🧩 Step 1: Tokenizer (AutoTokenizer)

python

CopyEdit

```
inputs = tokenizer("apple stock has crash", return_tensors="pt")
```

🔍 What Happens:

1. **Text is split** into *subword tokens* using WordPiece:

css

CopyEdit

```
["[CLS]", "apple", "stock", "has", "crash", "[SEP]"]
```

2. **Tokens are mapped** to IDs based on FinBERT's vocabulary:

yaml

CopyEdit

```
[101, 1653, 4518, 2038, 6245, 102]
```

(IDs may vary — just examples)

3. **Other tensors** are created:
    - attention_mask: [1, 1, 1, 1, 1, 1]
    - token_type_ids: [0, 0, 0, 0, 0, 0] (not always used)

These are output as **PyTorch tensors** to feed into the model.

🧩 Step 2: Embedding Layer (inside FinBERT)

text

CopyEdit

```
[101, 1653, 4518, 2038, 6245, 102]
↓
Each token ID is turned into a 768-dimensional vector
↓
[vec₁, vec₂, ..., vec₆] → shape: (1, 6, 768)
```

🔍 What Happens:

- Each token ID is mapped to a **learned embedding vector**
- Three types of embeddings are **added together**:
    - **Token Embeddings** (word meaning)

- **Position Embeddings** (token order in sentence)
- **Segment Embeddings** (used for sentence pairs, not here)

🧩 Step 3: Transformer Encoder Layers (Self-Attention)

python

CopyEdit

```
output = model(**inputs)
```

🔍 Internally:

FinBERT passes the embedded tokens through **12 layers of transformers**.

Each layer does:

1. **Multi-head self-attention**:
   - Every token "looks at" all other tokens in the sequence.
   - Computes *attention weights* to decide what's important.
   - Example: "crash" may attend more to "stock" or "apple".
2. **Feed-forward neural network**:
   - Refines contextual info.
3. **Residual connections + Layer Norm**:
   - Stabilize and speed up training/inference.

By the end of layer 12, every token embedding is *contextualized* — meaning it encodes knowledge of surrounding words.

🧩 Step 4: Classification Head (Finetuned)

The [CLS] token (first token) is used as the **summary representation** of the sentence.

This goes through a **fully connected (dense) layer** and outputs:

python

CopyEdit

```
logits = [-1.43, 0.12, 2.17]  # [negative, neutral, positive]
```

🧩 Step 5: Softmax + Argmax

python

CopyEdit

probs = softmax(logits) → [0.02, 0.06, 0.92]

label = argmax(probs) → index 2 → "positive"

💡 In reality for your example (“apple stock has crash”), the model would likely give:

python

CopyEdit

```
[2.05, 0.15, -1.2] → softmax → [0.88, 0.10, 0.02] → "negative"
```

✅ Final Output

- **Prediction**: "negative"
- **Confidence**:

json

CopyEdit

```json
{
  "negative": 0.88,
  "neutral": 0.10,
  "positive": 0.02
}
```

🔁 Full Flow Summary (Visually)

mermaid

CopyEdit

```mermaid
flowchart TD
  A["apple stock has crash"]
  A --> B[Tokenizer]
  B --> C[Token IDs → Embeddings]
  C --> D[Transformer Layers (Self-Attention)]
  D --> E[Classification Head (CLS token)]
  E --> F[Logits → Softmax]
  F --> G[Final Label: Negative]
```