Encoder is All You Need

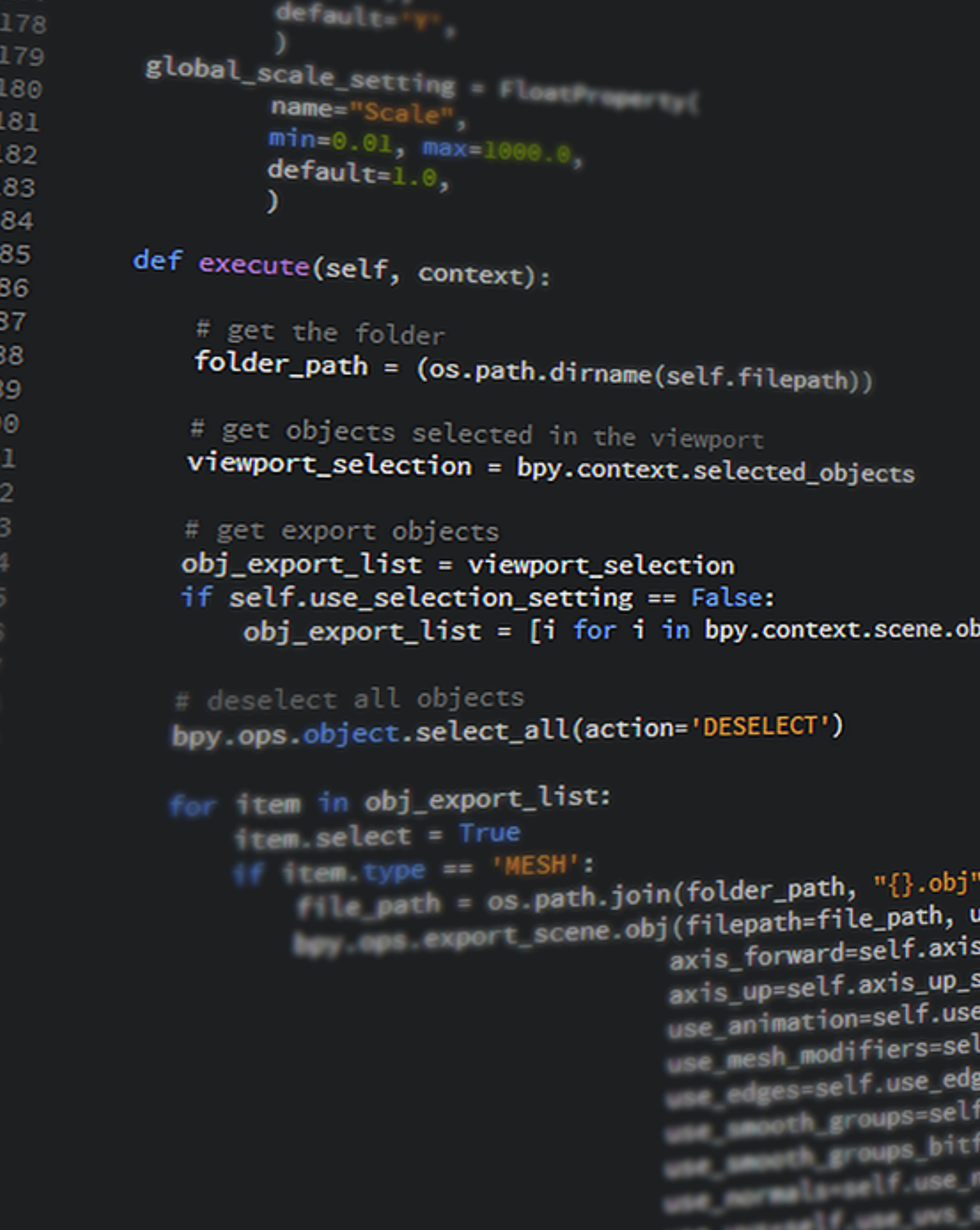
Clone Detection for Python Programming Language

Sangha Park (sangHa0411)
Young-Jin Ahn (snoop2head)

Executive Summary

Encoder is All You Need

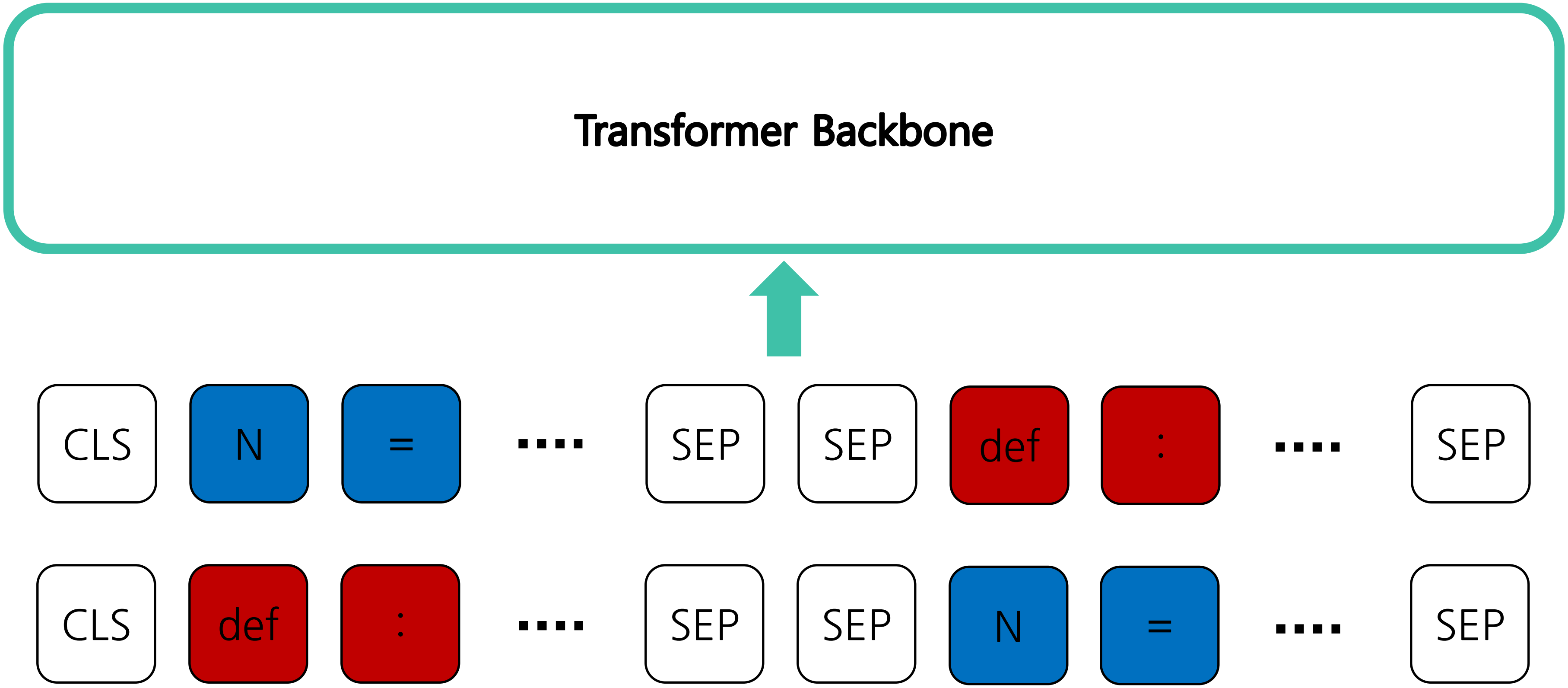
Tokenize & Preprocess	**Reducing Tokens to Meet the Constraint of 512 Tokens** • Removing “import” blocks in the beginning. • Filtering out unused functions in the main function. • Deleting comments(#), comment blocks('''), multiple spaces, multiple line breaks.
Modeling	**Utilizing Encoders Pretrained with Different Datasets and Pretraining Objectives** • **CodeBERT & GraphCodeBERT**: Finetuning RoBERTa-base model pretrained on CodeSearchNet with MLM and RTD objectives. • **Encoder-CodeT5**: Finetuning Encoder Layers extracted from the pretrained CodeT5. • **Encoder-PLBART**: Finetuning Encoder Layers extracted from the pretrained PLBART.
Train & Inference	**Exploiting Transformer Backbone for Better Classification Performance** • Kfold cross validation training with total 10M Python code pairs. • R-Drop Regularization: optimizing based on KL-Divergence loss from shuffled input sequence in addition to cross entropy loss. • Horizontal feature extraction from the last hidden states: [CLS], [SEP], <Python> and sentence pooling. • Vertical feature extraction concatenating single token’s last 4 layers’ hidden states. • Hardvoting ensemble RobertaRBERT, VHBartEncoder and VHT5Encoder.


global_scale_setting
name="Scale",
min=0.01, max=1000.0,
default=1.0,
def execute(self, context):
# get the folder
folder_path = (os.path.dirname(self.filepath))
# get objects selected in the viewport
viewport_selection = bpy.context.selected_objects
# get export objects
obj_export_list = viewport_selection
if self.use_selection_setting == False:
obj_export_list = [i for i in bpy.context.scene.obj
# deselect all objects
bpy.ops.object.select_all(action='DESELECT')
for item in obj_export_list:
item.select = True
if item.type == 'MESH':
file_path = os.path.join(folder_path, "{}.obj"
bpy.ops.export_scene.obj(filepath=file_path, u
axis_forward=self.axis
axis_up=self.axis_up_s
use_animation=self.use
use_mesh_modifiers=sel
use_edges=self.use_edg

CONTENTS

Defining Problems

Defining Problems



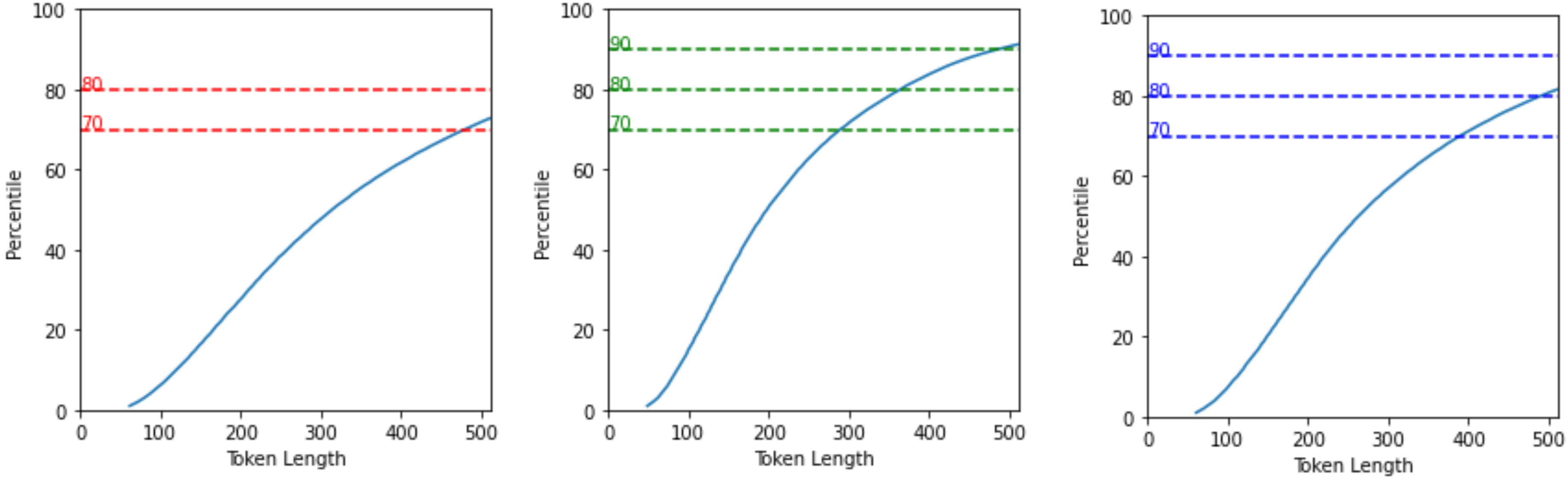
The code pair's order within the input sequence yielded two different outputs.
It was believed that the order within the pair affected the performance.

Defining Problems

PLM's Classification Performance (%)	Clone Detection (F1 Score)	Vulnerability Detection (Accuracy)
CodeBERT	96.5	62.08
GraphCodeBERT	97.1	-
PLBART(Encoder-Decoder)	**97.2**	63.18
CodeT5(Encoder-Decoder)	**97.2**	**65.78**

For programming language classification task,
CodeXGLUE Benchmark was available for comparing available PLMs.

Defining Problems



Test Set Coverage (%)	Within ≤ 512 Tokens
CodeBERT & GraphCodeBERT	72%
PLBART	**90%**
CodeT5	81%

PLBART and CodeT5 seemed to be suitable for the given task considering the percentage of test dataset without truncation.

Defining Problems

PLMs’ model size	#Params	Model Size (MB)	#Layers	Train Batch Size (FP32, 512 Token)
CodeBERT & GraphCodeBERT	0.49B	476	12	24
PLBART(Encoder-Decoder)	1.62B	1548	24	-
CodeT5(Encoder-Decoder)	0.89B	850	12	12

However, CodeT5-base and PLBART-large was too massive to process with given computation resource of 40GB memory (A100 GPU)

Tokenize & Preprocess

Defining Problems

```python
from collections import namedtuple
import queue
import sys
import math
import copy
import itertools
import bisect, heapq
import fractions

def insertion_sort(array, size, gap=1):
    """ AOJ??¬??¶?????¬???
    http://judge.u-aizu.ac.jp/onlinejudge/commentary.jsp?id=ALDS1_1_A

    :param array: ?????????????±??????????
    :return: ?????????????????????
    """
    global Count

    for i in range(gap, len(array)):  # i????
        v = array[i]  # ?????¨?????\??????????
        j = i - gap
        while j >= 0 and array[j] > v:
            array[j + gap] = array[j]
            j = j - gap
            Count += 1
        array[j + gap] = v
    return array
```

Removed “import” blocks at the top, only remaining library used within the code

Defining Problems

```
from collections import namedtuple
import queue
import sys
import math
import copy
import itertools
import bisect, heapq
import fractions

def insertion_sort(array, size, gap=1):
    """ AOJ??¬??¶?????¬???
    http://judge.u-aizu.ac.jp/onlinejudge/commentary.jsp?id=ALDS1_1_A

    :param array: ?????????????±??????????
    :return: ?????????????????????
    """
    global Count

    for i in range(gap, len(array)):  # i????
        v = array[i]  # ?????¨?????\??????????
        j = i - gap
        while j >= 0 and array[j] > v:
            array[j + gap] = array[j]
            j = j - gap
            Count += 1
        array[j + gap] = v
    return array
```

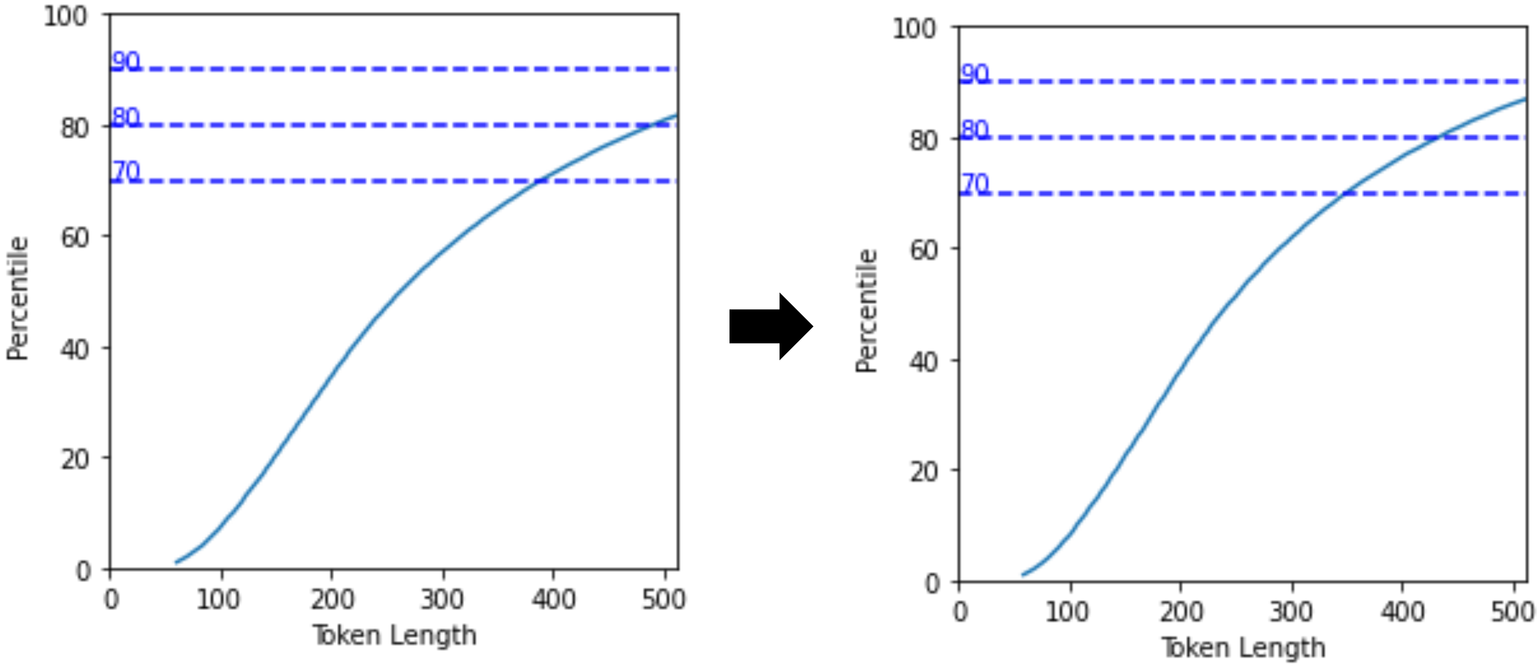
Deleted comments(#), comment blocks('''), multiple spaces, multiple line breaks

Defining Problems

```python
def LI(): return list(map(int, input().split()))
def LF(): return list(map(float, input().split()))
def LI_(): return list(map(lambda x: int(x)-1, input().split()))
def II(): return int(input())
def IF(): return float(input())
def S(): return input().rstrip()
def LS(): return S().split()
def IR(n): return [II() for _ in range(n)]
def LIR(n): return [LI() for _ in range(n)]
def FR(n): return [IF() for _ in range(n)]
def LFR(n): return [LI() for _ in range(n)]
def LIR_(n): return [LI_() for _ in range(n)]
def SR(n): return [S() for _ in range(n)]
def LSR(n): return [LS() for _ in range(n)]
mod = 1000000007
inf = 1e10

def solve():
    h, w, k = LI()
    s = SR(h)
    done = []
    ans = [[None] * w for i in range(h)]
    for y in range(h):
        f = 0
        for x in range(w):
            if f:
                if s[y][x] == "#":
                    k -= 1
                ans[y][x] = str(k)
            else:
                if s[y][x] == "#":
                    f = 1
                ans[y][x] = str(k)
        if f:
            done.append(y)
            k -= 1

if __name__ == '__main__':
    solve()
```

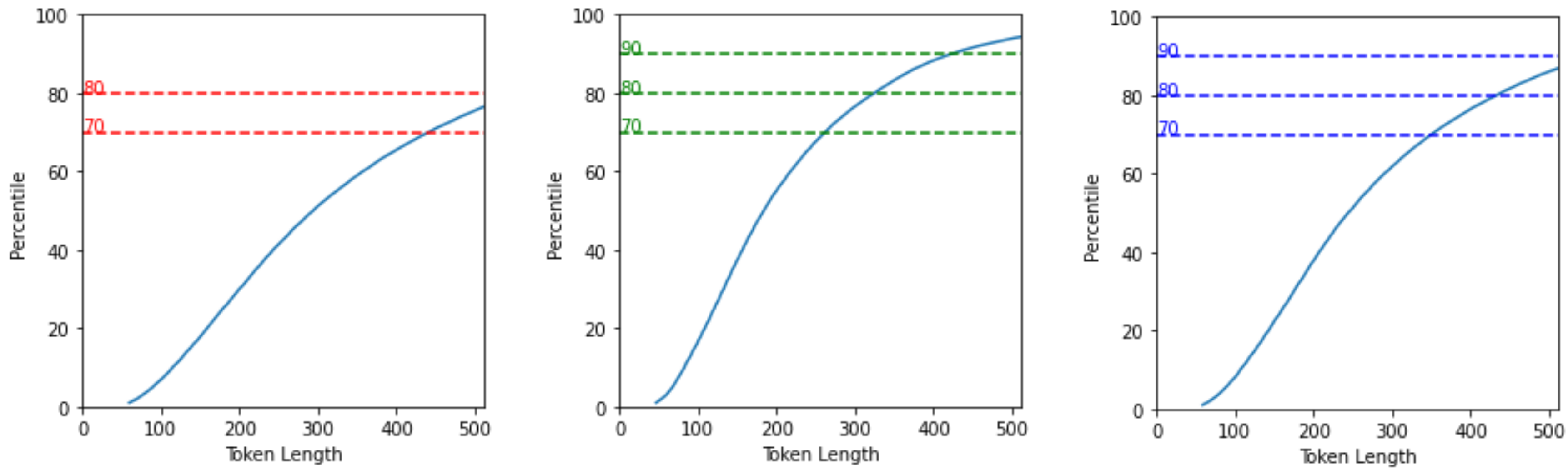
Filtered out unused functions in the __main__ function.

Tokenize & Preprocess



Applying preprocessing made +3%p of more test dataset
to fall within the range of 512 tokens.

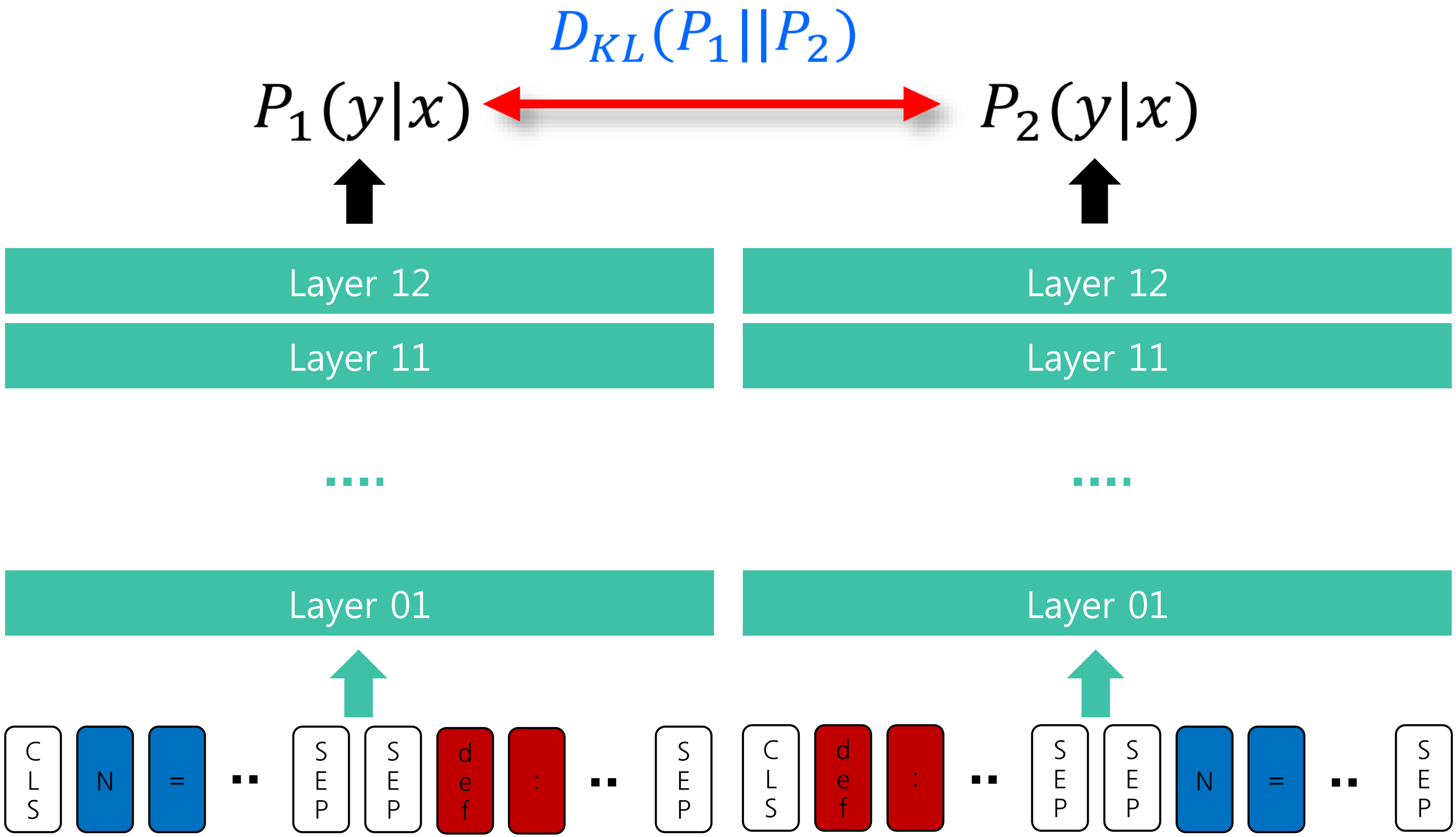
Defining Problems



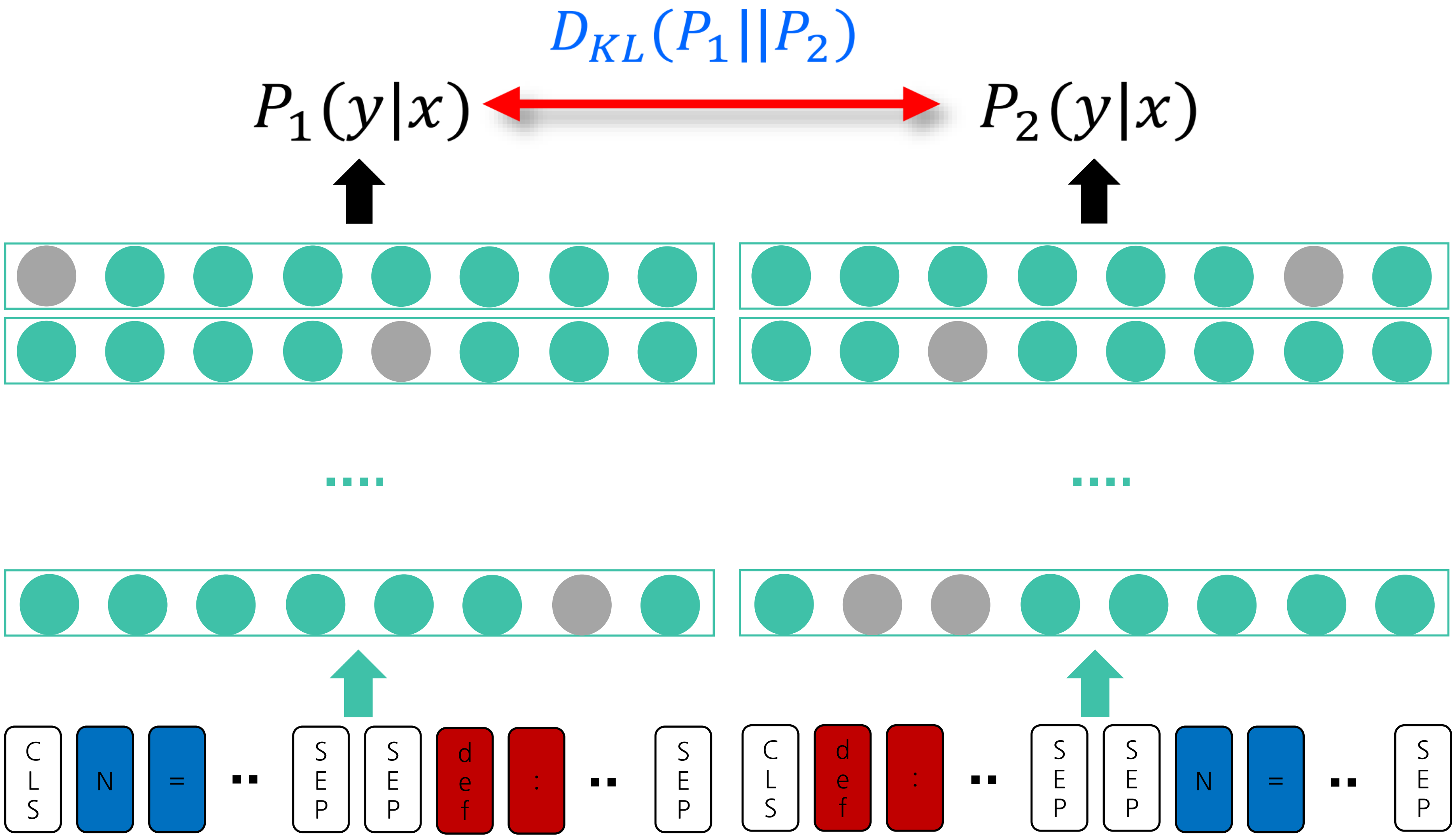
Test Set Coverage (%)	**Within ≤ 512 Tokens Before Preprocessing**	**Within ≤ 512 Tokens After Preprocessing**
CodeBERT & GraphCodeBERT	72%	**76%**
PLBART	90%	**94%**
CodeT5	81%	**84%**

Across the tokenizers, preprocessing made +3%p ~ +4%p
of more test dataset to fall within the range of 512 tokens.

Train & Inference



Optimization based on loss composed of KL-Divergence loss from shuffled input sequence and cross entropy loss



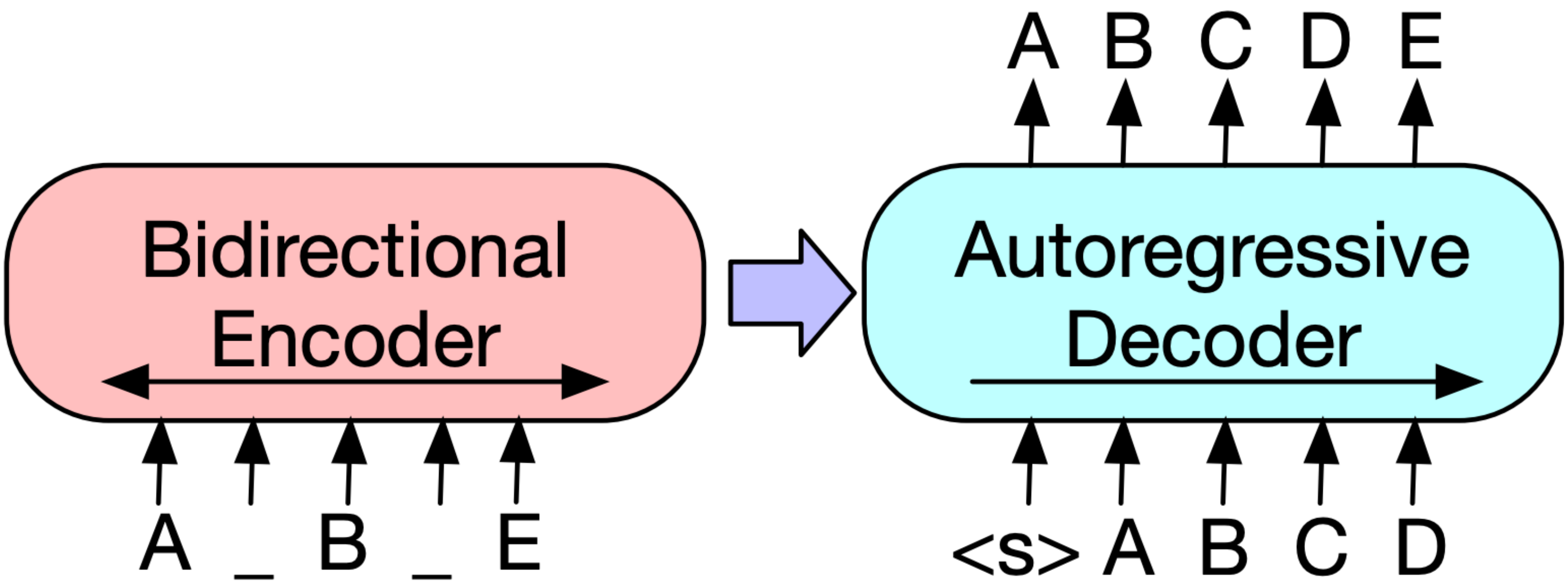
Regularization of the outputs from the randomly initialized dropouts within the Transformers backbone layers

Train & Inference

```
def compute_loss(self, model, inputs):
    num_labels = 2
    labels = inputs.pop("labels")

    # cls code1 sep sep code2 sep
    outputs1 = model(
      input_ids=inputs["input_ids"],
      attention_mask=inputs["attention_mask"]
    )
    logits1 = outputs1.logits

    # cls code2 sep sep code1 sep
    outputs2 = model(
      input_ids=inputs["input_ids2"],
      attention_mask=inputs["attention_mask2"]
    )
    logits2 = outputs2.logits

    # Crossentropy Loss
    loss_fct_1 = nn.CrossEntropyLoss()
    loss_nll = (
        loss_fct_1(logits1.view(-1, num_labels), labels.view(-1))
        + loss_fct_1(logits2.view(-1, num_labels), labels.view(-1))
    ) / 2

    # KL-Divergence Loss
    loss_fct_2 = nn.KLDivLoss(reduction="batchmean")
    loss_kl = self.get_kl_loss(loss_fct_2, logits1, logits2)

    # Sum Crossentropy Loss + KL-Divergence Loss
    return loss_nll + loss_kl
```

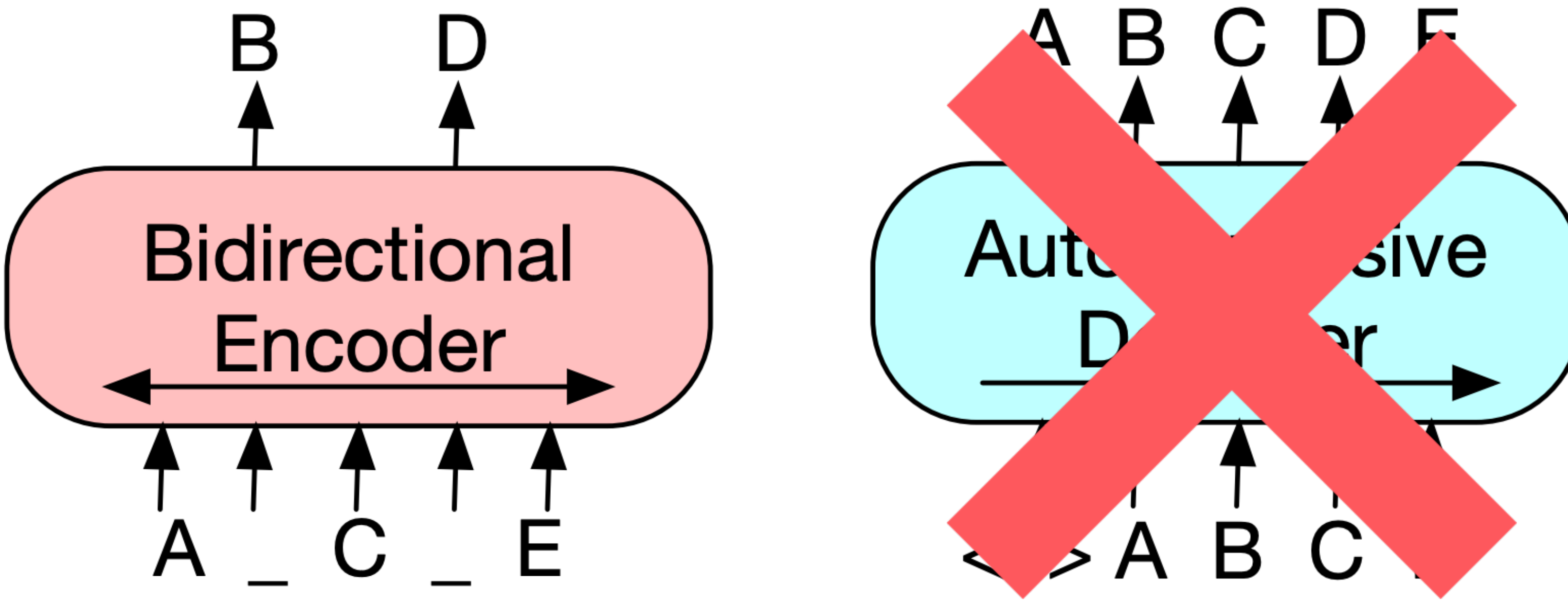
Optimizing total loss composed of KL-Divergence loss and cross entropy loss

Train & Inference



Both T5 and PLBART model was consisted with Encoder and Decoder Layers which made model size larger than CodeBERT.

Train & Inference



Removed decoder layers and
extracted features from the encoder layers for the classification task.

Train & Inference

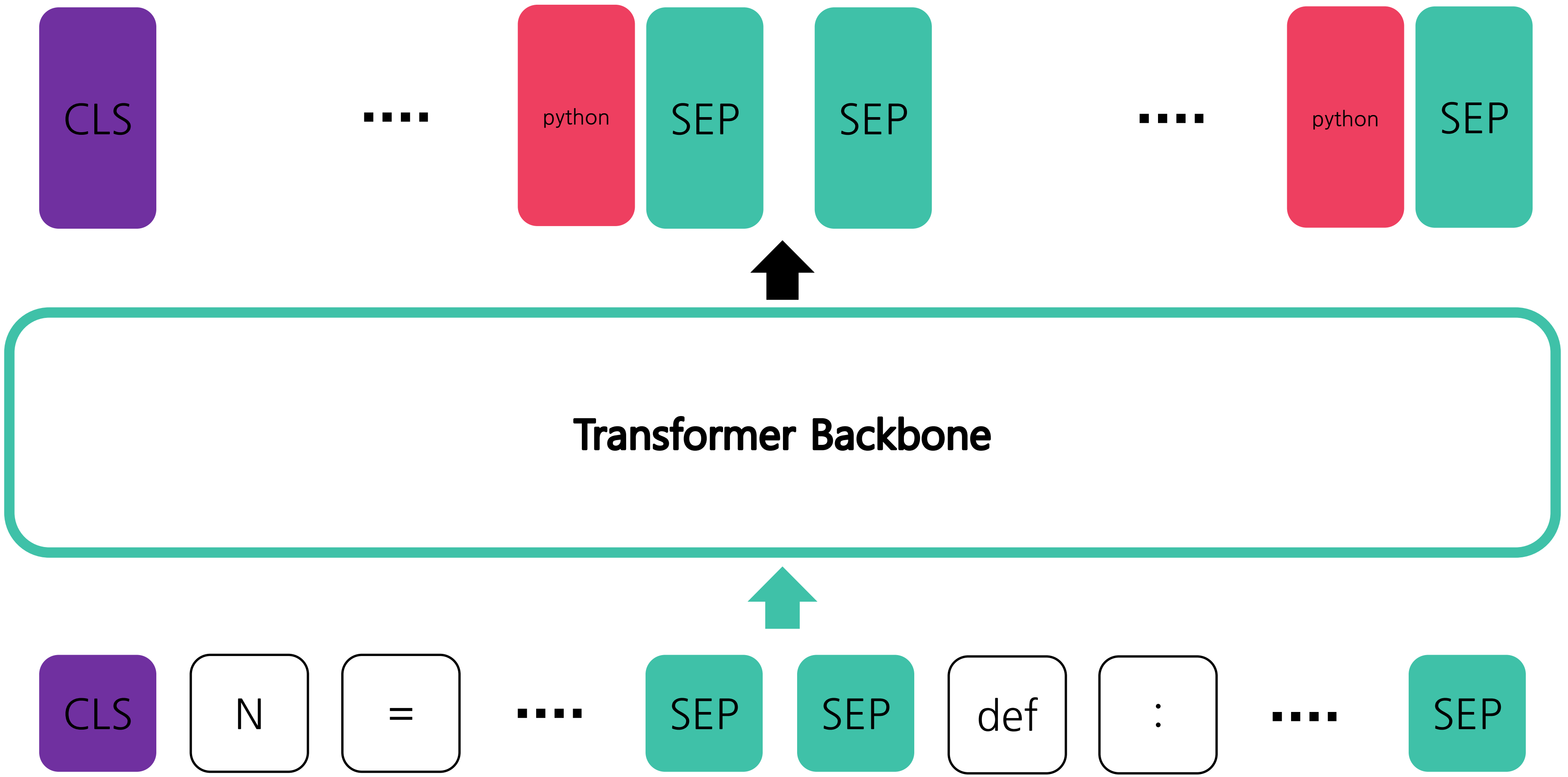
PLMs' model size	#Params	Model Size (MB)	#Layers	Train Batch Size (FP32, 512 Token)
CodeBERT & GraphCodeBERT	0.49B	476	12	24
PLBART(Encoder-Decoder)	**1.62B**	**1548**	**24**	**-**
CodeT5(Encoder-Decoder)	**0.89B**	**850**	**12**	**12**

⬇

PLMs' model size	#Params	Model Size (MB)	#Layers	Train Batch Size (FP32, 512 Token)
CodeBERT & GraphCodeBERT	0.49B	476	12	24
Encoder-PLBART	**0.81B**	**776**	**12**	**12**
Encoder-CodeT5	**0.44B**	**418**	**6**	**24**

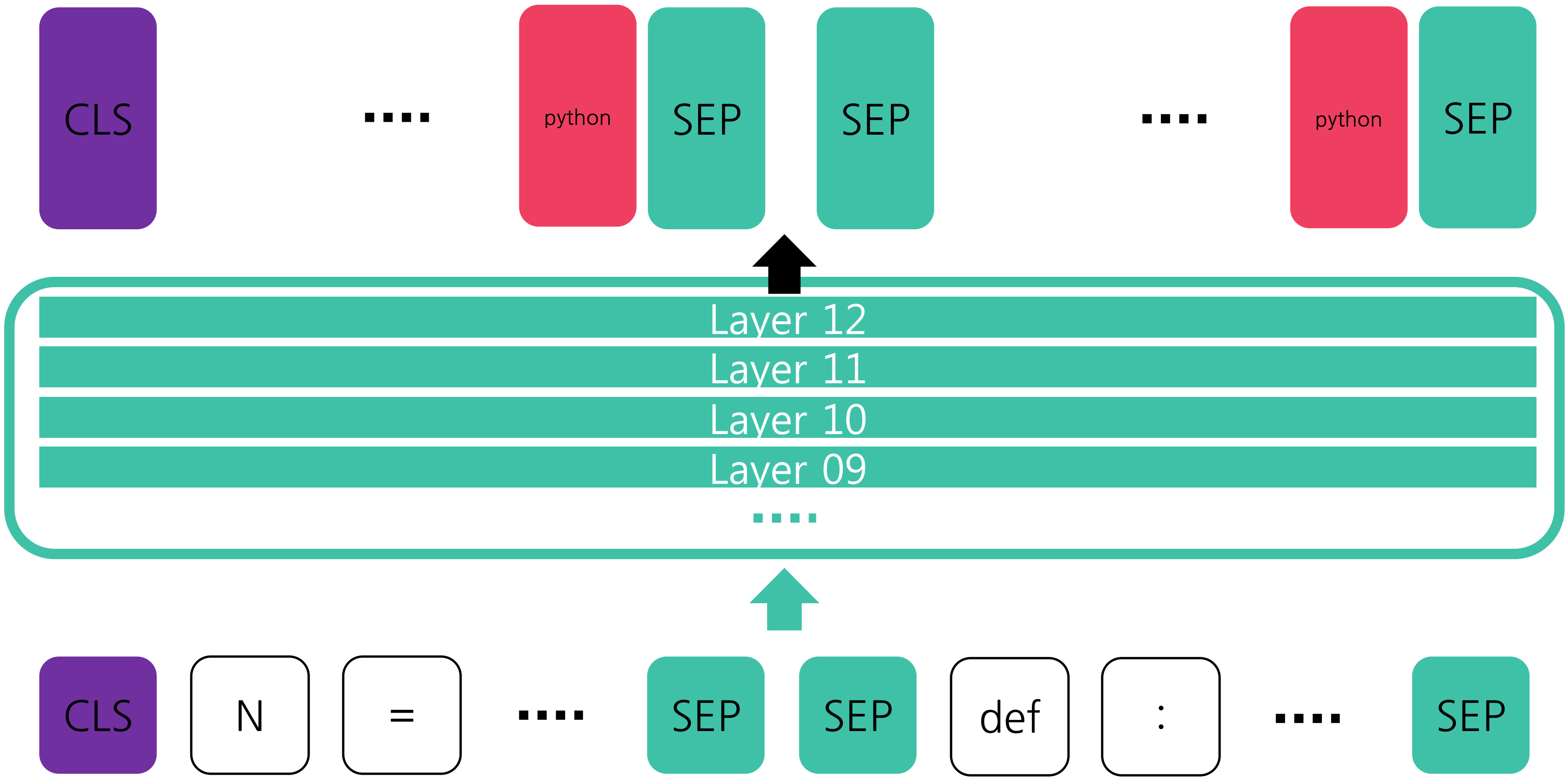
Deletion of the decoder layers made the model lighter,
whereas utilizing pretrained weights of the encoder layers.

Train & Inference



(Horizontal) Last Hidden State's tokens were sampled, pooled and concatenated.

Train & Inference



(Vertical) Individual tokens' 4 hidden states' outputs were concatenated.

Train & Inference

```python
class VHBartEncoderForSequenceClassification(PLBartModel):
    def __init__(self, config):
        super().__init__(config)
        self.config = config

        self.entity_fc_layer = FCLayer(
            self.config.hidden_size, self.config.hidden_size,
self.config.dropout_rate
        )
        self.proj_fc_layer = FCLayer(
            self.config.hidden_size * 4, self.config.hidden_size,
self.config.dropout_rate
        )
        self.label_classifier = FCLayer(
            self.config.hidden_size * 3,
            self.config.num_labels,
            self.config.dropout_rate,
            use_activation=True,
        )

    def forward(
        self, input_ids, attention_mask, code1_mask, code2_mask,
last_token_index, labels=None,
    ):
        outputs = self.encoder(
            input_ids=input_ids, attention_mask=attention_mask,
output_hidden_states=True,
        )

        idx_seq = torch.arange(input_ids.size(0)).to(input_ids.device)
        cls_concat = torch.cat(
            tuple(
                [outputs["hidden_states"][i][idx_seq, last_token_index] for
i in [-4, -3, -2, -1]]
            ),
            dim=-1,
        )
        cls_output = self.proj_fc_layer(cls_concat)
        sequence_output = outputs["last_hidden_state"]
```

```python
        code1_sentence_h = self.entity_average(
            sequence_output, code1_mask
        )
        code1_sentence_h = self.entity_fc_layer(
            code1_sentence_h
        )

        code2_sentence_h = self.entity_average(
            sequence_output, code2_mask
        )
        code2_sentence_h = self.entity_fc_layer(
            code2_sentence_h
        )

        concat = torch.cat(
            [
                code1_sentence_h,
                code2_sentence_h,
                cls_output,
            ],
            dim=-1,
        )

        logits = self.label_classifier(concat)
        prob = nn.functional.softmax(logits)

        if labels is not None:
            loss_fct = nn.CrossEntropyLoss()
            labels = labels.squeeze(-1)
            loss = loss_fct(logits, labels)
            return loss, prob
        else:
            return prob

    def entity_average(self, hidden_output, e_mask):
        e_mask_unsqueeze = e_mask.unsqueeze(1)
        length_tensor = (e_mask != 0).sum(dim=1).unsqueeze(1)
        sum_vector = torch.bmm(e_mask_unsqueeze.float(),
hidden_output).squeeze(1)
        avg_vector = sum_vector.float() / length_tensor.float()
        return avg_vector
```

Example of Vertical & Horizontal Token Sampling with Pretrained PLBART Encoder

Train & Inference

RobertaRBERT

VHT5Encoder

VHBartEncoder

Trained with total **10M Python code pairs with Kfold cross validation,**
hardvoting ensembled RobertaRBERT, VHBartEncoder and VHT5Encoder models.

End of the Document

Encoder is All You Need

Tokenize & Preprocess	**Reducing Tokens to Meet the Constraint of 512 Tokens** • Removing “import” blocks in the beginning. • Filtering out unused functions in the main function. • Deleting comments(#), comment blocks('''), multiple spaces, multiple line breaks.
Modeling	**Utilizing Encoders Pretrained with Different Datasets and Pretraining Objectives** • **CodeBERT & GraphCodeBERT**: Finetuning RoBERTa-base model pretrained on CodeSearchNet with MLM and RTD objectives. • **Encoder-CodeT5**: Finetuning Encoder Layers extracted from the pretrained CodeT5. • **Encoder-PLBART**: Finetuning Encoder Layers extracted from the pretrained PLBART.
Train & Inference	**Exploiting Transformer Backbone for Better Classification Performance** • Kfold cross validation training with total 10M Python code pairs. • R-Drop Regularization: optimizing based on KL-Divergence loss from shuffled input sequence in addition to cross entropy loss. • Horizontal feature extraction from the last hidden states: [CLS], [SEP], <Python> and sentence pooling. • Vertical feature extraction concatenating single token’s last 4 layers’ hidden states. • Hardvoting ensemble RobertaRBERT, VHBartEncoder and VHT5Encoder.

References

R-Drop Regularization

- R-Drop: Regularized Dropout for Neural Networks

Token Sampling Strategy (Vertical-Horizontal)

- Enriching Pre-trained Language Model with Entity Information for Relation Classification
- An Improved Baseline for Sentence-level Relation Extraction
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
- Matching the Blanks: Distributional Similarity for Relation Learning

Encoders for Sequence Classification

- EncT5: Fine-tuning T5 Encoder for Non-autoregressive Tasks

Backbone models and Benchmarks

- CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Gene ration
- CodeBERT: A Pre-Trained Model for Programming and Natural Languages
- GRAPHCODEBERT: PRE-TRAINING CODE REPRESENTATIONS WITH DATA FLOW
- Unified Pre-training for Program Understanding and Generation
- CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Underst anding and Generation

Appendix

Model	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	CoLA	Avg
BERT-base [9]	83.8	85.3	90.8	91.0	68.2	92.4	89.3	62.3	82.85
BERT-base + RD	85.5	87.3	92.0	91.4	71.1	93.0	89.6	62.6	**84.06**
RoBERTa-large [39]	90.2	90.9	94.7	92.2	86.6	96.4	92.4	68.0	88.93
XLNet-large [72]	90.8	90.8	94.9	92.3	85.9	97.0	92.5	69.0	89.15
ELECRTA-large [7]	90.9	90.8	95.0	92.4	88.0	96.9	92.6	69.1	89.46
RoBERTa-large + RD	90.9	91.4	95.2	92.5	88.4	96.9	92.5	70.0	**89.73**

Table 3: Fine-tuned model performances on GLUE language understanding benchmark.

Performance gain on Transformers backbone model (R-Drop Paper)

Appendix

Dataset # of training data Metrics	CoLA 8.5k Matthew	SST-2 67k Acc	MRPC 3.6k F1/Acc	STS-B 364k PCC/SCC	QQP 5.7k F1/Acc	MNLI 393k Mis/Matched	QNLI 105k Acc	RTE 2.5k Acc	GLUE Avg
T5-small*	41.0	91.8	89.7/86.6	85.6/85.0	70.0/88.0	82.4/82.3	90.3	69.9	78.5
T5.1.1-small	30.7	90.8	85.7/80.6	74.8/75.4	69.0/88.7	**83.6/82.3**	85.2	56.4	72.9
1decT5.1.1-small	27.6	87.9	86.2/80.8	72.3/70.4	**69.4/88.8**	83.2/82.4	84.1	56.4	71.6
EncT5.1.1-small	**32.5**	**91.2**	**87.0/81.6**	**74.9/73.6**	69.4/88.7	83.6/82.2	**89.0**	**59.1**	**74.0**
T5-base*	51.1	95.2	90.7/87.5	89.4/88.6	72.6/89.4	87.1/86.2	93.7	80.1	83.2
T51.1-base	49.7	**94.4**	91.0/87.7	**81.4/80.4**	72.6/89.8	**88.9/87.8**	93.2	**70.3**	**80.9**
1decT5.1.1-base	23.7	90.0	85.6/80.5	77.4/76.1	71.3/89.3	86.2/84.6	89.8	62.4	73.9
EncT5.1.1-base	**53.1**	94.0	**91.5/88.3**	80.5/79.3	**72.9/89.8**	88.0/86.7	**93.3**	67.8	80.8
T5-large*	61.2	96.3	92.4/89.9	89.9/89.2	73.9/89.9	89.9/89.6	94.8	87.2	86.5
T5.1.1-large	**54.2**	**96.7**	**91.4/88.3**	84.3/83.0	72.7/89.8	**90.4/90.3**	95.3	**83.9**	**84.4**
1decT5.1.1-large	49.2	94.3	90.0/86.4	**86.6/86.4**	72.0/89.5	89.8/89.1	94.3	73.2	82.0
EncT5.1.1-large	52.1	96.2	90.7/87.2	86.6/85.6	**72.9/89.9**	90.2/89.6	**95.6**	75.7	83.2
T5-3B*	67.1	97.4	92.5/90.0	90.6/89.8	74.4/89.8	91.2/91.4	96.3	91.1	88.3
T5.1.1-xl	62.3	96.5	**92.5/90.0**	**88.6/87.6**	72.7/89.8	90.8/90.4	95.2	86.7	86.5
1decT5.1.1-xl	13.4	95.5	92.4/89.6	87.1/86.9	72.7/90.0	90.8/90.2	95.5	83.8	79.8
EncT5.1.1-xl	**63.6**	**96.7**	91.8/88.9	87.7/86.9	**73.0/90.0**	**91.2/90.9**	**96.2**	**87.5**	**86.8**
T5-11B*	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2/91.9	96.9	92.8	89.8
T5.1.1-xxl	65.2	97.2	**92.9/90.4**	88.2/87.6	**73.2/90.0**	**91.7/91.5**	96.2	86.3	87.2
1decT5.1.1-xxl	18.6	85.3	86.6/81.7	88.5/88.3	70.5/89.1	91.2/90.8	89.1	84.6	77.6
EncT5.1.1-xxl	**67.7**	**97.4**	92.1/89.4	**89.2/88.8**	73.1/90.0	91.5/91.3	**96.5**	**89.8**	**88.0**
EncT5.1.1-base-rand	16.7	81.6	76.3/66.3	20.2/19.5	57.1/82.3	62.7/61.5	61.8	49.9	54.1

Encoder-Decoder model vs Encoder only model (EncT5 Paper)

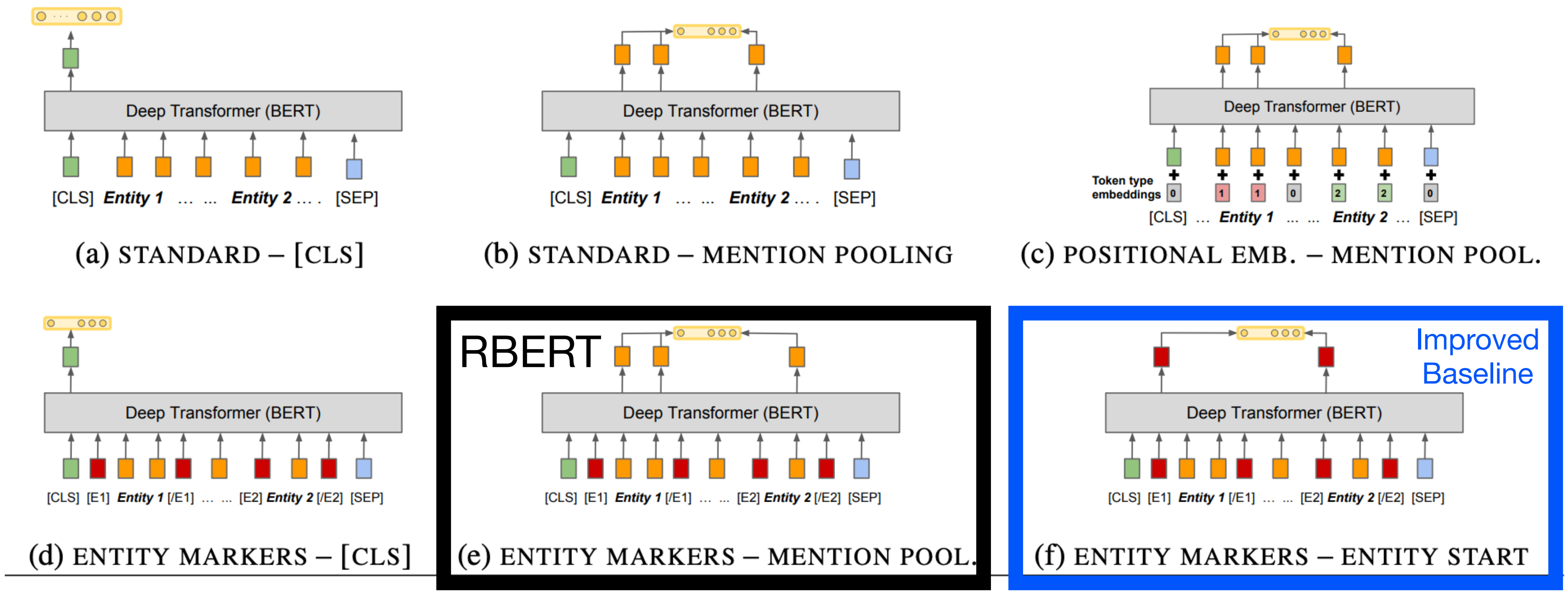
Appendix

System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	**93.1**
Fine-tuning approach		
$\text{BERT}_{\text{LARGE}}$	96.6	92.8
$\text{BERT}_{\text{BASE}}$	96.4	92.4
Feature-based approach ($\text{BERT}_{\text{BASE}}$)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

Last 4 Hidden States Token Sampling (BERT Paper)

Appendix



RBERT and Improved Baseline token sampling strategy from the last hidden state

Appendix

		SemEval 2010 Task 8		KBP37		TACRED		FewRel 5-way-1-shot
# training annotated examples		8,000 (6,500 for dev)		15,916		68,120		44,800
# relation types		19		37		42		100
		Dev F1	Test F1	Dev F1	Test F1	Dev F1	Test F1	Dev Acc.
Wang et al. (2016)*		–	88.0	–	–	–	–	–
Zhang and Wang (2015)*		–	79.6	–	58.8	–	–	–
Bilan and Roth (2018)*		–	84.8	–	–	–	68.2	–
Han et al. (2018)		–	–	–	–	–	–	71.6
Input type	**Output type**							
STANDARD	[CLS]	71.6	–	41.3	–	23.4	–	85.2
STANDARD	MENTION POOL.	78.8	–	48.3	–	66.7	–	87.5
POSITIONAL EMB.	MENTION POOL.	79.1	–	32.5	–	63.9	–	87.5
ENTITY MARKERS	[CLS]	81.2	–	68.7	–	65.7	–	85.2
ENTITY MARKERS	MENTION POOL.	80.4	–	68.2	–	69.5	–	87.6
ENTITY MARKERS	ENTITY START	**82.1**	**89.2**	**70**	**68.3**	**70.1**	**70.1**	**88.9**

Gains from the RBERT and Improved Baseline token sampling strategy

Appendix

	PLBART Encoder Input	PLBART Decoder Input
S	def maximum (a , b , c) : NEW_LINE INDENT return max ([a , b , c]) <python>	<En> Find the maximum of three numbers
G	Find the maximum of three numbers <En>	<java> public int maximum (int a , int b , int c) { return Math . max (a , Math . max (b , c)) }
T	public int maximum (int a , int b , int c) { return Math . max (a , Math . max (b , c)) } <java>	<python> def maximum (a , b , c) : NEW_LINE INDENT return max ([a , b , c])

Table 3: Example inputs to the encoder and decoder for fine-tuning PLBART on sequence generation tasks: source code summarization (S), generation (G), and translation (T).

PLBART's <python> token used as special token when pretraining.