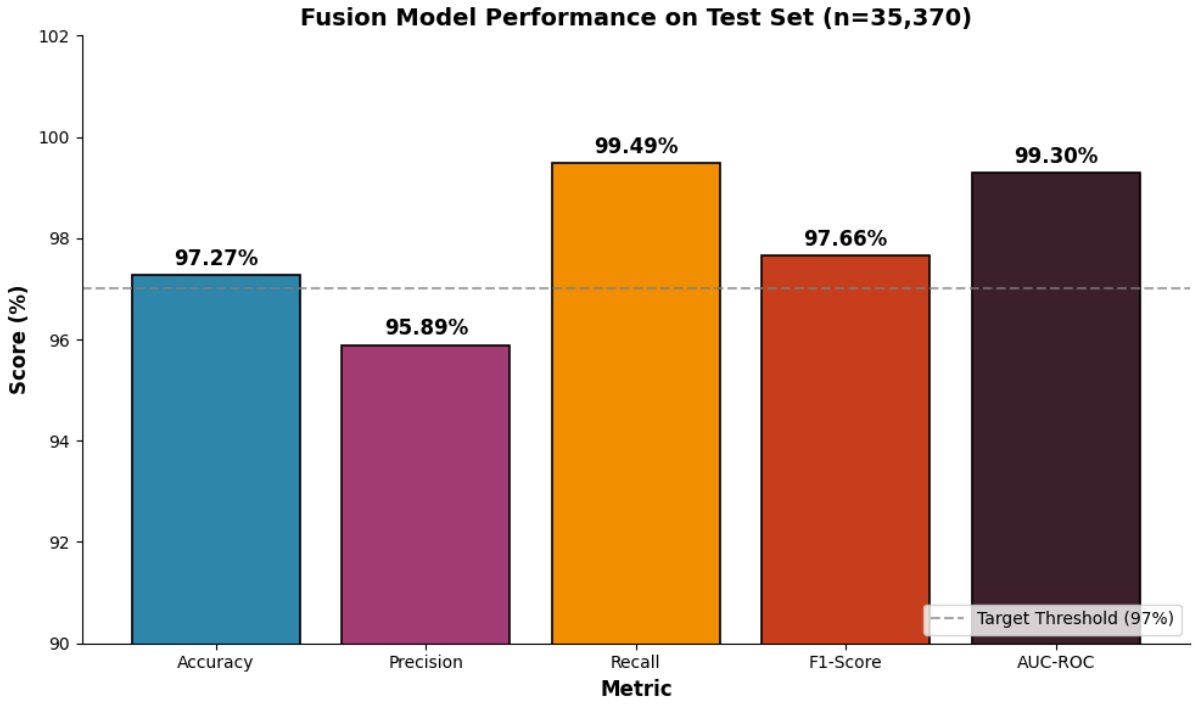
Step 2: Load the dataset (replace the api_key)

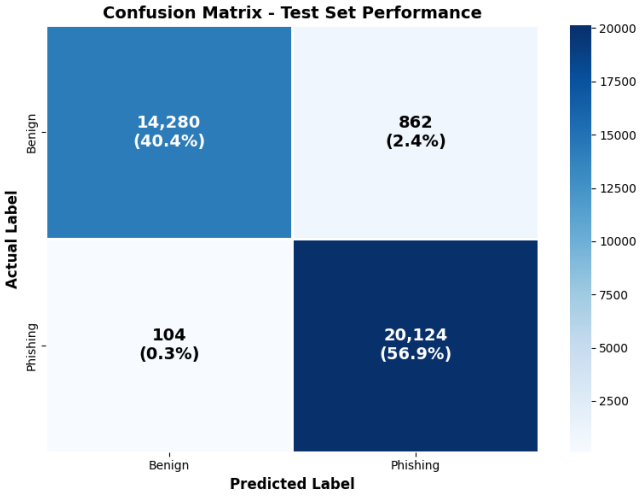
4. Final Results

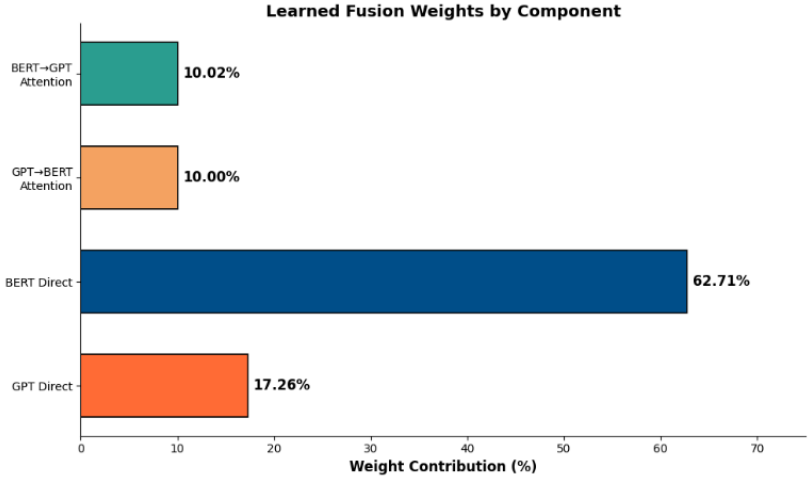
```python

# ============================================================================
#  7: GPT-3.5 Embedding Extraction Functions (Enhanced Large Model)
# ============================================================================

def get_gpt35_embeddings_batch(texts, model="text-embedding-3-large", batch_size=2048):
    """
    Extract GPT-3.5 embeddings for a batch of texts
    Uses OpenAI's text-embedding-3-large model (3,072 dimensions)
    Enhanced with larger model for better semantic understanding
    """
    embeddings = []

    for i in tqdm(range(0, len(texts), batch_size), desc="GPT-3.5-Large Embedding Batches"):
        batch = texts[i:i + batch_size]

        try:
            # Call OpenAI API
            response = openai.embeddings.create(
                input=batch,
                model=model
            )

            # Extract embeddings
            batch_embeddings = [item.embedding for item in response.data]
            embeddings.extend(batch_embeddings)

            # Rate limiting - small delay to avoid hitting limits
            time.sleep(0.5)

        except Exception as e:
            print(f"\nError in batch {i//batch_size}: {str(e)}")
            # Retry with smaller batch size
            for text in batch:
                try:
                    response = openai.embeddings.create(
                        input=[text]
# ============================================================================
# 8: DomURLs_BERT Embedding Extraction Functions
# ============================================================================

def load_domurlsbert_model():
    """
    Load DomURLs_BERT model and tokenizer
    """
    print("Loading DomURLs_BERT model...")
    tokenizer = AutoTokenizer.from_pretrained("amahdaouy/DomURLs_BERT")
    model = AutoModel.from_pretrained("amahdaouy/DomURLs_BERT")
    model.to(device)
    model.eval()
    print(f"✓ DomURLs_BERT loaded on {device}")
    return tokenizer, model

def get_bert_embeddings_batch(urls, tokenizer, model, batch_size=32):
    """
    Extract BERT embeddings for URLs
    Returns 768-dimensional embeddings
    """
    embeddings = []

    with torch.no_grad():
        for i in tqdm(range(0, len(urls), batch_size), desc="BERT Embedding Batches"):
            batch = urls[i:i + batch_size]

            # Tokenize
            inputs = tokenizer(
                batch,
                padding=True,
                truncation=True,
                max_length=512,
                return_tensors="pt"
            ).to(device)

            # Get embeddings
            outputs = model(**inputs)

            # Use [CLS] token embedding (first token) as URL representation
            batch_embeddings = outputs.last_hidden_state[:, 0, :].cpu().numpy()
```

```python
# ============================================================================
# 9: Extract Embeddings for Train/Val/Test Sets
# ============================================================================

print("=" * 80)
print("STAGE 2: EMBEDDING EXTRACTION PIPELINE")
print("=" * 80)

# FORCE REGENERATE FLAG
# Set to True to regenerate embeddings even if they exist
# Set to False to use cached embeddings if available
FORCE_REGENERATE = True

print(f"\nForce regenerate: {FORCE_REGENERATE}")
if FORCE_REGENERATE:
    print("⚠ Will regenerate all embeddings (ignoring cached files)")
else:
    print("ℹ Will use cached embeddings if available")

# Process training set
print("\n[1/3] Processing TRAINING set embeddings...")
train_urls_combined = pd.DataFrame({
    'URL': train_df['URL'],
    'label': train_df['label']
}).reset_index(drop=True)

# Add enriched URLs to the dataframe
if 'enriched_url' in train_df.columns:
    train_urls_combined['enriched_url'] = train_df['enriched_url']

    # PREVIEW: Show what's being sent to GPT API
    print("\n" + "=" * 80)
    print("PREVIEW: Sample URL data being sent to GPT-3.5-Large API")
    print("=" * 80)
    sample_idx = 0
    print(f"\n[Sample URL #{sample_idx+1}]")
    print(f"Original URL: {train_urls_combined['URL'].iloc[sample_idx]}")
    print(f"Label: {train_urls_combined['label'].iloc[sample_idx]} ({'Benign' if train_urls_combined['label'].iloc[sample_idx] == 0 else 'Phishing'})")
    print(f"\nEnriched text sent to GPT API:")
    print("-" * 80)
    print(train_urls_combined['enriched_url'].iloc[sample_idx])
    print("-" * 80)
```

```python
# ============================================================================
#  STAGE 3.1: CREATE PYTORCH DATASETS AND DATALOADERS
# ============================================================================

class PhishingFusionDataset(Dataset):
    """
    Custom Dataset for dual-embedding fusion model
    """
    def __init__(self, gpt_embeddings, bert_embeddings, labels):
        self.gpt_embeddings = torch.FloatTensor(gpt_embeddings)
        self.bert_embeddings = torch.FloatTensor(bert_embeddings)
        self.labels = torch.LongTensor(labels)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return {
            'gpt_emb': self.gpt_embeddings[idx],
            'bert_emb': self.bert_embeddings[idx],
            'label': self.labels[idx]
        }

# Create datasets
print("Creating PyTorch datasets...")
train_dataset = PhishingFusionDataset(train_gpt_embeddings, train_bert_embeddings, train_labels)
val_dataset = PhishingFusionDataset(val_gpt_embeddings, val_bert_embeddings, val_labels)
test_dataset = PhishingFusionDataset(test_gpt_embeddings, test_bert_embeddings, test_labels)

# Create dataloaders
BATCH_SIZE = 256

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=2, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=2, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=2, pin_memory=True)

print(f"\nDataset sizes:")
print(f"  Train: {len(train_dataset):,} samples, {len(train_loader)} batches")
print(f"  Val: {len(val_dataset):,} samples, {len(val_loader)} batches")
print(f"  Test: {len(test_dataset):,} samples, {len(test_loader)} batches")
print(f"  Batch size: {BATCH_SIZE}")
```


Fusion Model Performance on Test Set (n=35,370)
Score (%)
Metric
Accuracy
Precision
Recall
F1-Score
AUC-ROC
97.27%
95.89%
99.49%
97.66%
99.30%
Target Threshold (97%)


Confusion Matrix - Test Set Performance
Actual Label
Predicted Label
Benign
Phishing
14,280
(40.4%)
862
(2.4%)
104
(0.3%)
20,124
(56.9%)


Learned Fusion Weights by Component
BERT→GPT Attention
GPT→BERT Attention
BERT Direct
GPT Direct
10.02%
10.00%
62.71%
17.26%
Weight Contribution (%)

5. References

Google Colab (Reference): https://colab.research.google.com/

Hannousse, A., & Yahiouche, S. (2021). Towards benchmark datasets for machine learning based website phishing detection: An experimental study. *Engineering Applications of Artificial Intelligence*, 104, 104347. https://doi.org/10.1016/j.engappai.2021.104347