Customer Query Categorization using DistilBERT

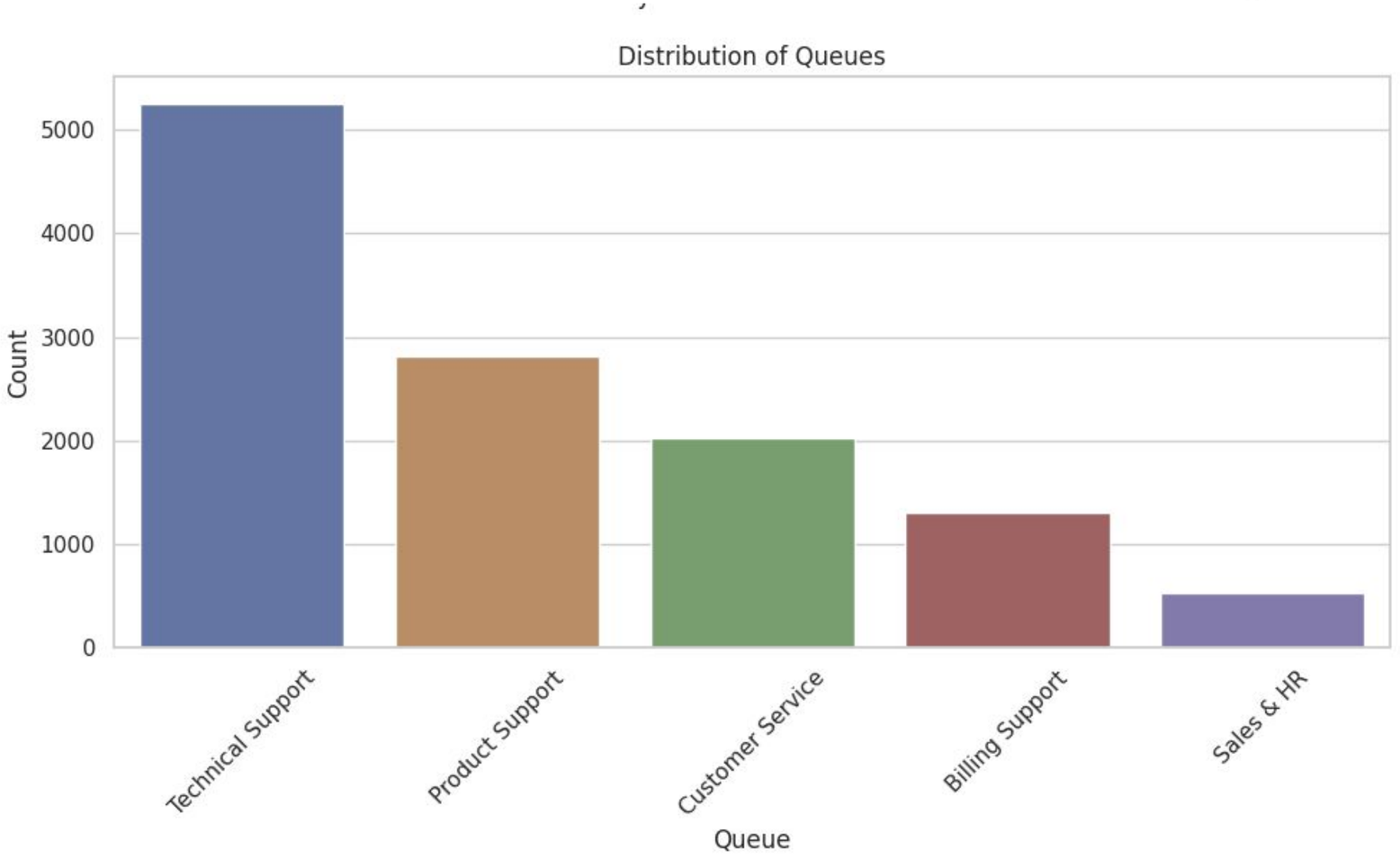
Dataset

- Customer Support Query dataset.
- Total **11923** data points.
- **Subject**
- **Body**
- Predict **queue** (department).
- **5** unique values of queue.

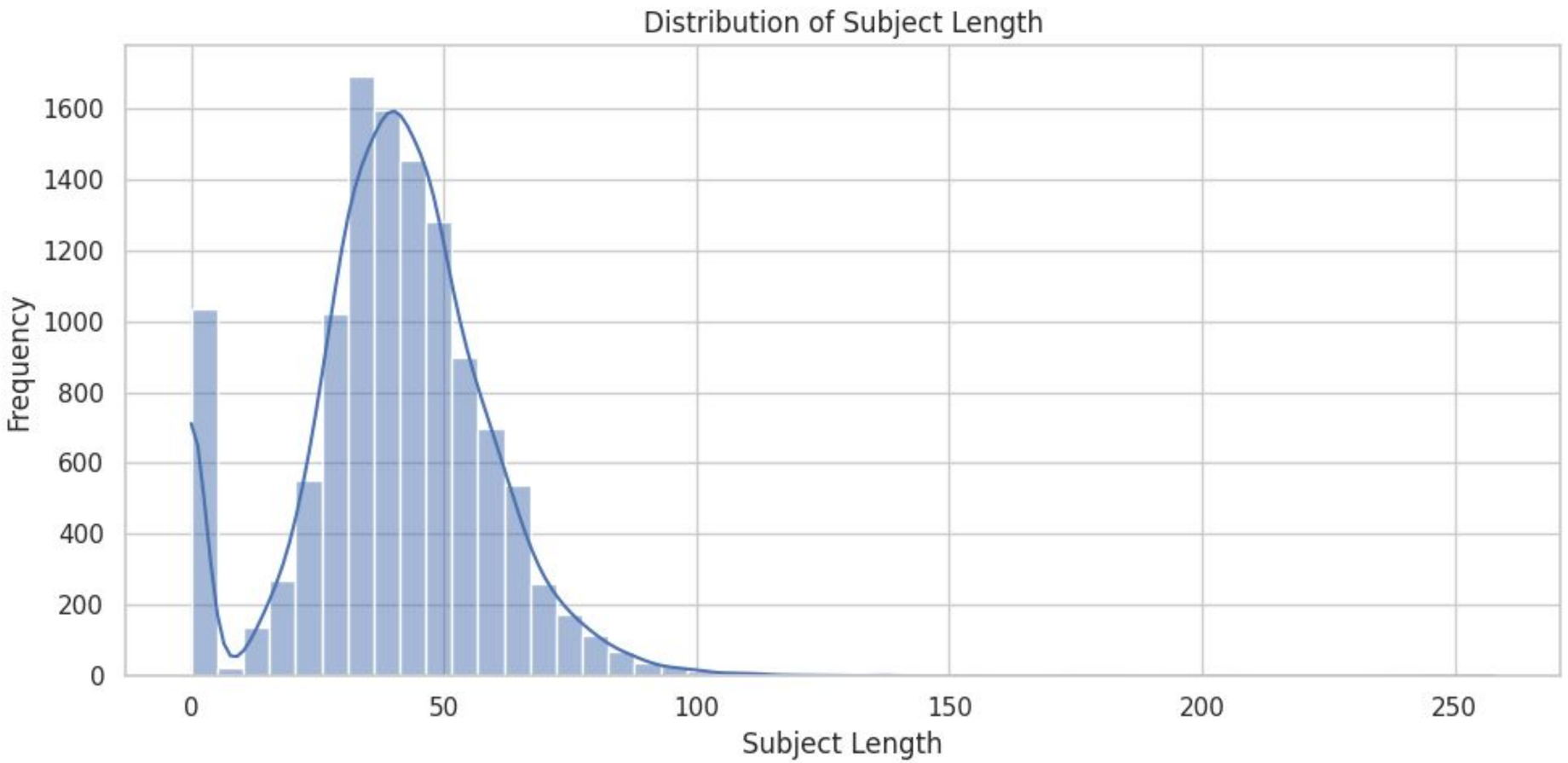
Preprocessing

- **Dropped** columns which are not needed
- Filled missing Subject or Body with **space** for concatenation
- Text cleaning with regular expressions
  - HTML tags
  - URLs
  - Email ids
  - Phone numbers
  - Non ASCII characters
- Dropped any duplicate rows.
- **Encoded queue** since this is categorical

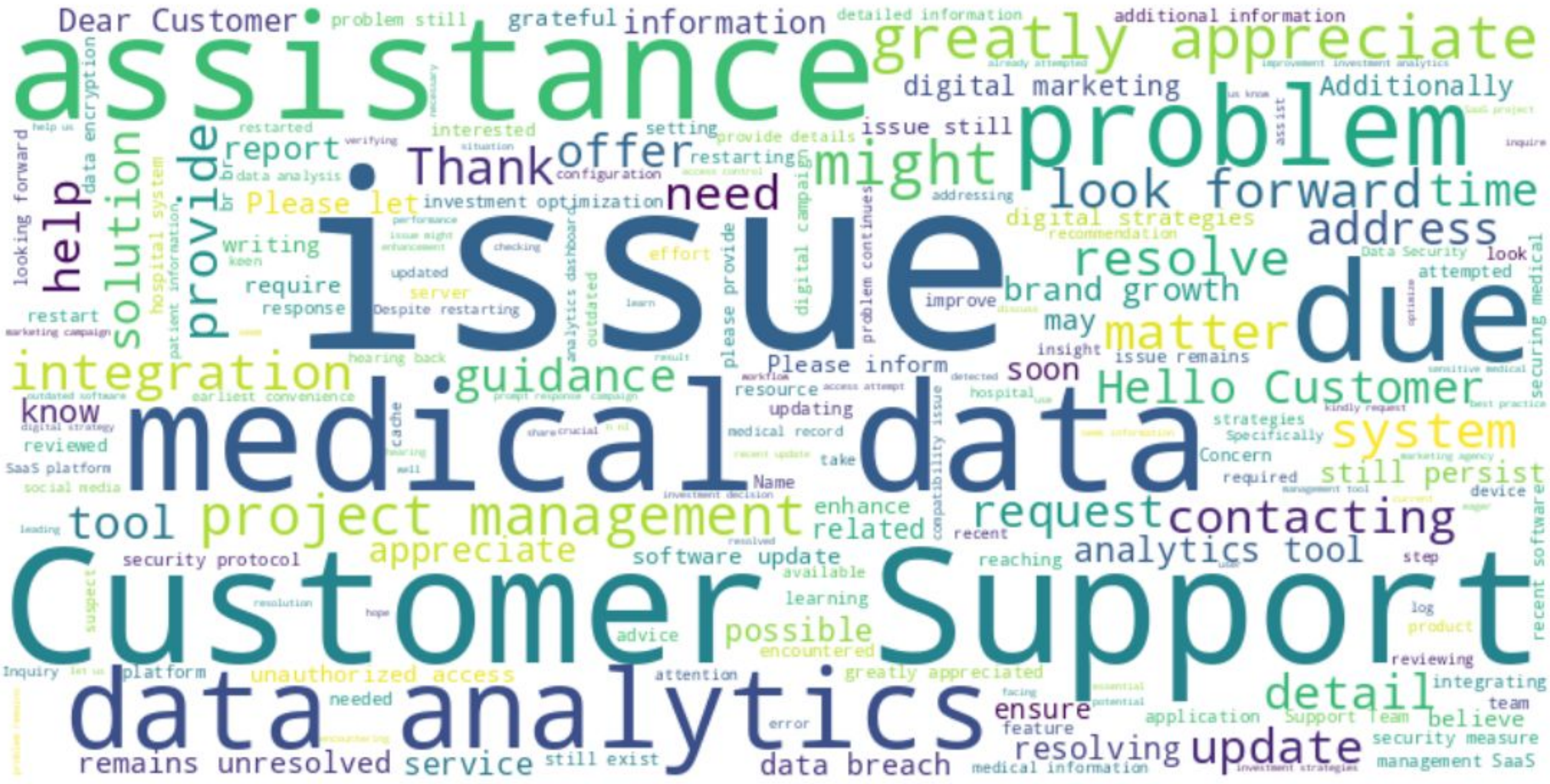
Datapoints Distribution



Subject Length



Word Cloud

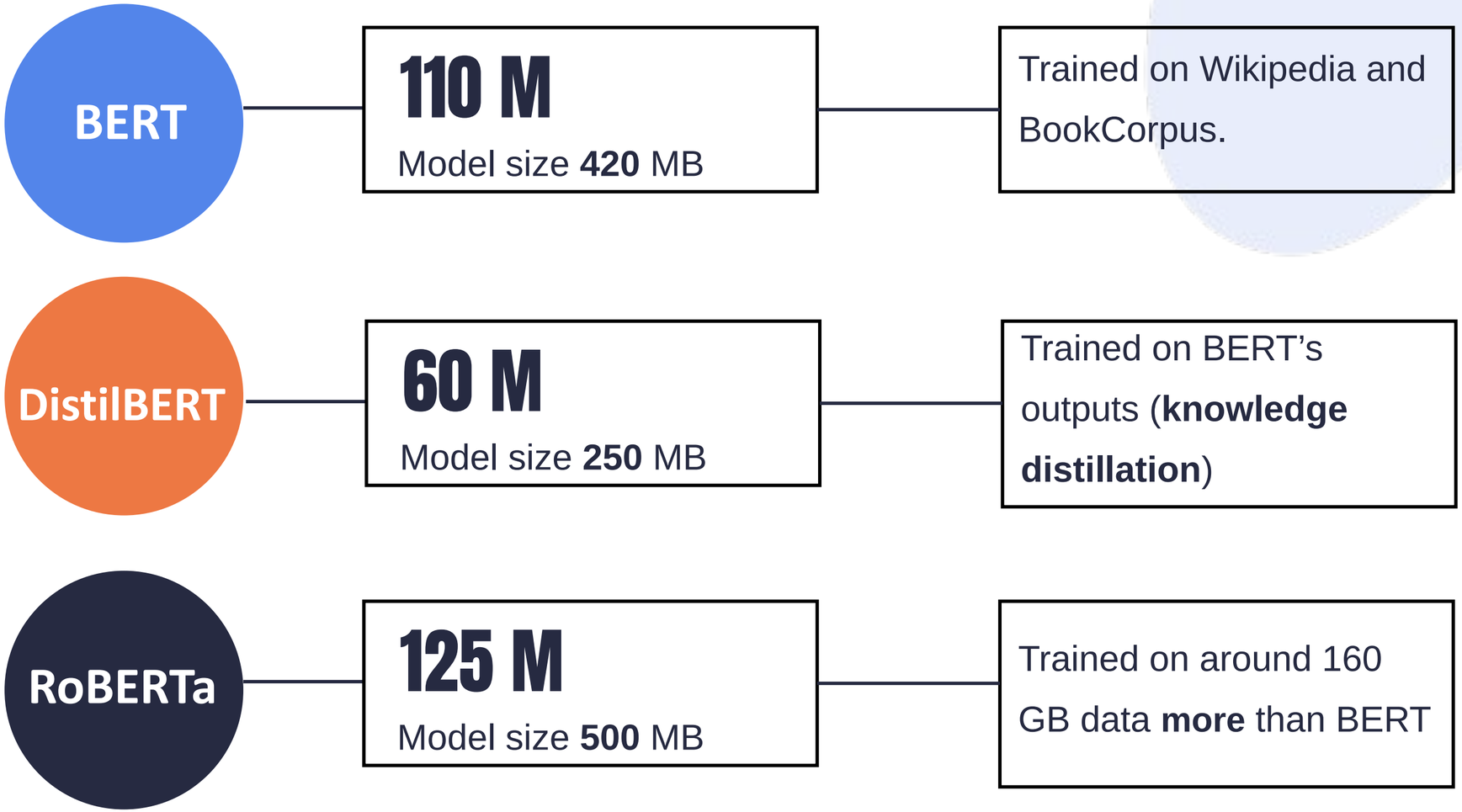


Word Cloud of Full Text

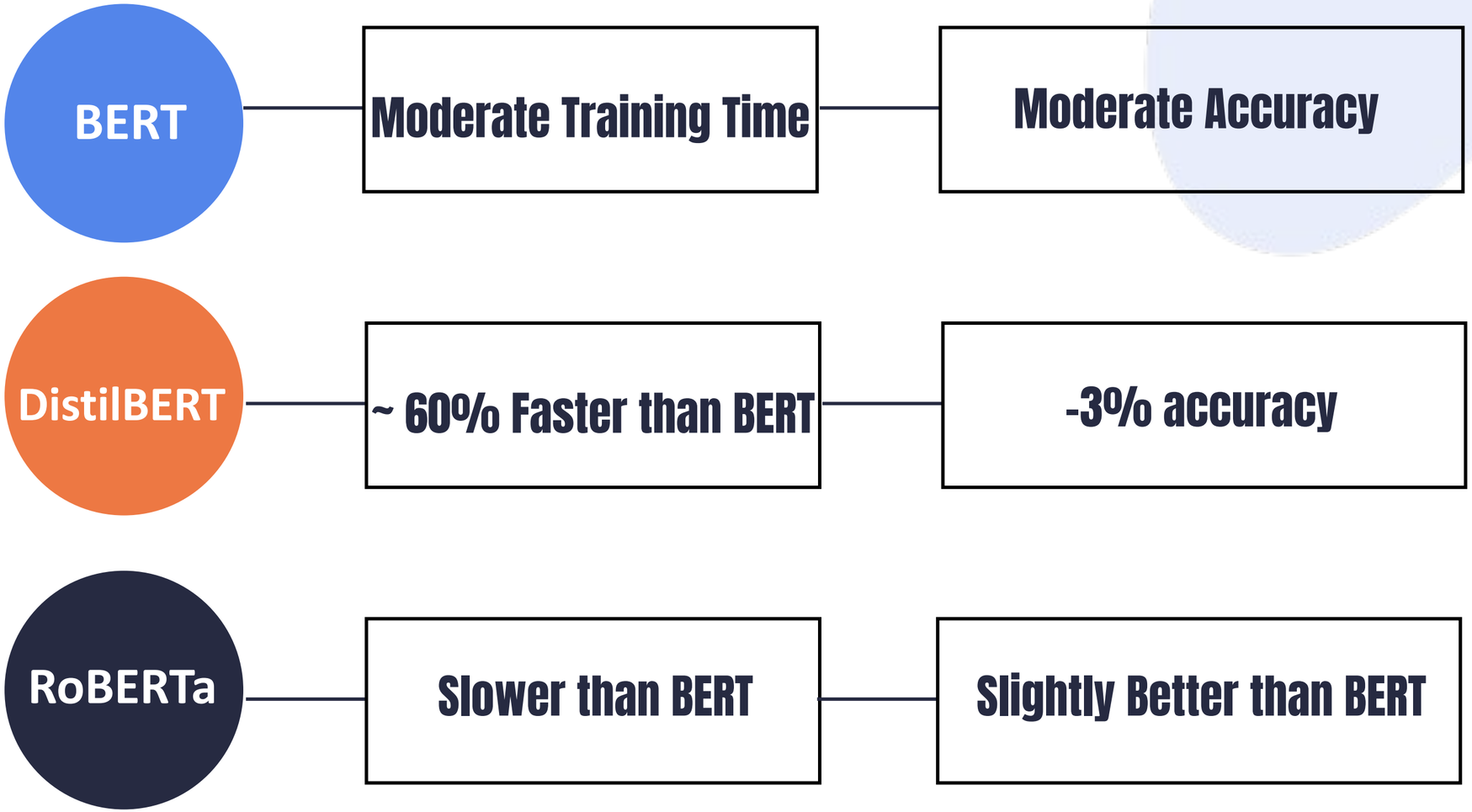
Model Selection

- BERT models
- **Transformer** based so **parallel** processing
- Read text **bidirectionally** improving accuracy
- Already trained on large corpus
- Fine tuned easily for downstream task

BERT Based Models



Training and Accuracy (BERT)



Training on 3 Models

- Loaded the dataset
- *full_text* as **feature** and *queue* as **label**
- **Split dataset** into train and validation
- Defined ***TicketDataset*** class to prepare data to fed into BERT models
- ***CustomTrainer*** extending HF Trainer class
  - Uses class weights in in loss function
  - To handle class imbalance
  - Penalise mis-classification of minority classes

loss_fn = torch.nn.CrossEntropyLoss(weight=self.class_weights)

Training Loop

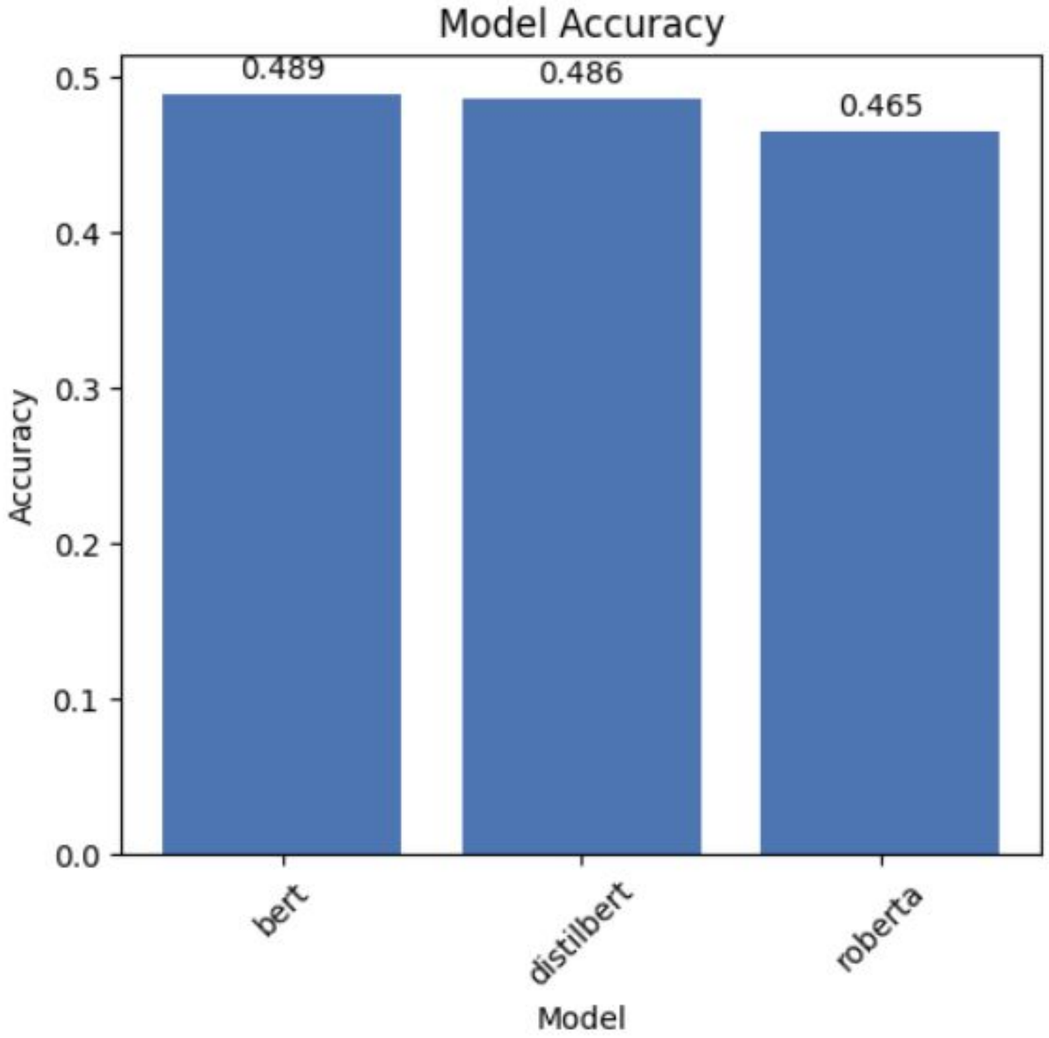
- Load Transformer *AutoModelForSequenceClassification*
- Load Tokenizer *AutoTokenizer*
- Adds padding for RoBERTa
- **Calculates class weights** using *compute_class_weight*
- Configured training arguments
  - 3 epocs
  - Per epoc logging and evaluation
  - Uses GPU

Training Loop

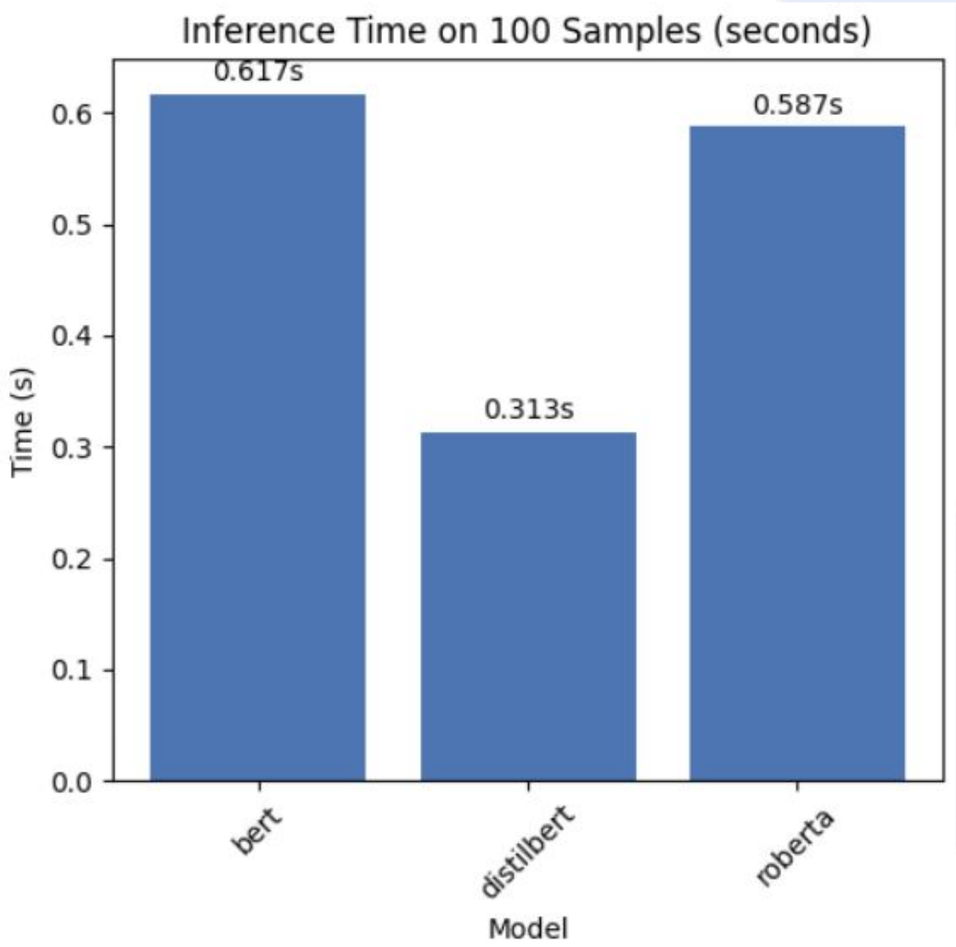
- Measure training time
- Checks **inference time** on 100 data points
- Takes care of memory management

  torch.cuda.empty_cache()
- Return result
  - Model Name
  - Training Time
  - Inference Time
  - Validation Accuracy

Results of Comparison



Results of Comparison



Summarization Needed?

- DistilBERT has token limit of 512
- Truncates long sentences
- **Might loose meaning**
- *facebook/bart-large-cnn*
- Our dataset

```
Text length statistics:
Mean words: 62.7
Median words: 58.0
Max words: 276
Texts longer than 300 words: 0
```

Downsampling

```
Original class distribution:
queue
Technical Support    5245
Product Support      2814
Customer Service     2027
Billing Support      1302
Sales & HR            535
Name: count, dtype: int64
```

```
New (downsampled) class distribution:
queue
Product Support      535
Sales & HR           535
Technical Support    535
Billing Support      535
Customer Service     535
Name: count, dtype: int64
```

Downsampling

Accuracy - 0.506542

With downsampling dataset becomes small


```
Dataset split:
Training samples: 2140
Validation samples: 535
```

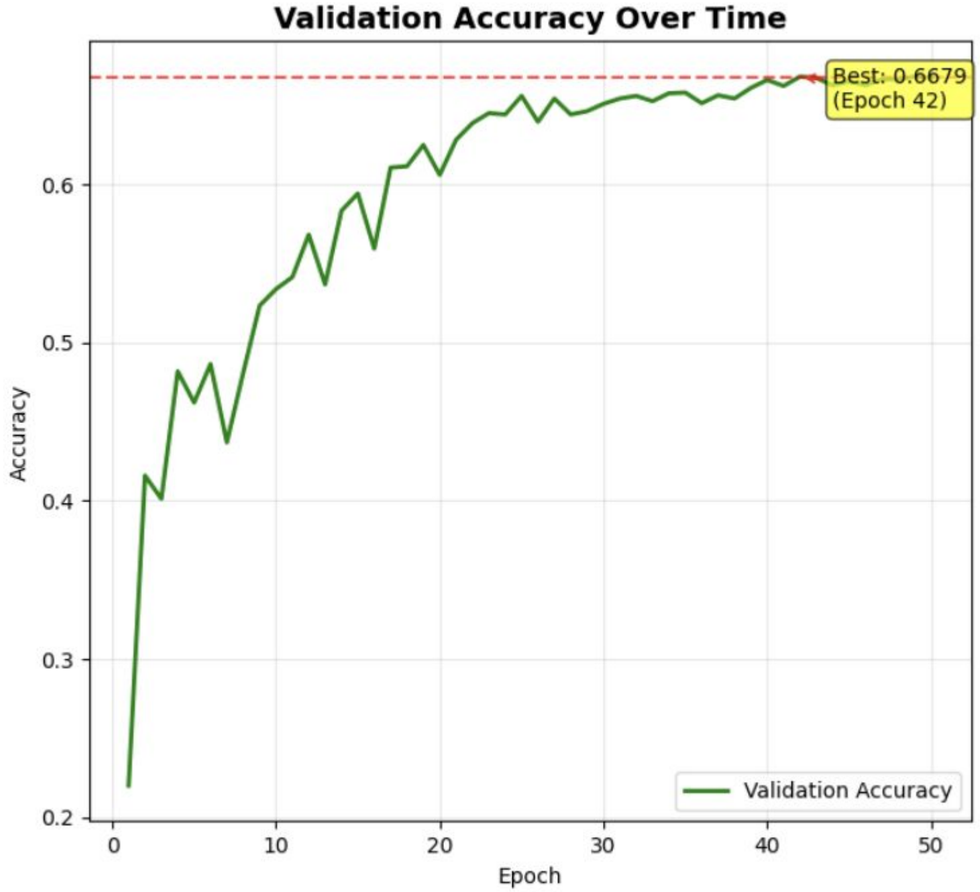
Final Model Used

- **DistilBERT**
- Full Dataset - **11923** samples
- Prediction for queue -
  - Billing Support
  - Customer Service
  - Product Support
  - Sales & HR
  - Technical Support
- ***TicketDataset*** to prepare datasets
- ***CustomTrainer*** with class weights

Final Model Used

- Epocs - **30**
- Batch size - **64**
- Learning Rate - **2e-5**
- Eval Strategy - Every Epoch
- **GPU**
- Saved:
  - Model weights
  - Tokenizer
  - Label encoder
  - Training config

Training Result



Final Results

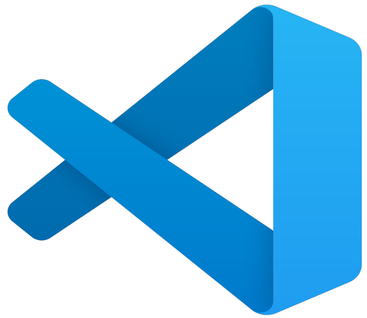
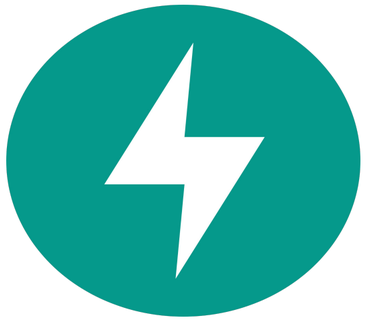
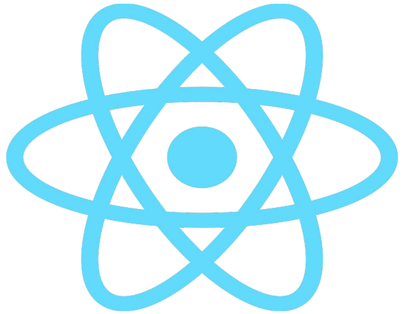
Detailed Classification Report:

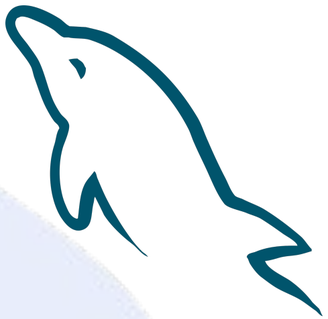
	precision	recall	f1-score	support
Billing Support	0.8559	0.7769	0.8145	260
Customer Service	0.4954	0.5345	0.5142	406
Product Support	0.5747	0.5737	0.5742	563
Sales & HR	0.5854	0.4486	0.5079	107
Technical Support	0.7404	0.7531	0.7467	1049
accuracy			0.6625	2385
macro avg	0.6504	0.6174	0.6315	2385
weighted avg	0.6652	0.6625	0.6631	2385

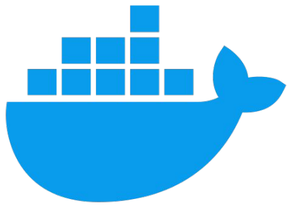
Training Time: 6025.57 seconds (1.67 hours)
Final Validation Accuracy: 0.6625
Inference Time (100 samples): 0.3029 seconds
Average Time per Sample: 0.003029 seconds

Deployment

Tools Used

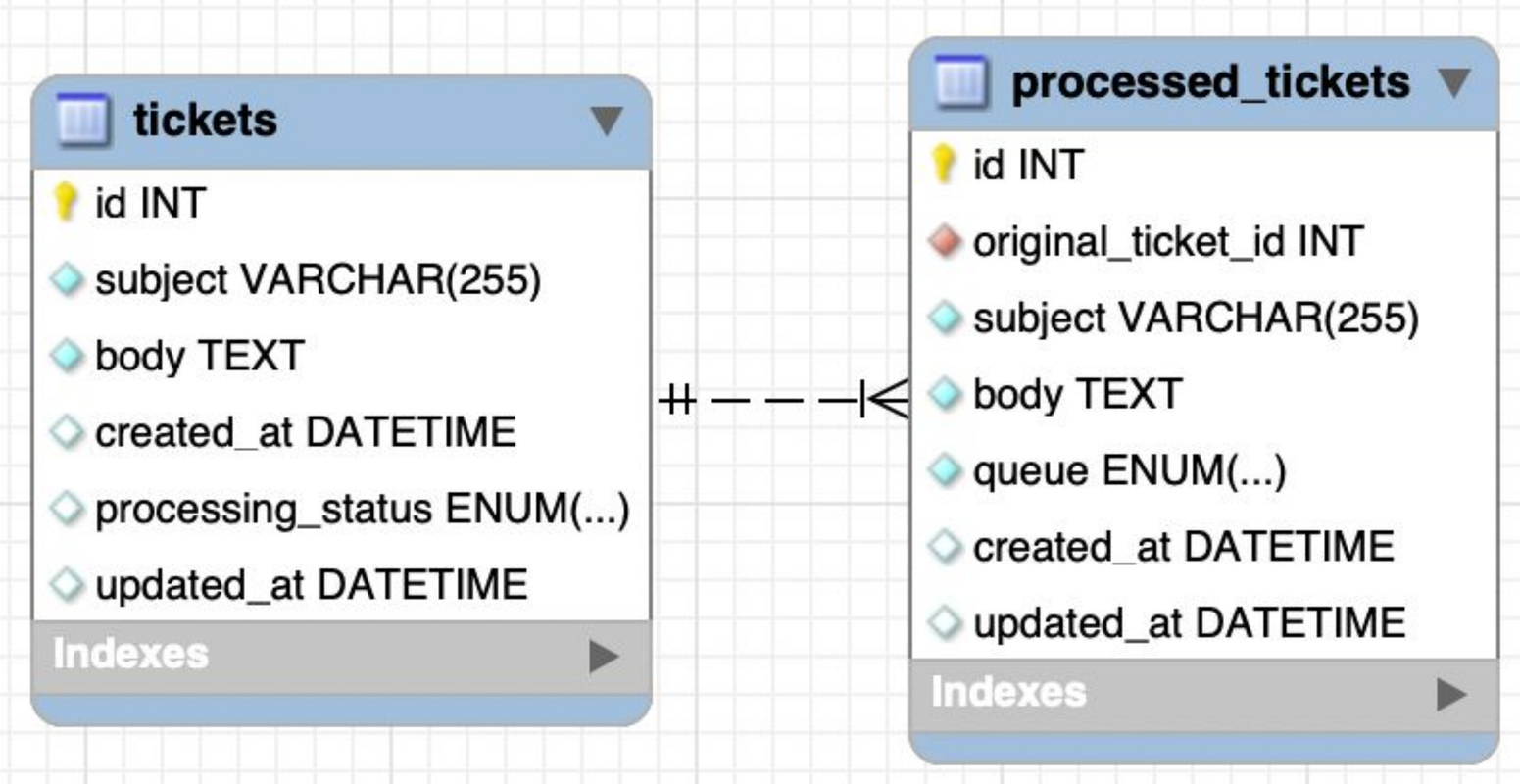






Database

- MySQL
- Database - *ticket_system*



Backend

- *FastAPI*
- *SQLAlchemy* as ORM
- *pydantic* for request and response schemas
- *asyncio* for non-blocking background task
- Model is placed in the backend folder
- **ModelService** loads the model
- **TicketProcessor** runs every **10** seconds
- Updates ticket status from Pending -> Processing -> Completed/Failed

Backend APIs

tickets

POST	`/tickets/`	Create Ticket
GET	`/tickets/`	Get All Tickets
GET	`/tickets/{ticket_id}`	Get Ticket

support

GET	`/support/tickets`	Get Processed Tickets
GET	`/support/tickets/{ticket_id}`	Get Processed Ticket
GET	`/support/dropdown-options`	Get Dropdown Options
PUT	`/support/tickets/{ticket_id}/queue`	Update Ticket Queue

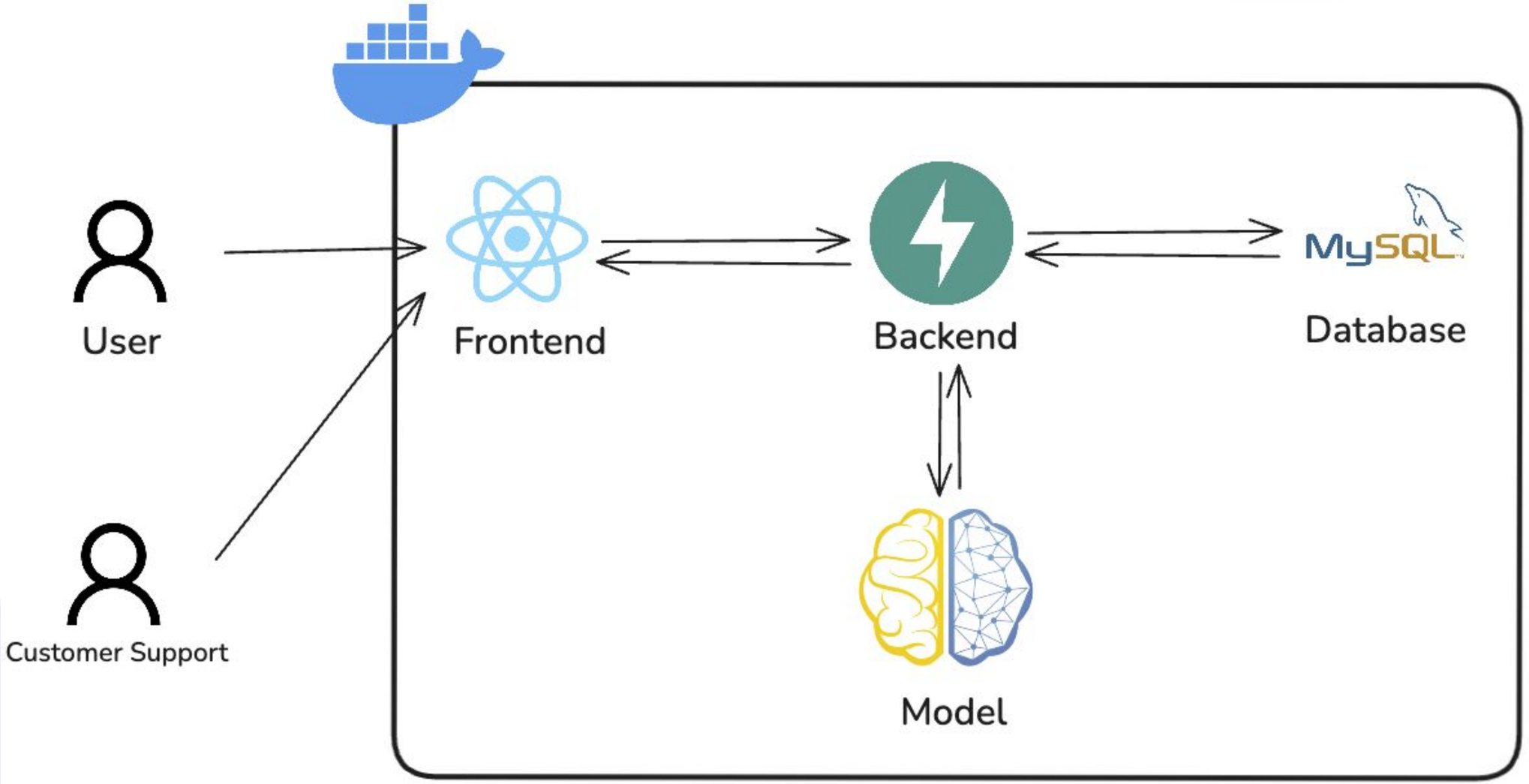
FrontEnd

- Using React via Vite
- **axios** for API calls
- User Page -
  - Subject
  - Body
  - Trying to **simulate an email**
- Customer Support Page -
  - Can see tickets in their queue
  - Can **change the queue** if wrongly classified

Containerization

- Separate containers for Backend, Frontend and Database
- Containers managed through **Docker Compose**
- Ports
  - Backend - 8000
  - Frontend - 1573
  - MySQL - 3307
- Backend waits for MySQL to be ready (avoids **Segmentation Fault**)
- used **volume** *mysql_data*
- Frontend - *nginx*

Final Workflow



Demo