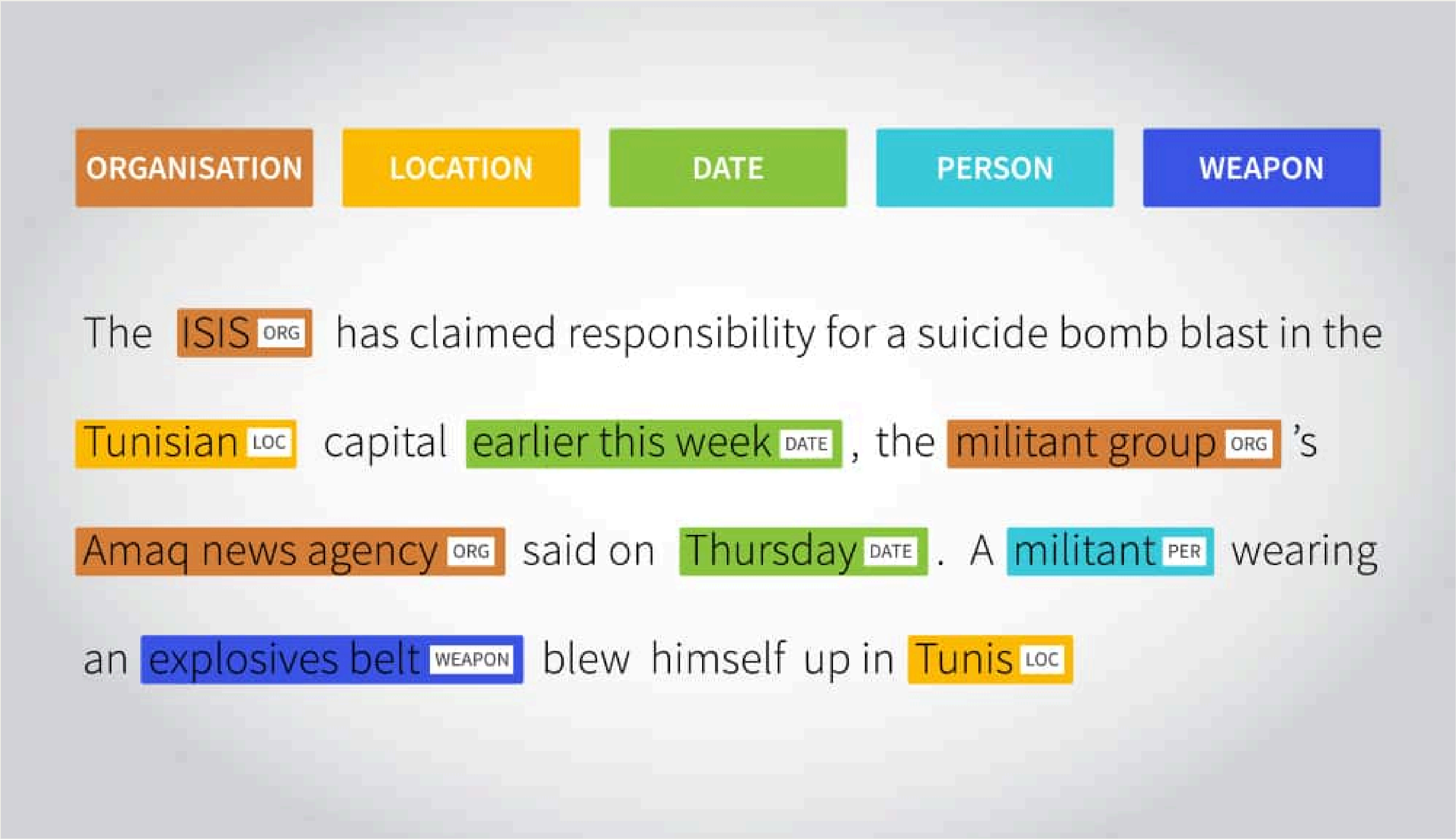
Using BERT for Name Entity Recognition

Alexios Doganis

CONTENTS

INTRODUCTION



What is Named Entity Recognition (NER)?

NER identifies and classifies named entities (people, places, organizations, etc.) in text, enabling computers to understand key information.

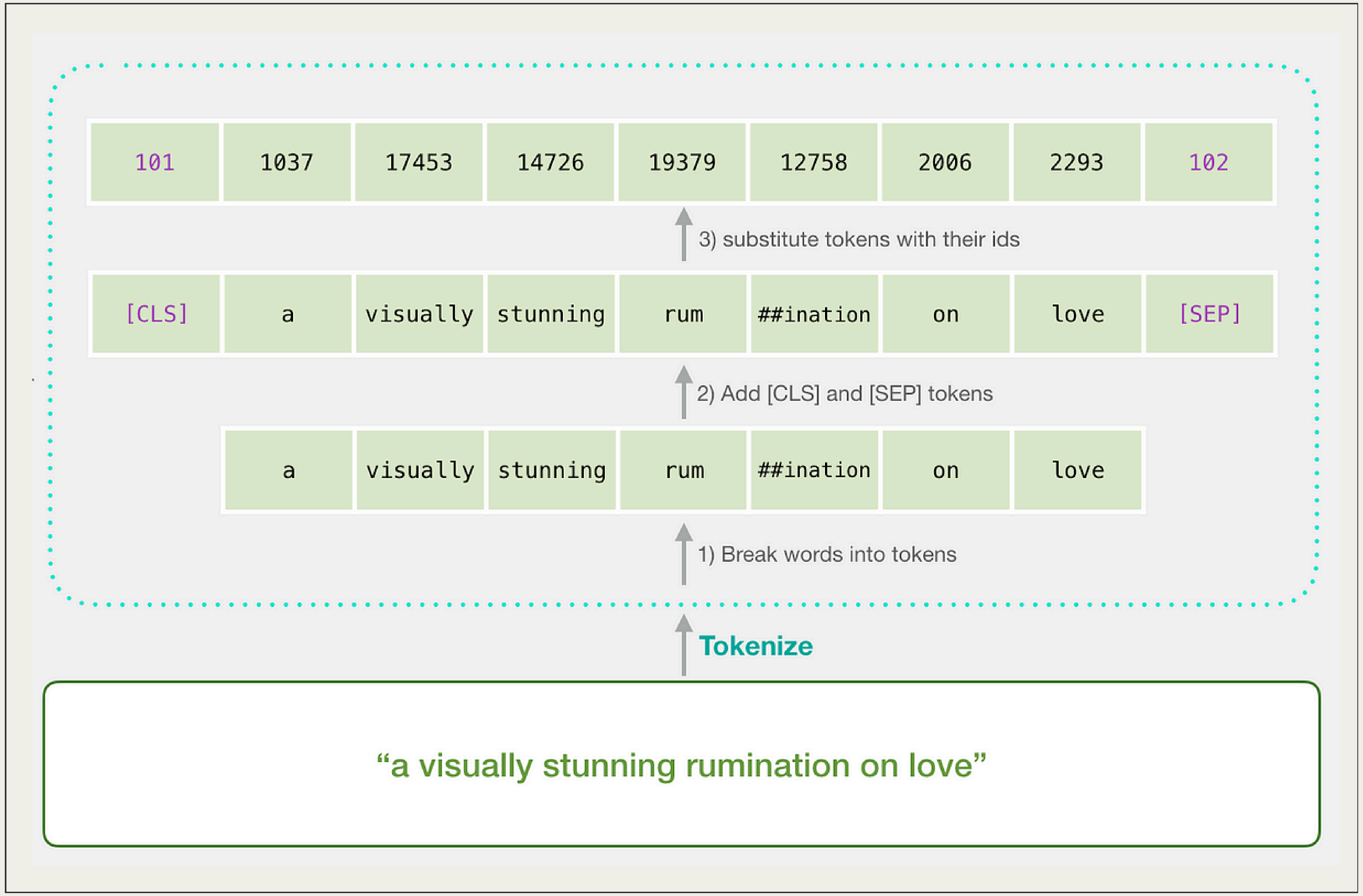
DATA LOADING AND PREPROCESSING

	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	NaN	of	IN	O
2	NaN	demonstrators	NNS	O
3	NaN	have	VBP	O
4	NaN	marched	VBN	O
5	NaN	through	IN	O
6	NaN	London	NNP	B-geo

Getting the dataset ready for training your model

- Read dataset using Pandas
- Fill missing sentence identifiers using forward fill
- Group sentences by their IDs
- Map NER labels and POS tags to integers

TOKENIZATION



Using BertTokenizer

- BERT uses the WordPiece Tokenizer
- WordPiece breaks word to subwords
- Nedd for alingment of NER labels and POS tags with subword tokens
- Pad sequences to fixed length

DATASET CREATION

```
class NERDatasetWithPOS(Dataset):
    def __init__(self, input_ids, attention_masks, labels, pos_tags):
        self.input_ids = input_ids
        self.attention_masks = attention_masks
        self.labels = labels
        self.pos_tags = pos_tags

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

    def __getitem__(self, idx):
        return {
            'input_ids': torch.tensor(self.input_ids[idx]),
            'attention_mask': torch.tensor(self.attention_masks[idx]),
            'labels': torch.tensor(self.labels[idx]),
            'pos_tags': torch.tensor(self.pos_tags[idx])
        }
```

Defining a custom PyTorch Dataset class

- After completing all the necessary steps we store input IDs, attention masks, labels, and POS tags
- Prepare data for model training using a DataLoader

MODEL ARCHITECTURE

Input Sentence → Tokenization → BERT Embeddings
↓ ↓
POS Tags → POS Embeddings ↓
↓
Concatenation [POS Embeddings + BERT Embeddings]
↓
Dropout Layer
↓
Classification Layer
↓
Named Entity Predictions

BERT Embeddings

- Input sentence goes through BERT Base Model and creates contextual word embeddings

POS Embeddings

- POS tags are converted into dense vectors

MODEL ARCHITECTURE

```
BERTNERWithPOS(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(28996, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
```

⋮

```
  )
  (pos_embedding): Embedding(26, 768)
  (dropout): Dropout(p=0.2, inplace=False)
  (classifier): Linear(in_features=1536, out_features=11, bias=True)
)
```

Concatenation

- The two embeddings are concatenated along the last dimension

Dropout

- Dropout layer is applied to the combined embeddings to prevent overfitting.

Classification Layer

- Linear layer reduces the dimension from 1536 → num_of_labels (11 classes)
- Final logits predict the NER tag for each token.

MODEL ARCHITECTURE

Input Sentence → Tokenization → Dataset →
DataLoader → BERTNERWithPOS Model

↓

Forward Pass

↓

Loss Calculation

↓

Backpropagation (Gradient Descent)

↓

Model Weights Update

↓

Repeat for each Epoch

Loss Function

- Uses CrossEntropyLoss
- Special tokens like subwords, [PAD] or [CLS] are ignored from calculation by setting their labels to -100

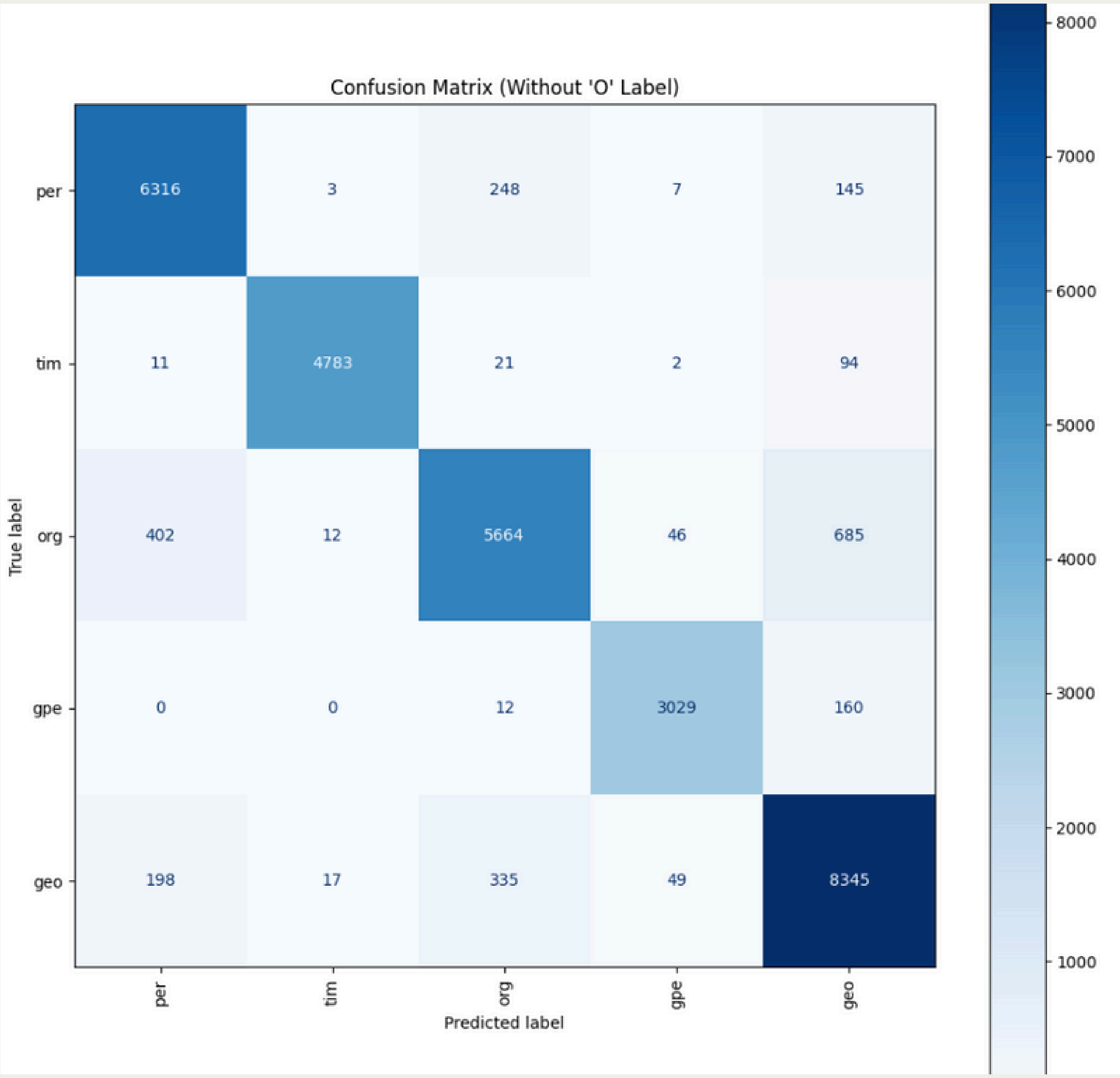
Optimizer and Learning Rate Scheduler

- Optimizer: AdamW
- Linear Warmup Scheduler gradually increases the learning rate in the first few steps and then decays it.

TRAINING AND EVALUATION

	precision	recall	f1-score	support
geo	0.85	0.91	0.88	7664
gpe	0.95	0.95	0.95	3175
org	0.74	0.70	0.72	3913
per	0.78	0.79	0.78	3389
tim	0.88	0.87	0.87	4049
micro avg	0.84	0.85	0.85	22190
macro avg	0.84	0.84	0.84	22190
weighted avg	0.84	0.85	0.84	22190

F1-score: 0.8445392317033534
Precision: 0.8396905841557749
Recall: 0.851194231635872



CONCLUSION

Limitations

- Computational Cost
- Limited variety of Label Tags

Future Improvements

- Expand the dataset to include more labels
- The POS impact wasn’t significant but can be improved by using pre-trained-embeddings
- Try more combinations during hyperparameter tuning

Thank you!