Sentiment Analysis Project Documentation

Chapter 2: ML Model Output

2.1 Intro to ML Models and Serialization

Serialization is the process of saving a trained model into a format that allows it to be reloaded later for inference or continued training. In machine learning workflows, serialization is crucial for deployment and reproducibility.

🧠 Think of serialization as saving a model's brain so it can be reloaded later without retraining.

Common formats include:

- PyTorch: `.pt`, `.pth`, or Hugging Face format

- TensorFlow: SavedModel, `.h5`

- ONNX: `.onnx` for cross-framework compatibility

2.2 Saving & Loading in PyTorch and TensorFlow

PyTorch:

```
# Save
torch.save(model.state_dict(), 'model_weights.pth')

# Load
model.load_state_dict(torch.load('model_weights.pth'))
model.eval()
```

Hugging Face-style:

```
model.save_pretrained('model/')
tokenizer.save_pretrained('model/')
```

TensorFlow:

```
# Save
model.save('saved_model/')

# Load
model = tf.keras.models.load_model('saved_model/')
```

2.3 Tokenizer & Preprocessing Files Explained

Tokenizer artifacts ensure that raw text is converted into consistent numerical inputs:

- `vocab.txt`: Vocabulary of tokens

- `tokenizer_config.json`: Tokenizer behavior (e.g., lowercasing, padding)

- `special_tokens_map.json`: Designates `[CLS]`, `[PAD]`, etc.

These files must be used alongside the model to ensure reliable preprocessing.

2.4 Understanding Hugging Face Outputs

Calling the model returns a `ModelOutput` object:

```
outputs = model(**inputs)
logits = outputs.logits
predicted_class = torch.argmax(logits, dim=1)
```

This abstraction keeps outputs clean and ready for downstream tasks (classification, regression, etc).

📦 Common Output Files (Especially with Hugging Face Transformers & BERT)

Let's explain the files you mentioned first:

Filename	Description	Used By	Format
`model.safetensors` or `pytorch_model.bin`	Model weights (state_dict)	PyTorch, Transformers	Binary (PyTorch format)
`config.json`	Model architecture & hyperparameters	Hugging Face Transformers	JSON
`tokenizer_config.json`	Tokenizer-specific parameters	Transformers	JSON
`special_tokens_map.json`	Maps special tokens like `[CLS]`, `[PAD]`	Transformers	JSON
`vocab.txt` or `merges.txt` + `vocab.json`	Tokenizer vocabulary	BERT uses `vocab.txt`, GPT uses BPE files	Text/JSON
`label_encoder.pkl`	Serialized label encoder (e.g., for classification)	Scikit-learn or manually created	Pickle

2.5 Interoperability: ONNX and Cross-framework Conversion

ONNX (Open Neural Network Exchange) allows converting PyTorch or TensorFlow models into a format that can be run across platforms (C++, JavaScript, etc.):

```
# Example for exporting PyTorch model
torch.onnx.export(model, dummy_input, 'model.onnx')
```

🔄 ONNX is ideal for embedded deployment, browser inference, or cross-platform systems.

2.6 Deployment Ready: Choosing Formats for Inference

Format	Best Use Case
.pt/.pth:	PyTorch experiments
HuggingFace:	Cloud-based inference, NLP tasks
.onnx:	Cross-platform, mobile, edge
SavedModel:	TensorFlow/Keras pipelines

Use `.safetensors` for secure & efficient model loading.

⚙️ Deployment Tools and Their Compatible Formats

Tool	Compatible Formats	Use Case
TorchServe	`.mar` (archived PyTorch model)	PyTorch REST API server
TensorFlow Serving	`SavedModel/`	High-performance inference
ONNX Runtime	`.onnx`	Cross-platform, optimized runtime
Triton Inference Server	PyTorch, TensorFlow, ONNX, XGBoost	Scalable deployment
FastAPI + PyTorch/TF	`.pt`, `.h5`, `.pkl`	Lightweight REST APIs
TFLite	`.tflite`	Mobile/IoT inference
TensorFlow.js	`.json` + weights	Browser-based inference
Hugging Face Inference API	Hugging Face model hub files	No backend needed

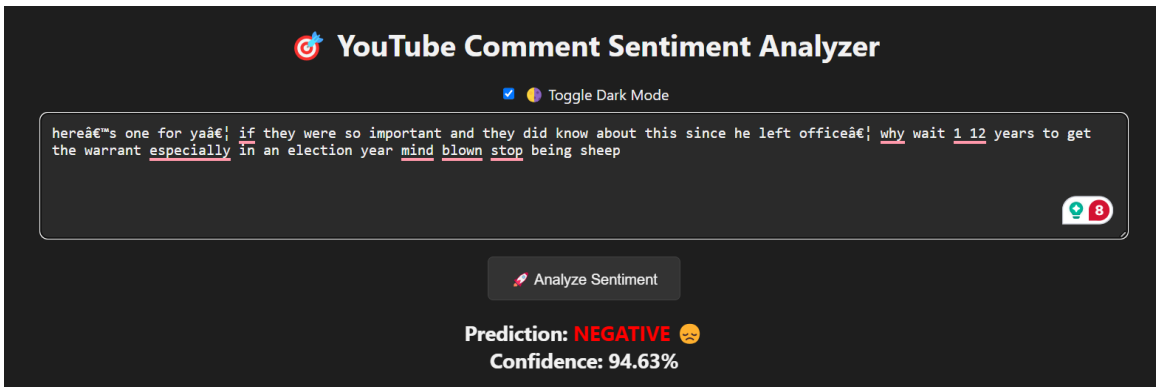
2.7 Case Study: YouTube Comment Classification with BERT

Model: `BertForSequenceClassification`

Dataset: 18k YouTube comments (balanced manually)

Output: POSITIVE or NEGATIVE + confidence score

Sample Input/Output:



2.8 Serving with FastAPI, TorchServe, and Hugging Face

- **FastAPI**: Lightweight Python backend
- **TorchServe**: Model server for PyTorch (less used here)
- **Hugging Face Inference API**: GPU-backed hosting & zero setup

Deployment snippet (FastAPI):

```
@app.post("/predict")
async def predict(input: InputText):
    tokens = tokenizer(input.text, return_tensors="pt")
    outputs = model(**tokens)
    prediction = torch.argmax(outputs.logits, dim=1)
    return {"sentiment": label[prediction.item()]}
```

2.9 Best Practices for Model Packaging

- Always save tokenizer and model together
- Compress models for deployment
- Use .safetensors for speed and safety
- Include requirements.txt for reproducibility
- Add version metadata (e.g., config.json with model info)

2.10 Appendix: File Format Cheat Sheet

File Name	Purpose
config.json	Model architecture
pytorch_model.bin	Weights (unsafe format)
model.safetensors	Weights (safe + fast)
vocab.txt	Tokenizer vocabulary
tokenizer_config.json	Preprocessing behavior
special_tokens_map.json	Special token mapping