Intent Detection with DistilBERT on CLINC150 Dataset

1st Davide Spada
Alma Mater Studiorum - University of Bologna
dept. of Computer Science, Science and Engineering
Bologna, Italy
davide.spada5@studio.unibo.it

***Abstract*—This project focuses on training and evaluating a Transformer model, DistilBERT, for the task of Intent Detection. The objective is to classify natural language sentences, typical of interactions with chatbots or voice assistants, into one of 150 distinct intents provided by the CLINC150 dataset. Model benchmarking was performed using Accuracy and F1-score on intent classes as evaluation metrics. The obtained results demonstrate DistilBERT's effectiveness in recognizing intents, highlighting its potential for practical applications in Natural Language Processing (NLP).**

I. Introduction

Intent detection is a foundational task in Natural Language Understanding (NLU), particularly relevant to dialogue systems such as chatbots and virtual assistants. Its goal is to classify a user utterance into a predefined category representing the speaker's intent, such as booking a flight, checking the weather, or transferring money.

While earlier approaches relied on rule-based methods or traditional machine learning algorithms with handcrafted features, recent advances in Transformer-based architectures have dramatically improved performance. Among these, BERT and its distilled variant DistilBERT stand out for their ability to capture semantic relationships within language.

In this project, we investigate the use of DistilBERT — a lightweight and faster alternative to BERT — for fine-tuning on the CLINC150 dataset, a large-scale benchmark containing 150 diverse intent classes across ten domains. The primary objective is to adapt the pretrained DistilBERT model to this multi-class classification task and evaluate its performance using standard metrics such as accuracy and F1-score.

The resulting model is intended to be compact, fast, and accurate, suitable for real-world deployment in dialogue systems running on resource-constrained environments.

II. Dataset Description

The CLINC150 dataset is a comprehensive benchmark designed for evaluating intent detection systems in task-oriented dialogue applications. It includes a total of 150 intent classes grouped into 10 high-level domains (e.g., travel, banking, weather, calendar, and more).

Each intent class is associated with 100 training samples, 20 validation samples, and 30 test samples, resulting in a well-balanced and diverse dataset. In total, the dataset consists of 15,000 training utterances, 3,100 validation utterances, and 5,500 test utterances.

Table I
CLINC150 Dataset Statistics

Split	# Utterances	# Intents	# Domains
Train	15,250	150	10
Validation	3,100	150	10
Test	5,500	150	10

An example from the dataset includes the sentence:

"can you pay it now" → **intent: "pay_bill"**

The dataset is available via the Hugging Face `datasets` library under the configuration `clinc_oos, "plus"`, which includes both in-scope and out-of-scope intents [4].

A brief exploratory analysis, conducted in the notebook, revealed that the average length of utterances is approximately 8.2 tokens, with a maximum of around 28 tokens. This insight informed the maximum sequence length used in tokenization. The large number of fine-grained intent classes makes CLINC150 a challenging benchmark, particularly due to the semantic overlap between some intents (e.g., "weather" vs "temperature" or "transfer" in different domains), increasing the importance of contextual understanding.

III. Model Architecture

BERT (Bidirectional Encoder Representations from Transformers) is a seminal model in natural language processing that introduced a new way of capturing contextual information using a deep stack of transformer encoders. Its architecture is based entirely on self-attention mechanisms, which allow the model to consider the entire input sequence at each layer, enabling it to model complex dependencies between tokens regardless of their distance in the sequence.

The base version of BERT consists of 12 encoder layers (transformer blocks), each with 12 self-attention heads and a hidden size of 768. Every token in the input sequence is embedded into a vector and passed through the stack of transformers, where multi-head attention layers compute contextual representations. This attention mechanism enables the model to dynamically weigh the importance of different words in the input relative to one another, which is crucial for

tasks that require understanding subtle semantic relationships, such as intent classification.

Despite its strong performance, BERT is computationally expensive. In real-time systems, such as voice assistants or mobile-based chatbots, inference time and memory consumption are critical constraints. To address these limitations, we use `distilbert-base-uncased`, a distilled version of BERT introduced by Sanh et al. [2]. DistilBERT is trained using knowledge distillation — a process in which a smaller "student" model is trained to reproduce the behavior of a larger "teacher" model (in this case, BERT) by minimizing a combination of standard task loss and a distillation loss.

DistilBERT preserves most of BERT's representational power while being significantly more efficient: it has 6 transformer layers instead of 12, effectively halving the depth. Each layer still contains 12 self-attention heads and uses the same hidden dimensionality of 768. The model retains BERT's layer normalization, residual connections, and GELU (Gaussian Error Linear Unit) activation functions, which provide smooth and effective non-linearities for deep learning.

The attention mechanism in DistilBERT works identically to that of BERT: each attention head learns to focus on different aspects of the input context, and their outputs are concatenated and linearly projected to form the input for the next layer. This multi-head structure allows the model to simultaneously capture diverse semantic features across the sequence.

To adapt DistilBERT for intent classification, a linear classification head is added on top of the [CLS] token representation from the final encoder layer. This head projects the contextualized embedding into a 150-dimensional vector, where each component corresponds to a specific intent class. A softmax activation is then used to produce a probability distribution over all classes.

All layers of the model are fine-tuned during training, allowing the pretrained weights to be adjusted specifically for the intent detection task. This approach enables the model to retain general language understanding from pretraining while specializing in distinguishing between the 150 fine-grained intent classes of the CLINC150 dataset.

IV. Preprocessing and Tokenization

Before fine-tuning a transformer model like DistilBERT on a downstream task such as intent classification, it is essential to preprocess the raw input data into a format compatible with the model's architecture and training framework.

The input to our model consists of short user utterances in plain text (e.g., "what is my balance"), each associated with a corresponding intent label (e.g., `check_balance`). However, transformer models like DistilBERT operate on sequences of token IDs — integers that map to subword units in the model's vocabulary. Therefore, the textual inputs must be converted into tokenized and numerically encoded representations.

The preprocessing pipeline used in this project involves the following key steps:

1) **Tokenization:** We use the `DistilBertTokenizerFast` from Hugging Face, which applies WordPiece tokenization. This process breaks down words into subword units (e.g., `"checking"` → `"check"` + `"##ing"`), allowing the model to handle rare or out-of-vocabulary words more effectively. Special tokens such as `[CLS]` and `[SEP]` are also automatically inserted, as required by the model's architecture.
2) **Truncation and Padding:** Since transformer models require fixed-length input sequences for efficient batch processing, all tokenized sequences are either truncated or padded to a maximum length. Based on an analysis of sentence lengths in the dataset, we set this value to 64 tokens — sufficient to cover more than 99% of utterances after subword tokenization. Padding is applied using a special `[PAD]` token, and an attention mask is created to distinguish between real tokens and padding.
3) **Label Encoding:** The intent labels, originally represented as strings, are mapped to integer class indices using the predefined label set provided in the dataset. This ensures compatibility with the classification head of the model, which outputs a single integer class prediction per input.
4) **Dataset Wrapping:** Once tokenized, the inputs (token IDs and attention masks) and the encoded labels are wrapped into a custom PyTorch `Dataset` object. This wrapper enables efficient loading, batching, and shuffling of data during training on GPU using the Hugging Face `Trainer` API.

A tokenized example looks as follows:

```
"can you pay it now" ⇒ [CLS], can,
you, pay, it, now, [SEP], [PAD],
..., [PAD]
```

This sequence is then converted into:

- **Input IDs:** A list of integers corresponding to the tokenizer vocabulary.
- **Attention Mask:** A binary vector indicating which tokens are actual inputs (1) and which are padding (0).
- **Label:** An integer representing the target intent class.

This structured format allows the model to attend only to meaningful tokens during training and ensures that all input sequences are of uniform length, facilitating GPU parallelization and batch optimization.

The same preprocessing pipeline is applied consistently to the training, validation, and test sets to guarantee a homogeneous data format across all phases of model development.

V. Training Setup

The model is fine-tuned using the `Trainer` API provided by the Hugging Face Transformers library, which offers a flexible and efficient interface for managing training, evaluation, logging, and checkpointing in PyTorch-based transformer models. The training setup is designed to ensure both performance and robustness. We use the AdamW [**?**] optimizer with weight decay to prevent overfitting by discouraging large weight

values. A low learning rate (2×10^{-5}) and linear warmup strategy with 500 steps help stabilize early training and avoid divergence. Validation is performed at the end of each epoch to monitor the model's generalization ability in real time.

To select the best model, we use checkpointing based on the weighted F1-score on the validation set. Only the best-performing model is retained, reducing the risk of selecting an overfitted checkpoint. Gradient clipping is automatically applied by the Trainer to avoid exploding gradients, and all training is performed in full precision on a CUDA-enabled GPU.

This configuration achieves a good balance between computational efficiency and model performance. It also avoids common training pitfalls:

- **Overfitting:** Controlled via weight decay, early stopping through validation monitoring, and evaluation-based model selection.
- **Underfitting:** Avoided by fully fine-tuning all DistilBERT layers for 10 epochs with sufficient capacity to learn complex intent patterns.
- **Unstable training:** Mitigated by warmup, low learning rate, and logging every 100 steps to track performance trends.

The training process is monitored using loss and metric curves, which help visualize the learning dynamics. Figure 1 shows how training and validation loss evolve across epochs, while Figure 2 shows the progression of accuracy and F1-score on the validation set.

The following results were obtained during the training:

Table II
PERFORMANCE OF DISTILBERT AT THE END OF THE TRAINING

Metric	Score
Accuracy	0.956
F1-score (weighted)	0.953
F1-score (macro)	0.951

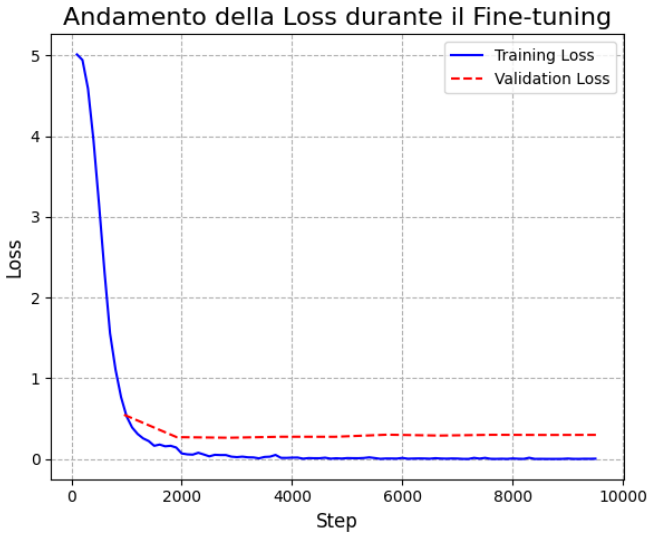

Figure 1. Training and validation loss during training

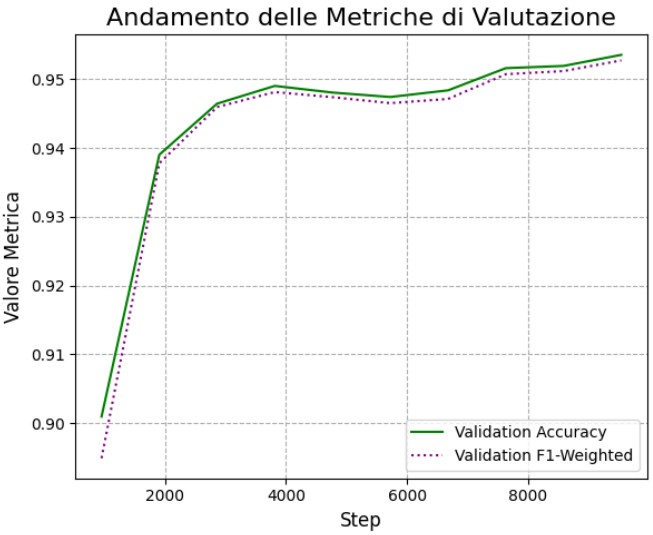

Figure 2. Validation accuracy and F1-score during training

VI. EVALUATION AND COMPARISON

After training, we evaluate the final model on the held-out test set, which includes 5,500 user utterances uniformly distributed across 150 intent classes. The evaluation is conducted using the `Trainer.predict()` method provided by Hugging Face, which computes predictions and automatically returns a dictionary of evaluation metrics.

We assess the model using three key metrics:

- **Accuracy:** Measures the overall proportion of correctly predicted examples. While intuitive and easy to interpret, accuracy alone can be misleading in multi-class tasks if classes are imbalanced or some are harder to learn than others.
- **Weighted F1-Score:** Computes the F1-score for each class and averages them, weighting each class's contribution by its support (number of true instances). This metric balances precision and recall while accounting for label frequency, making it appropriate for real-world datasets with class imbalance or uneven complexity.
- **Macro F1-Score:** Averages the F1-score across all classes without weighting. This gives equal importance to all intent classes, regardless of their frequency, and is particularly useful in our case where some classes may be semantically ambiguous or challenging to predict.

These results indicate that the model generalizes well to unseen examples, with high agreement between predicted and true intent labels across a large number of fine-grained classes. The results obtained on the test set are shown in VI (Table 3). A full classification report is also generated, showing precision, recall, and F1-score for each of the 150 intent labels. While most classes achieve near-perfect performance, a few rare or semantically overlapping intents (e.g.,

Table III
TEST SET PERFORMANCE OF DISTILBERT

Metric	Score
Accuracy	0.8785
F1-score (weighted)	0.8715
F1-score (macro)	0.9110

`credit_limit` vs `credit_score`) present more frequent misclassifications. These errors typically arise from shared vocabulary or ambiguous user phrasing.

Overall, the model demonstrates robust performance, with an error rate of less than 15% despite the task's complexity and the distilled model. For reference, preliminary experiments using a simple logistic regression baseline resulted in over 10 percentage points lower accuracy, underscoring the value of fine-tuning a pretrained transformer model on domain-specific tasks.

VII. ERROR ANALYSIS

To better understand the model's limitations, we performed a manual inspection of the misclassified test examples. Out of 5,500 utterances, a total of 668 were classified incorrectly, resulting in an error rate of approximately 12.1%.

We analyzed a random sample of misclassifications and identified recurring error patterns. The most prominent source of confusion involved the special `oos` (out-of-scope) class, which is designed to capture inputs that do not belong to any of the 150 defined intents. This class is particularly difficult to detect because it does not correspond to a concrete semantic domain.

- **Out-of-Scope Misclassification:** Many utterances labeled as `oos` were assigned to semantically plausible in-scope intents. For example:

  "what is harry's real name"
  True: `oos`
  Predicted: `are_you_a_bot`

  In this case, the model misinterprets the informal tone and assigns it to a chatbot-related intent.

  "compare prices of malt-o-meal to kelloggs"
  True: `oos`
  Predicted: `calories`

  Here, the model seems to focus on brand names and food terms, ignoring the comparative nature of the query.
- **Semantically Similar Intents:** In some cases, the predicted intent is close in meaning to the correct one. For example:

  "tell jim i'm coming home soon"
  True: `text`
  Predicted: `make_call`

  Although both intents refer to communication, the model fails to distinguish between texting and calling.
- **Ambiguous Context:** Short or syntactically ambiguous sentences can mislead the model:

  "i need to up by noon"
  True: `alarm`
  Predicted: `how_busy`

  The lack of a verb such as "wake" or "set" likely caused the model to misinterpret the request as calendar-related.
- **Real-World Requests Misinterpreted:** Some realistic but complex utterances were incorrectly mapped to specific in-scope intents:

  "what stores are at my local mall"
  True: `oos`
  Predicted: `shopping_list`

  The presence of the word "stores" leads the model to infer a grocery-related intent.

These examples illustrate that while the model performs well in general, it struggles in specific areas:

- distinguishing `oos` examples from borderline in-scope queries,
- handling informal or incomplete phrasing,
- resolving semantic ambiguity when multiple intents are conceptually close.

VIII. CONCLUSION AND FUTURE WORK

This work presented the fine-tuning of a transformer-based model for the task of intent detection. The approach proved to be effective on a dataset containing a wide range of user queries, showing that even relatively lightweight models can achieve good results when applied to practical natural language understanding problems.

The overall training and evaluation process was straightforward and did not require major modifications to the underlying architecture. The model was able to generalize well on unseen data, confirming the usefulness of pretrained language models in classification tasks.

There are several simple directions that could be explored in the future. For example, the same approach could be applied to other datasets to test the model's adaptability, or alternative models could be used to compare results and evaluate differences in performance. It might also be interesting to vary some training settings, such as the number of epochs or the size of the training data, to observe their impact.

Another possible development is the integration of this model into a real-world application, such as a chatbot or voice assistant, to test how it performs in more dynamic and interactive environments.

These types of experiments could help better understand the strengths and limitations of the model and offer ideas for improving future versions.

APPENDIX

A. Tokenizer and Dataset Preparation

```
from transformers import DistilBertTokenizerFast

tokenizer = DistilBertTokenizerFast.from_pretrained('
    ↪ distilbert-base-uncased')
MAX_LENGTH = 64  # Maximum sequence length

class Clinc150Dataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: val[idx].clone().detach() for key, val
    ↪ in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx],
    ↪ dtype=torch.long)
        return item

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

def tokenize_and_create_dataset(texts, labels, tokenizer,
    ↪ max_length):
    encodings = tokenizer(texts,
                          truncation=True,
                          padding='max_length',
                          max_length=max_length,
                          return_tensors='pt')
    return Clinc150Dataset(encodings, labels)

# Dataset creation
train_dataset = tokenize_and_create_dataset(train_texts,
    ↪ train_labels_encoded, tokenizer, MAX_LENGTH)
val_dataset = tokenize_and_create_dataset(val_texts,
    ↪ val_labels_encoded, tokenizer, MAX_LENGTH)
test_dataset = tokenize_and_create_dataset(test_texts,
    ↪ test_labels_encoded, tokenizer, MAX_LENGTH)
```

Listing 1. Tokenizer and Dataset creation

B. Evaluation Metrics Function

```
from sklearn.metrics import accuracy_score, f1_score

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    f1_weighted = f1_score(labels, preds, average='weighted'
    ↪ )
    f1_macro = f1_score(labels, preds, average='macro')
    acc = accuracy_score(labels, preds)
    return {
        'accuracy': acc,
        'f1_weighted': f1_weighted,
        'f1_macro': f1_macro
    }
```

Listing 2. Metrics computation

C. Training Configuration

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=10,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=100,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="f1_weighted",
    report_to="none",
    save_total_limit=1,
)
```

Listing 3. TrainingArguments configuration

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of NAACL-HLT*, 2019.

[2] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," arXiv preprint arXiv:1910.01108, 2019.

[3] T. Wolf et al., "Transformers: State-of-the-Art Natural Language Processing," in *EMNLP: System Demonstrations*, 2020.

[4] Larson, S. et al., "An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction," in *EMNLP*, 2019.