Legacy code transformation using AI

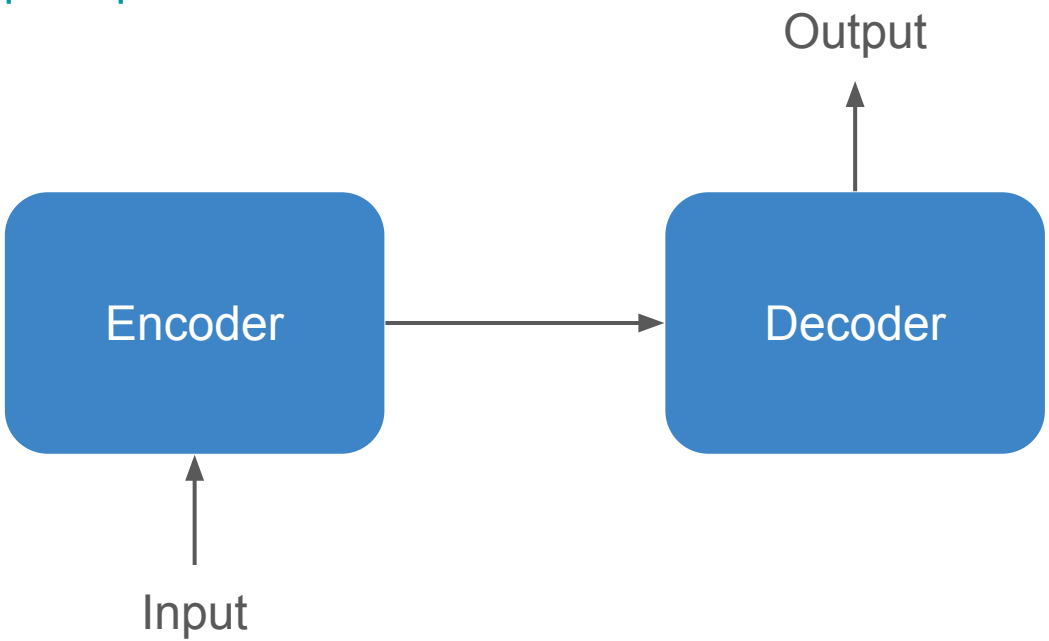
Translating PL/I to Kotlin using a Seq2Seq Transformer

Content

- Introduction
- Setting up the environment
- Data preparation and tokenization
- Translation and transpilation
- Model training and results
- Running the transpiler
- Demo
- Questions?

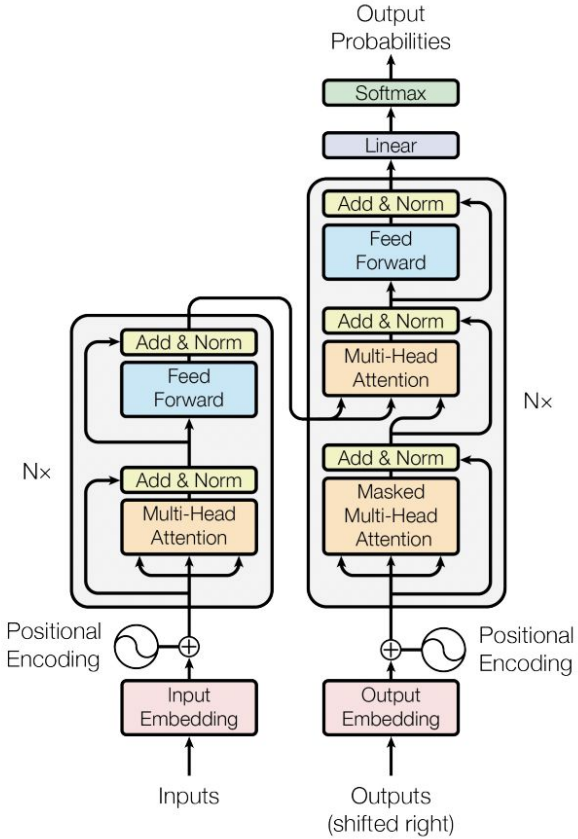
Introduction

- What are Seq2Seq models?



Introduction

- What are Seq2Seq models?
- Transformer-based Seq2Seq model

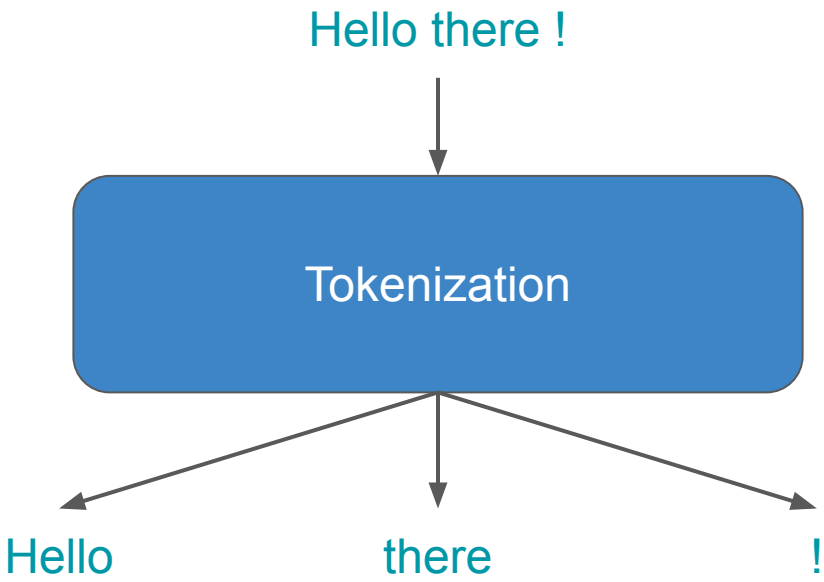


Setting Up the Environment

- 3 simple libraries
  - PyTorch for neural network building
  - Torchtext for handling textual data
  - ANTLR4 for parsing the PL/I code

Data Preparation and Tokenization

- Tokenizing the input data



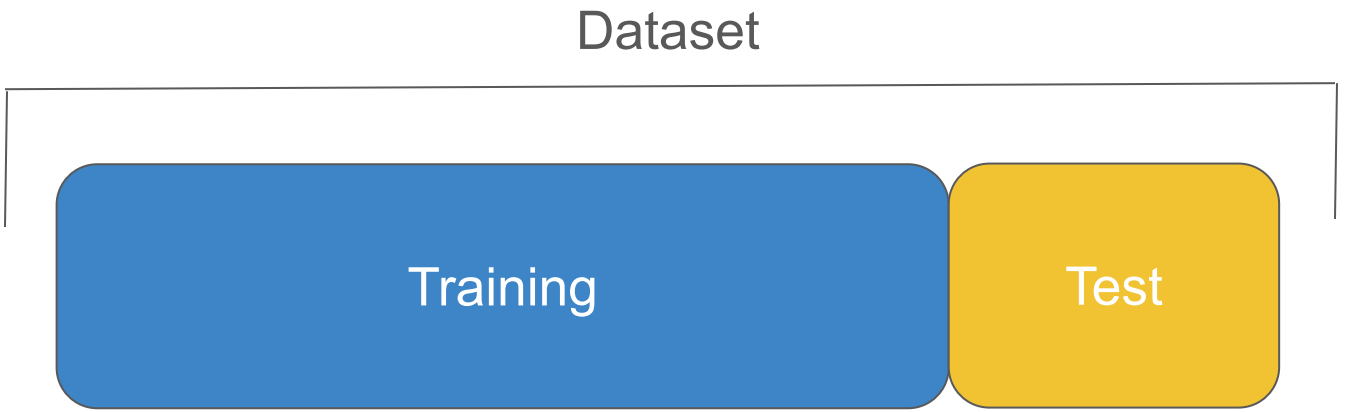
Data Preparation and Tokenization

- Tokenizing the input data

```
Sample 1:
  pli tokens: ['procedure', 'main', '{{type0}}', '{{type1}}']
  ktl tokens: ['fun', 'main', '(args: {{type0}}<{{type1}}>)']
Sample 2:
  pli tokens: ['procedure', '{{name}}']
  ktl tokens: ['fun', '{{name}}()']
```

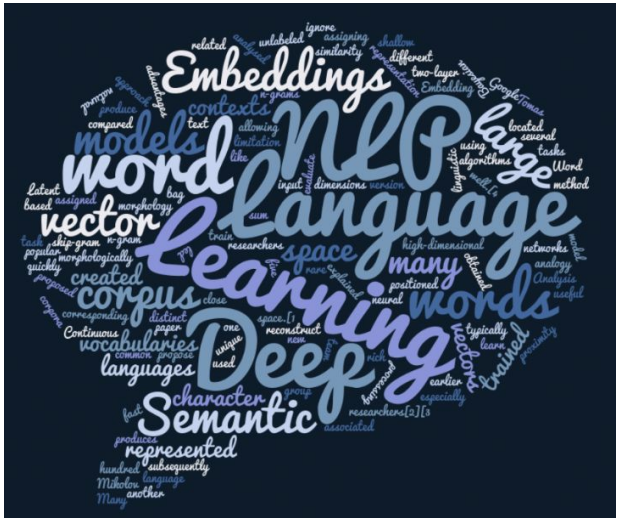
Data Preparation and Tokenization

- Tokenizing the input data
- Defining dataset



Data Preparation and Tokenization

- Tokenizing the input data
- Defining datasets
- Building a vocabulary



Data Preparation and Tokenization

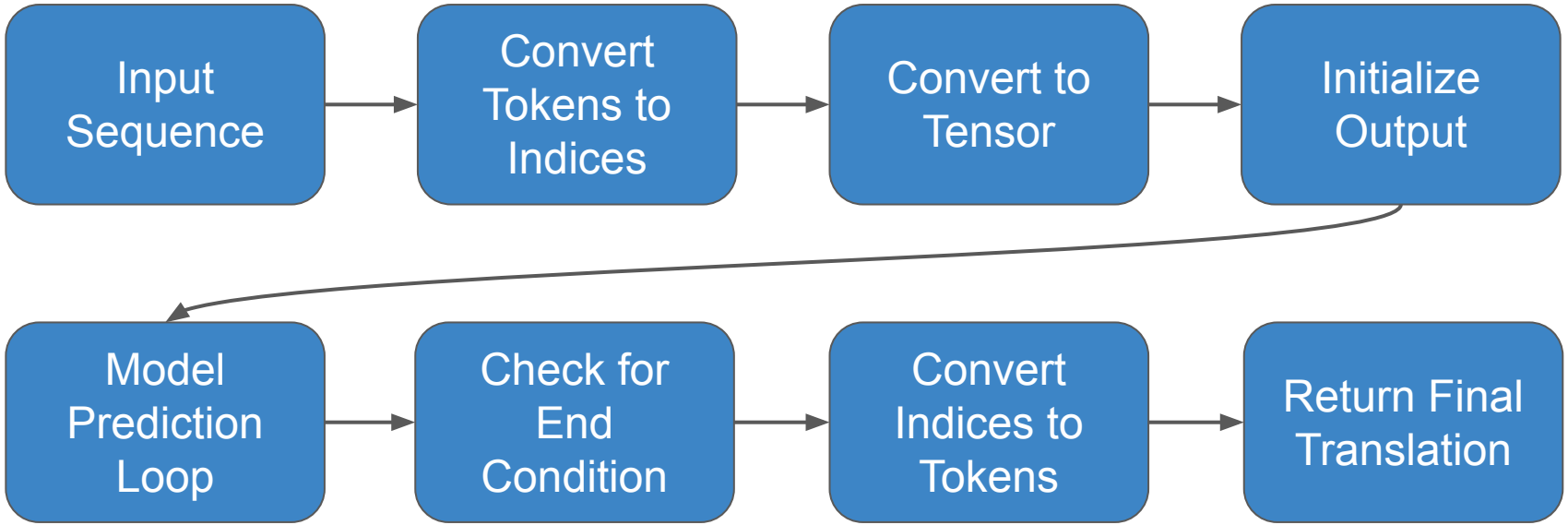
- Tokenizing the input data
- Defining datasets
- Building a vocabulary

```
PLI Vocabulary:
Token: <unk>, Index: 0
Token: <pad>, Index: 1
Token: <sos>, Index: 2
Token: <eos>, Index: 3
Token: {{name}}, Index: 4
Token: procedure, Index: 5
```

```
KTL Vocabulary:
Token: <unk>, Index: 0
Token: <pad>, Index: 1
Token: <sos>, Index: 2
Token: <eos>, Index: 3
Token: fun, Index: 4
Token: {{name}}(), Index: 5
```

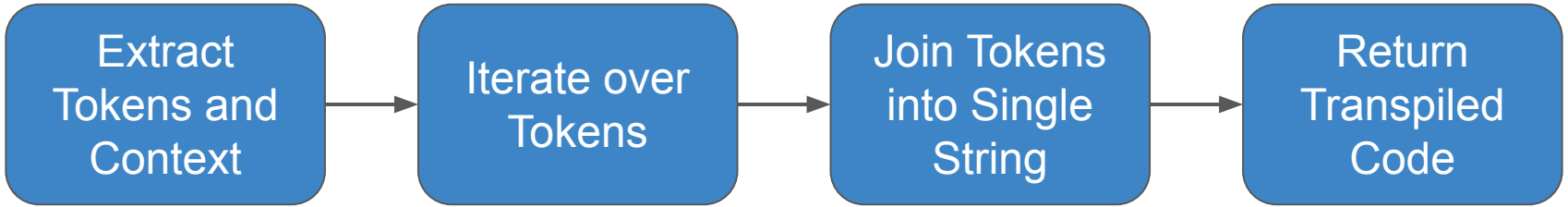
Translation & Transpilation

- Translation sequence



Translation & Transpilation

- Translation sequence
- Transpilation sequence



Model Training and Results

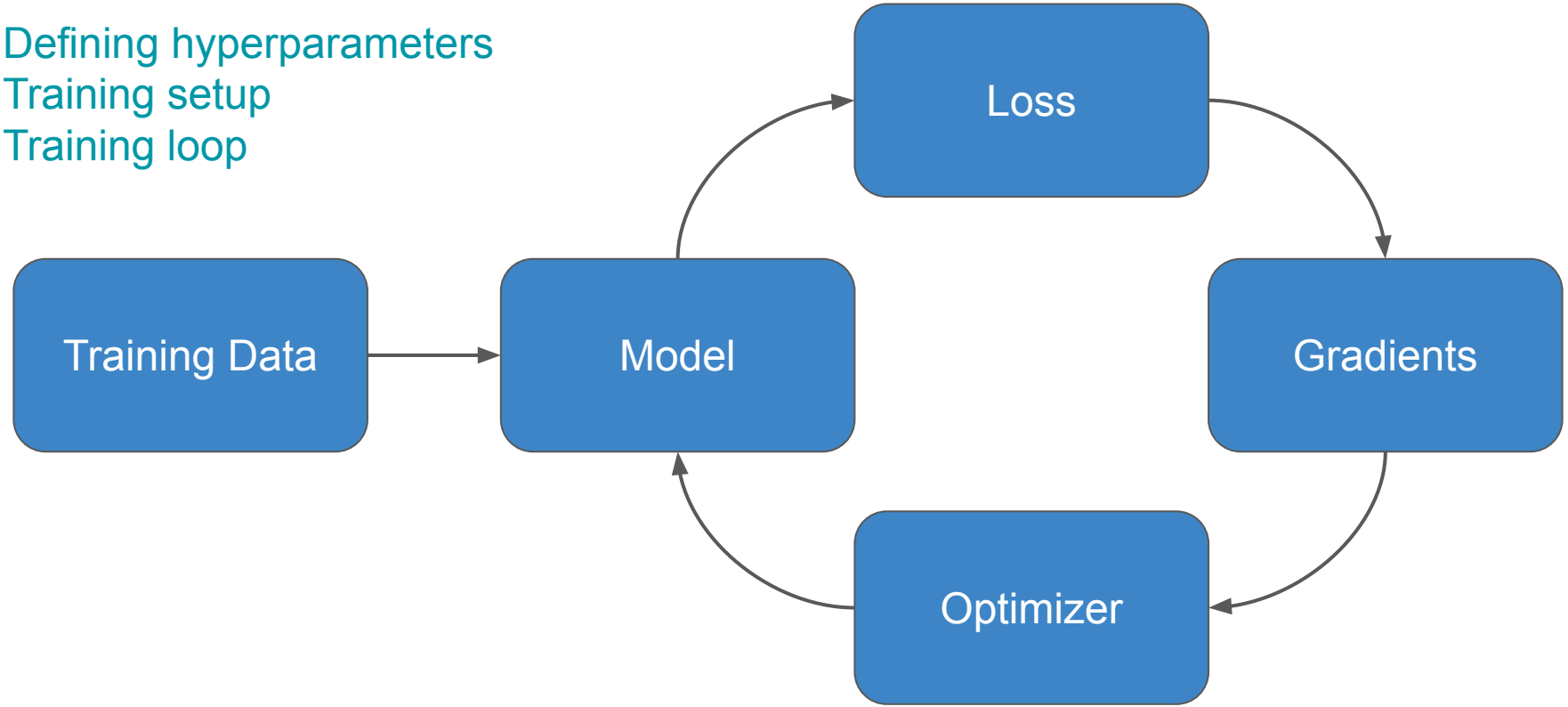
- Defining hyperparameters
  - Epochs
  - Learning rate
  - Batch size

Model Training and Results

- Defining hyperparameters
- Training setup
  - Loss Function
  - Gradients
  - Optimizer

Model Training and Results

- Defining hyperparameters
- Training setup
- Training loop

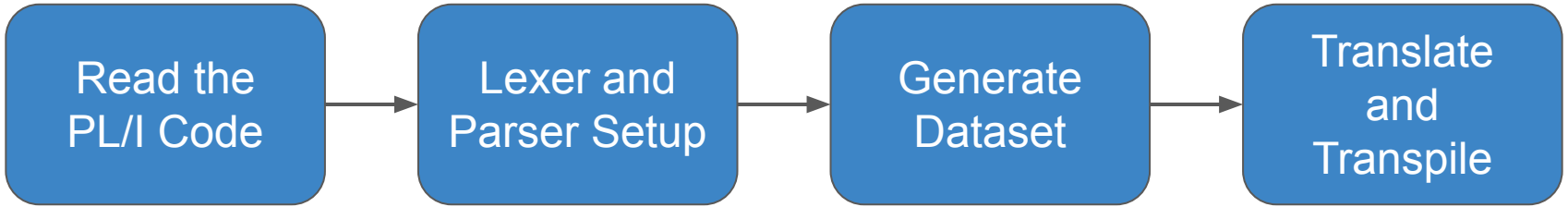


Model Training and Results

- Defining hyperparameters and initializing the model
- Training setup
- Training loop
- Training results

Running the Transpiler

- Running model



Running the Transpiler

- Running model
- Results

Input

```
PL1:
 Factorial: proc options (main);
    dcl (n,result) fixed bin(31);
    n  = 5;
    result = Compute_factorial(n);
 end Factorial;
  /*********************************************/
  /* Subroutine                                */
  /*********************************************/
  Compute_factorial: proc (n)  returns (fixed bin(31));
     dcl n fixed bin(15);
      if n <= 1 then
        return(1);
     return( n*Compute_factorial(n-1) );
  end Compute_factorial;
```

Output

```
KTL:
fun main (args: Array<String>)
{
    var     n : Int
    var     result : Int
    n = 5
    result = compute_factorial(n)
}
fun compute_factorial(n : Int) : Int
{
    if(n<=1)
    {
        return 1
    }
    return n*compute_factorial(n-1)
}
```

Demo

Questions ?

Thank you !

Contacts

https://www.linkedin.com/in/gian-cunningham/

https://giancunningham.com/