IndoML Datathon Report - Phase 2

1. Team Information

Team Name: Dumb Ducks

Team Members:

- Santhosh G S - santhoshgs013@gmail.com
- Eshan Gujarathi - eshangujarathi15@gmail.com
- Adithya Sai Lenka - adithya.lenka@gmail.com

2. Detailed Problem-Solving Approach

2.1 Models Explored

We experimented with the following models during our development process. However, we finalized the Flan T5 Small model for our final submission.

1. Flan T5 Small
2. Flan T5 Base

2.2 Architecture Details

Flan T5 Small

We utilized the Flan T5 Small model. Flan T5 is a variant of the T5 (Text-to-Text Transfer Transformer) model that has been fine-tuned on a diverse set of tasks using instruction tuning. The small version has approximately 80 million parameters.

Key features of Flan T5 Small:

- Encoder-decoder architecture
- 6 layers in both encoder and decoder
- 12 attention heads
- 512 hidden dimension size

Flan T5 Base

We experimented with the Flan T5 Base model as well. The base version has approximately 250 million parameters.

Key features of Flan T5 Base:

- Encoder-decoder architecture
- 12 layers in both encoder and decoder
- 12 attention heads
- 768 hidden dimension size

2.3 Model Selection Rationale

Our final choice of the Flan-T5 Small model was based on comprehensive analysis of both the problem structure and practical considerations:

2.3.1 Data Relationship Analysis

1. **Hierarchical Dependencies**
    - Identified a clear hierarchical relationship between the first three labels:
        - Supergroup → Group → Module (cascading dependencies)
    - Each level has a one-to-many relationship with the next level
    - Brand stands as an independent variable, capable of appearing across multiple supergroups
2. **Problem Structure Considerations**
    - The task exhibits characteristics of:
        - Sequential label prediction (for hierarchical categories) - Autoregressive prediction
        - Input could be processed parallely for increased throughput

2.3.2 Model Architecture Requirements

1. **Encoder-Decoder Architecture Benefits**
    - **Encoder**: Processes input text features in parallel, capturing semantic meaning
    - **Decoder**: Enables autoregressive predictions, crucial for handling hierarchical dependencies
2. **Flan-T5 Small Advantages**
    - **Pre-training**:
        - Instruction-tuned on 1000+ downstream tasks
        - Enhanced zero-shot and few-shot capabilities
        - Strong natural language understanding capabilities
    - **Parameter Efficiency**:
        - 80M parameters (compared to 250M in base version)
        - Faster inference time

2.3.3 Performance Metrics

1. **Accuracy Metrics**
   - Item Classification Accuracy: 41%
   - Comparable performance to larger Flan-T5 Base model
2. **Computational Efficiency**
   - 3.7x smaller model size compared to base version
   - Lower computational requirements during inference and faster training iterations

2.4 Novel Aspects of Final Solution

1. **Enhanced Feature Engineering**
   - Incorporation of semantically meaningful product descriptions beyond basic product titles
   - Utilization of FAISS (Facebook AI Semantic Search) for efficient semantic similarity search
   - Semantic matching of titles with external datasets and retrieving descriptions
2. **Efficient Data Enrichment Pipeline**
   - Implementation of high-performance vector similarity search
   - Processing time optimization from potential 180 hours to 16 hours total
   - Scalable approach for handling large-scale product datasets

2.5 Learning Process and Experiments

Successful Approaches:

1. **FAISS Implementation**
   - Implementation details:
     - Used all-MiniLM-L6-v2 model for generating 384-dimensional embeddings
     - Implemented efficient indexing of external dataset titles
     - Achieved 70ms query time per data point
   - Results:
     - Successfully processed training data in 12 hours
     - Processed test data in 4 hours
     - Improved item accuracy by ~5% (from 36% to 41%)
2. **External Dataset Integration**
   - Implementation details:
     - Utilized Amazon Product Dataset from Hugging Face Hub
     - Implemented cosine similarity-based matching
     - Retrieved relevant product descriptions for semantic enrichment
   - Results:
     - Enhanced model understanding of product context

- Downside:
  - If we manually inspect the retrievals, the product description were totally different from the actual product descriptions on a very high level inspection

Failed Experiments:

1. **Web Scraping Approach**
   - Why it didn't work:
     - Projected 180 hours for training data alone
     - Legal restrictions from Google and Amazon
   - Learnings:
     - Need for more efficient data collection methods
     - Importance of considering legal implications
     - Value of exploring alternative data sources
2. **Traditional Vector Database Implementation**
   - Why it didn't work:
     - Excessive database creation time
     - Poor scalability with large datasets
     - High computational resource requirements
   - Learnings:
     - Importance of efficient indexing mechanisms
     - Value of specialized tools like FAISS

2.6 Performance Improvement Techniques

1. **Semantic Data Enrichment**
   - Implementation:
     - Integration of external product descriptions
     - Use of sentence transformers for embedding generation
     - Efficient similarity matching using FAISS
   - Impact on results:
     - 5% improvement in item accuracy
     - Enhanced model understanding of product context
     - More robust feature representation
2. **Model Architecture Optimization**
   - Implementation:
     - Tested both small and base versions of Flan-T5
     - Experimented with different epoch counts
   - Impact on results:
     - Maintained consistent 39-41% accuracy
     - Identified optimal model size (small)
     - Determined optimal training duration (3 epochs)

3. Performance Benchmarks

3.1 Training Performance Analysis

Number of model parameters that are used for training
76961152

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg	# of Calls	Total MFLOPs
aten::mm	1.65%	27.797ms	5.86%	98.936ms	227.438us	7.480ms	3.09%	7.480ms	17.197us	435	48633.741
aten::bmm	0.25%	4.250ms	0.43%	7.345ms	51.005us	3.904ms	1.61%	3.904ms	27.114us	144	1985.790
aten::mul	1.72%	29.015ms	3.37%	57.013ms	84.463us	1.783ms	0.74%	1.783ms	2.641us	675	53.616
aten::add	0.21%	3.524ms	0.37%	6.216ms	40.361us	343.040us	0.14%	343.040us	2.228us	154	9.800
aten::empty	0.15%	2.558ms	0.22%	3.662ms	9.272us	0.000us	0.00%	0.000us	0.000us	395	--
aten::random_	0.00%	36.920us	0.00%	36.920us	18.460us	0.000us	0.00%	0.000us	0.000us	2	--
aten::item	0.03%	480.170us	0.03%	571.680us	1.489us	0.000us	0.00%	0.000us	0.000us	384	--
aten::_local_scalar_dense	0.01%	91.510us	0.01%	91.510us	0.238us	0.000us	0.00%	0.000us	0.000us	384	--
enumerate(DataLoader)#_SingleProcessDataLoaderIter._...	0.08%	1.404ms	0.11%	1.802ms	900.771us	0.000us	0.00%	0.000us	0.000us	2	--
aten::randperm	0.01%	119.246us	0.01%	235.818us	58.955us	0.000us	0.00%	0.000us	0.000us	4	--

Self CPU time total: 1.689s
Self CUDA time total: 241.826ms

GFLOPs during training
50.682946828

The data shows detailed performance metrics during model training with 76,961,152 parameters. The model consumes most resources at 50.68 GFLOPs total throughput.

3.2 Inference Performance Analysis

```
Inference time :0.46082472801208496
GFLOPs during testing
15.197619447
```

The model achieves an inference time of 0.46 seconds with 15.19 GFLOPs during testing.

4. Setup Instructions

4.1 Project Structure

```
submission/
    ├── data/
    ├── logs/
    ├── creating_ext_vectors.py
    ├── descriptions_retrieval.py
    ├── tuning_t5.py
    ├── Efficiency.ipynb
    ├── Data_Analysis.ipynb
    ├── requirements.txt
    ├── Dockerfile
    └── entrypoint.sh
```

4.2 File Purposes and Execution Order

1. Creating External Vectors (Optional - Pre-processed data already provided)

File: `creating_ext_vectors.py`

- **Purpose:** Creates vector embeddings (`ext_vectors.npy`) from external Amazon product dataset
- **Output:** Generates `ext_vectors.npy` required for description retrieval
- **Note:** This step can be skipped as the processed data is already provided

2. Description Retrieval (Optional - Pre-processed data already provided)

File: `descriptions_retrieval.py`

- **Purpose:** Uses FAISS to find similar products and retrieve their descriptions
- **Dependencies:** Requires `ext_vectors.npy` from step 1
- **Outputs:**

    - `processed_train_features.csv`
    - `processed_test_features.csv`
- **Note:** These files are already available in the data directory

3. Model Training and Evaluation

File: `tuning_t5.py`

- **Purpose:** Main script for training and evaluating the Flan-T5 model
- **Input:** Uses processed CSV files from the data directory
- **Functionality:**
    - Loads and preprocesses data
    - Trains the Flan-T5 model
    - Performs evaluation and generates predictions

4. Analysis Notebooks (For insights and benchmarking)

Files:

1. `Data_Analysis.ipynb`
    - **Purpose:** Provides detailed analysis of the dataset
    - **Contents:**
        - Checking for redundancies in dataset
        - Checking for unique labels
        - Label relationship exploration
2. `Efficiency.ipynb`
    - **Purpose:** Benchmarks and performance analysis
    - **Contents:**
        - Model efficiency metrics
        - Resource utilization
        -

4.3 Environment Setup

Option 1: Using Docker

```
# Build the Docker image

docker build -t product_classifier .

# Run the container

docker run -it --name product_classifier product_classifier
```

Option 2: Local Setup

```
# Create virtual environment
python3 -m venv venv

# Activate environment
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows

# Install requirements
pip install --upgrade pip
pip install -r requirements.txt
```

5. Execution Instructions

5.1 Using Docker Container

After running the container, you'll see the following instructions:

```
===========================================
To activate the virtual environment, run:
    . venv/bin/activate
Once activated, you can run:
    python tuning_t5.py
===========================================
```

Follow these steps:

1. Activate the virtual environment:

   ```
   . venv/bin/activate
   ```

2. Run the training script:

   ```
   python tuning_t5.py
   ```

5.2 Local Execution

If running locally after environment setup:

```
# Activate virtual environment (if not already activated)
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows


# Run the training script
python tuning_t5.py
```

5.3 Additional Notes

- The `data/` directory already contains all necessary processed files
- Steps 1 and 2 (creating vectors and retrieving descriptions) are provided for reproducibility but can be skipped
- The main training script (`tuning_t5.py`) is self-contained and works directly with the provided processed data
- Analysis notebooks can be run independently for additional insights

6. Computational Requirements

6.1 Development Environment

Our team used the shared instance of the following setup:

- **GPU:** NVIDIA A100 (80GB VRAM)
- **CPU:** AMD EPYC Processor
- **RAM:** 220GB

6.2 Minimum System Requirements

- CUDA-capable GPU with at least 16GB VRAM
- 32GB RAM

- 50GB available storage
- Python 3.8+

Note: The code can run on CPU, but GPU is highly recommended for reasonable training and inference times.