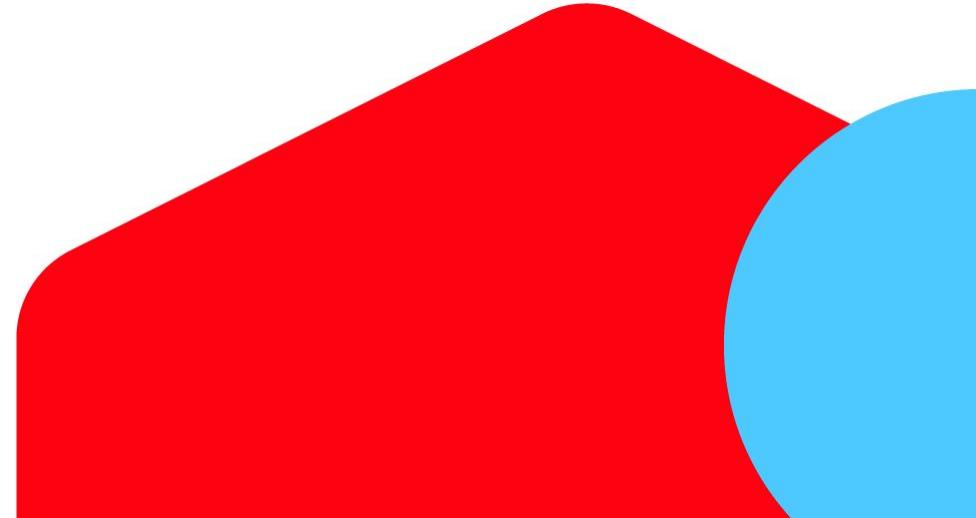


AI for Efficient Routing of Customer Inquiries at Mercari



Prashant Anand

mercari





Prashant Anand

- ML Engineer at Mercari
- Working in the Customer Support domain
- [@primaprashant](https://twitter.com/primaprashant)



| Takeaways

- 01 Translate business problem to ML one**
- 02 Solving the ML problem using NLP and Python**
- 03 Insights into the routing of customer inquiries**

| Table of Contents

01 Understanding the business problem

02 Baseline solution

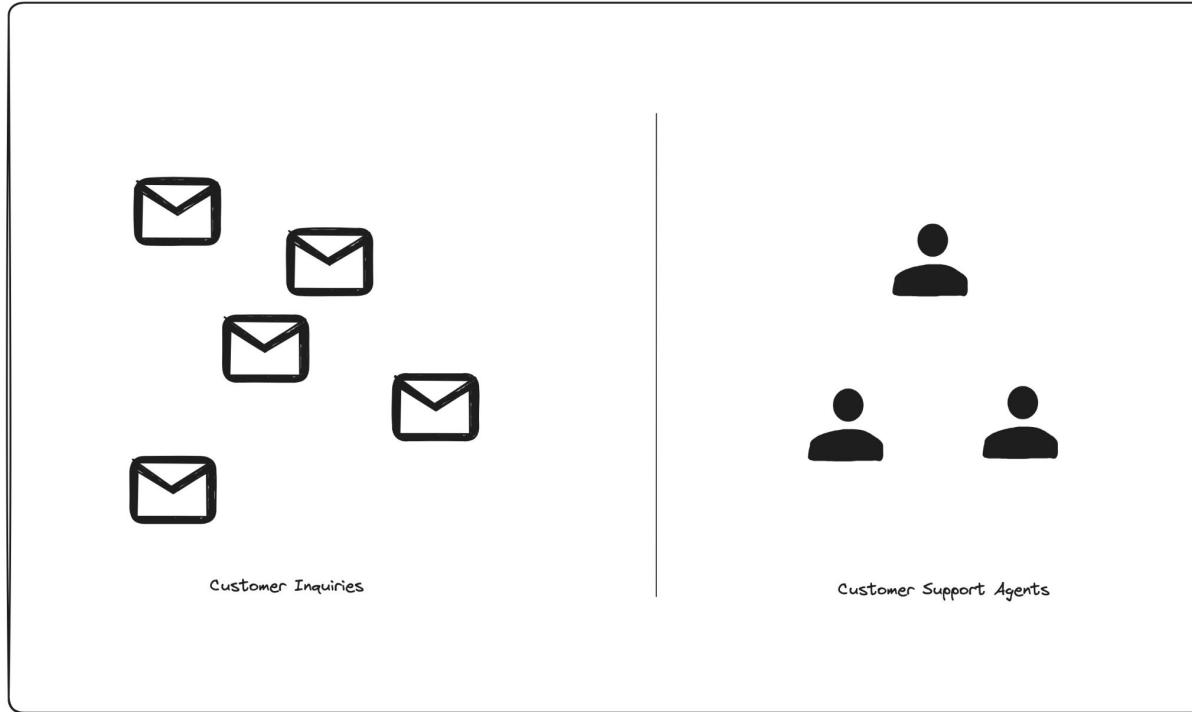
03 Developing the ML solution

04 Business impact

01

Understanding the business problem

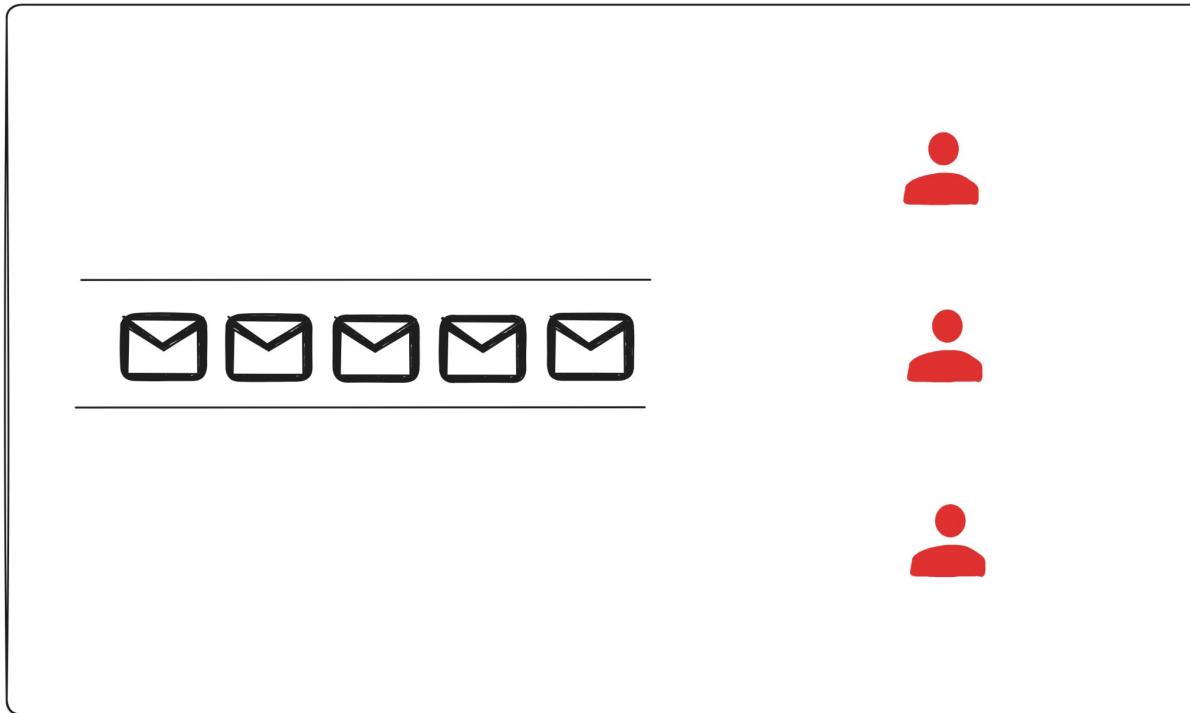
| Routing Customer Inquiries



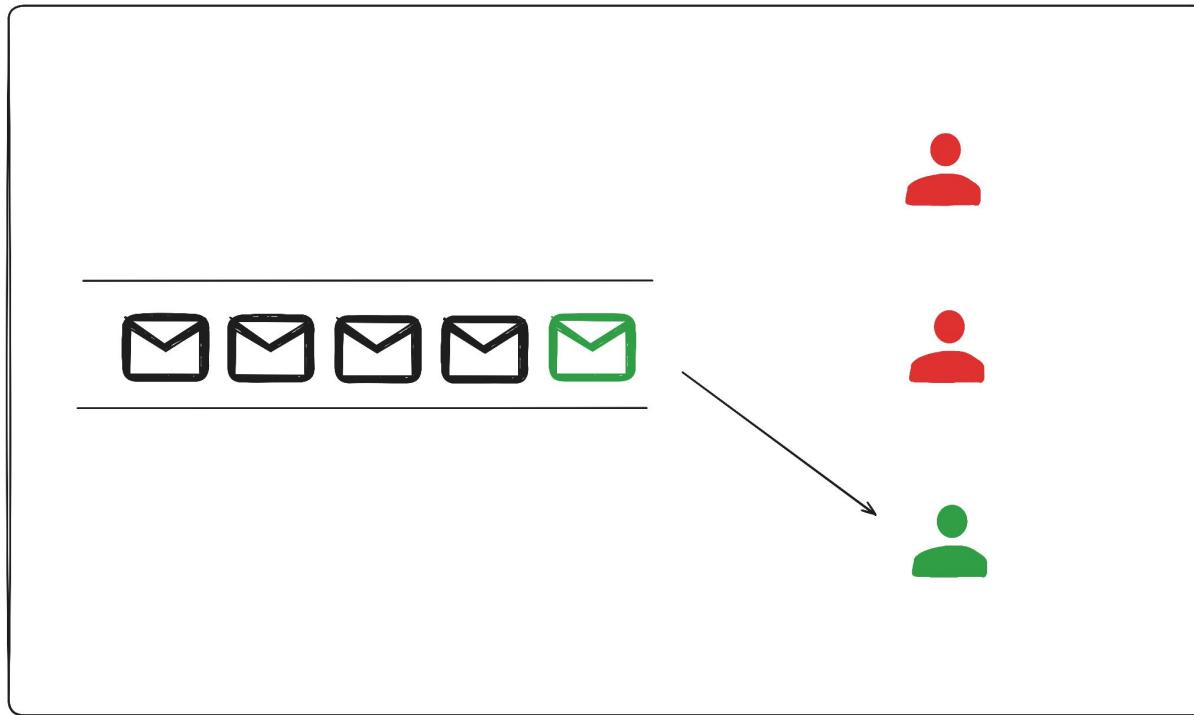
| Random Allocation



| Random Allocation



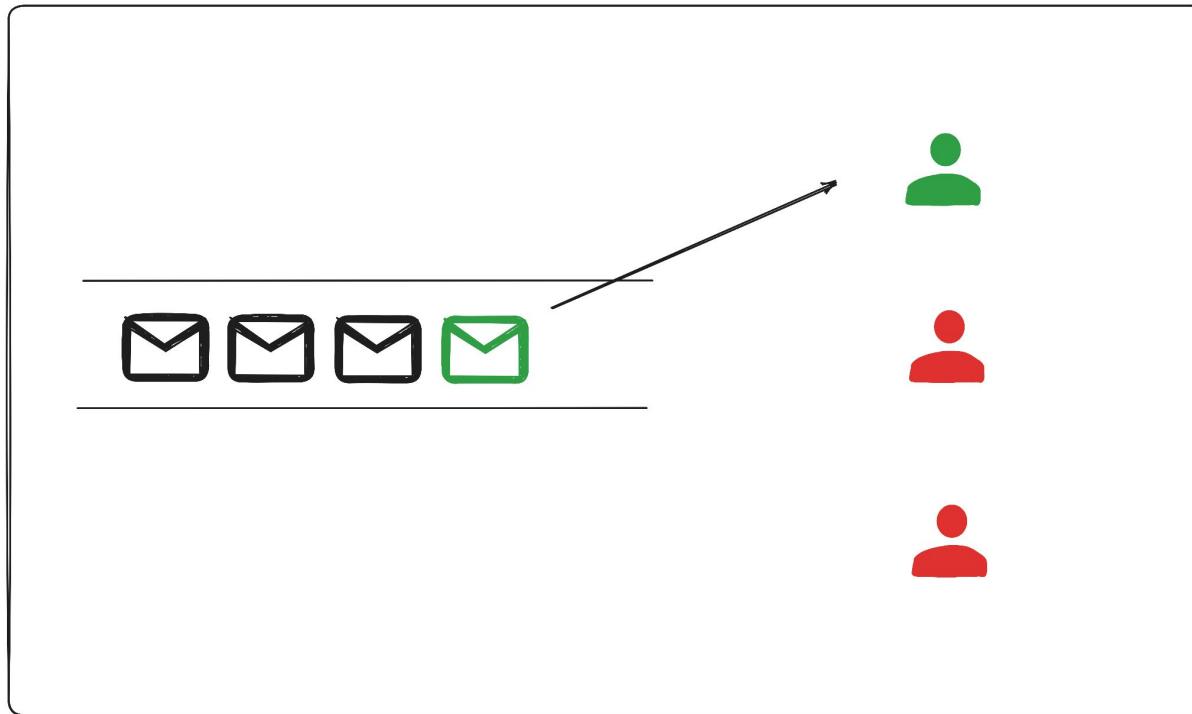
| Random Allocation



| Random Allocation



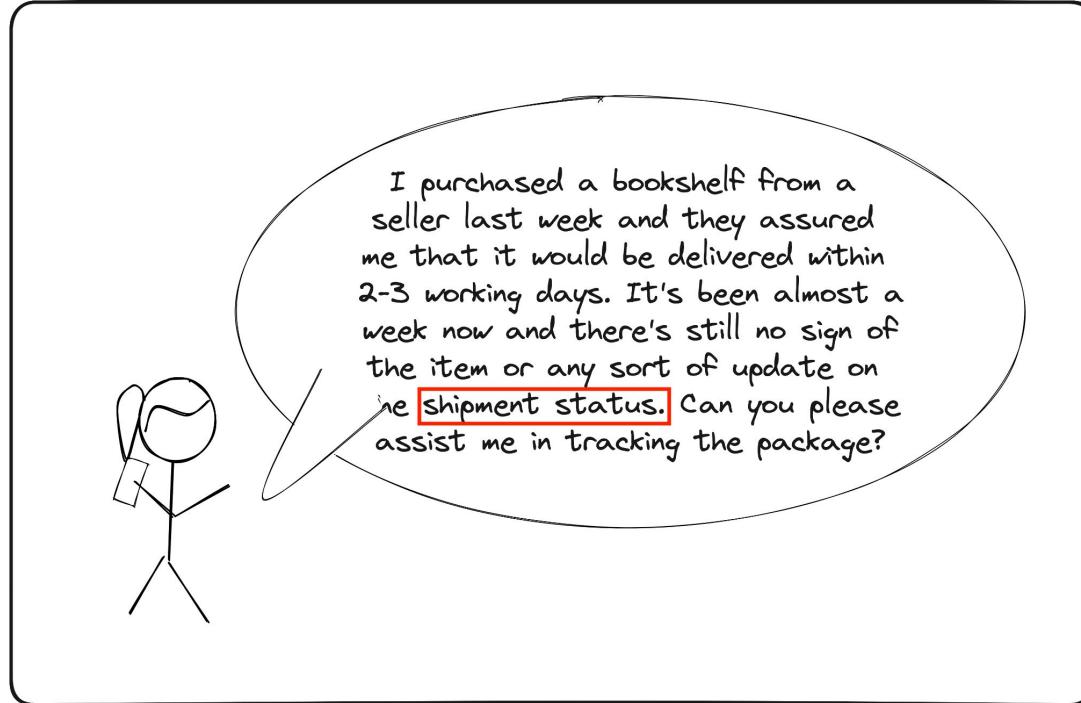
| Random Allocation



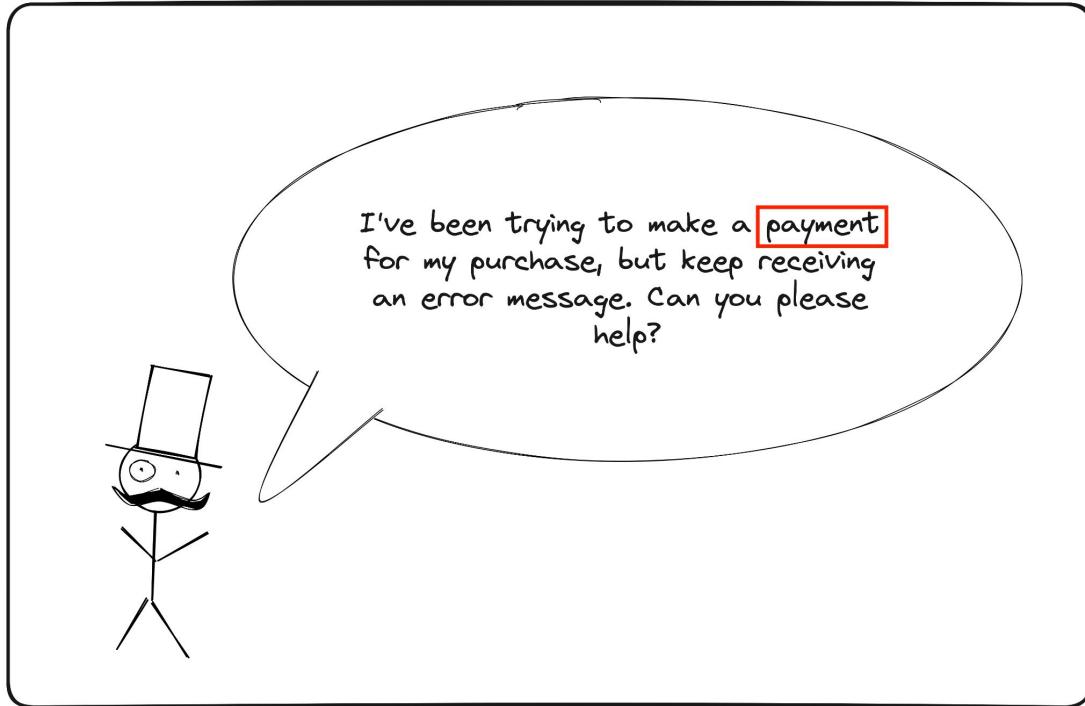
I Sample Inquiries



| Sample Inquiries



I Sample Inquiries

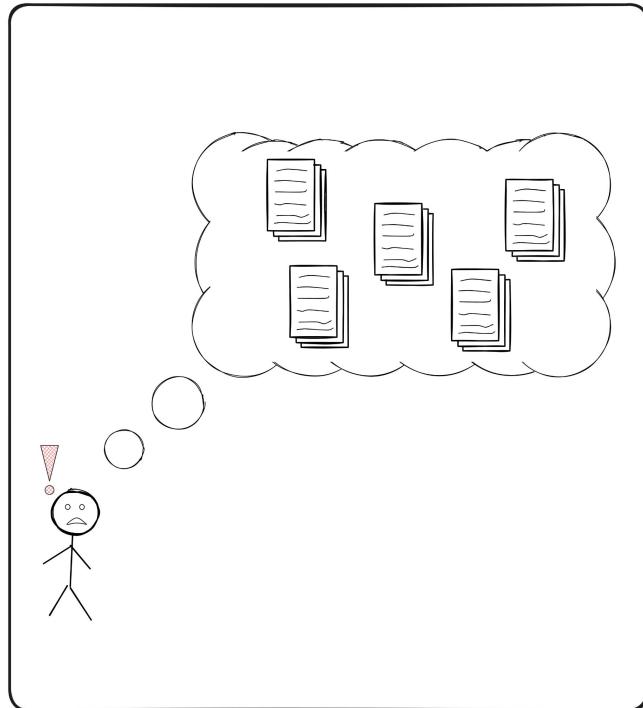


I Sample Inquiries

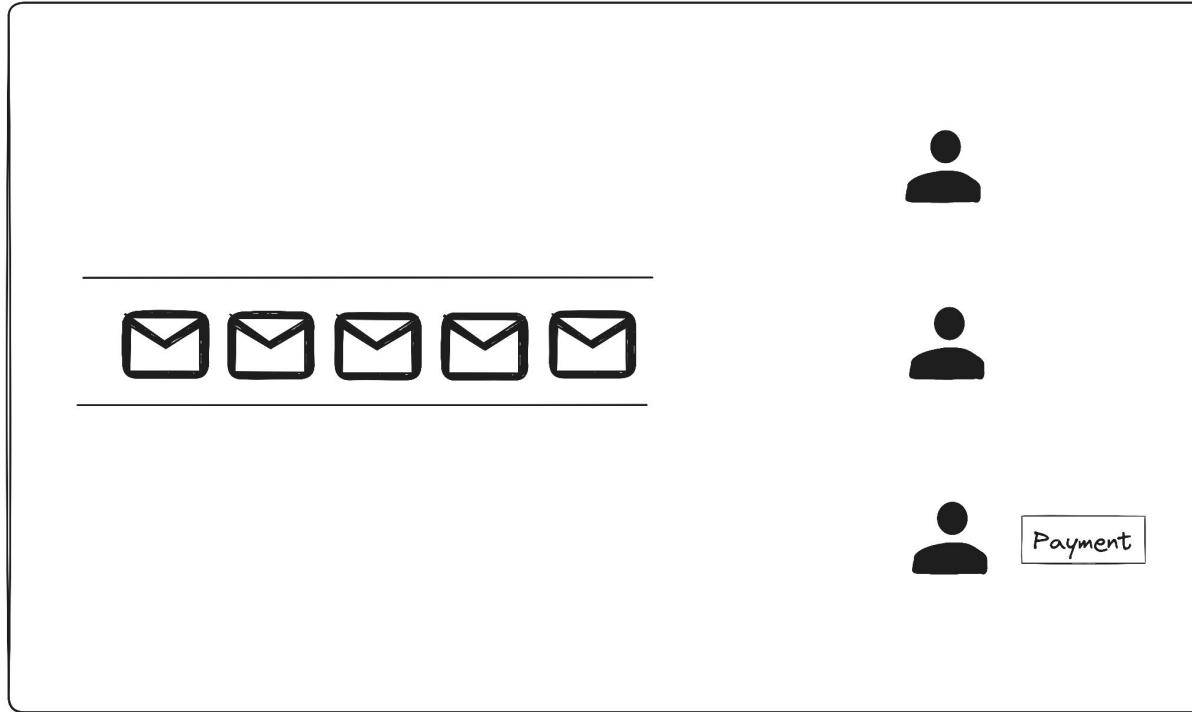


| Random Allocation Problems

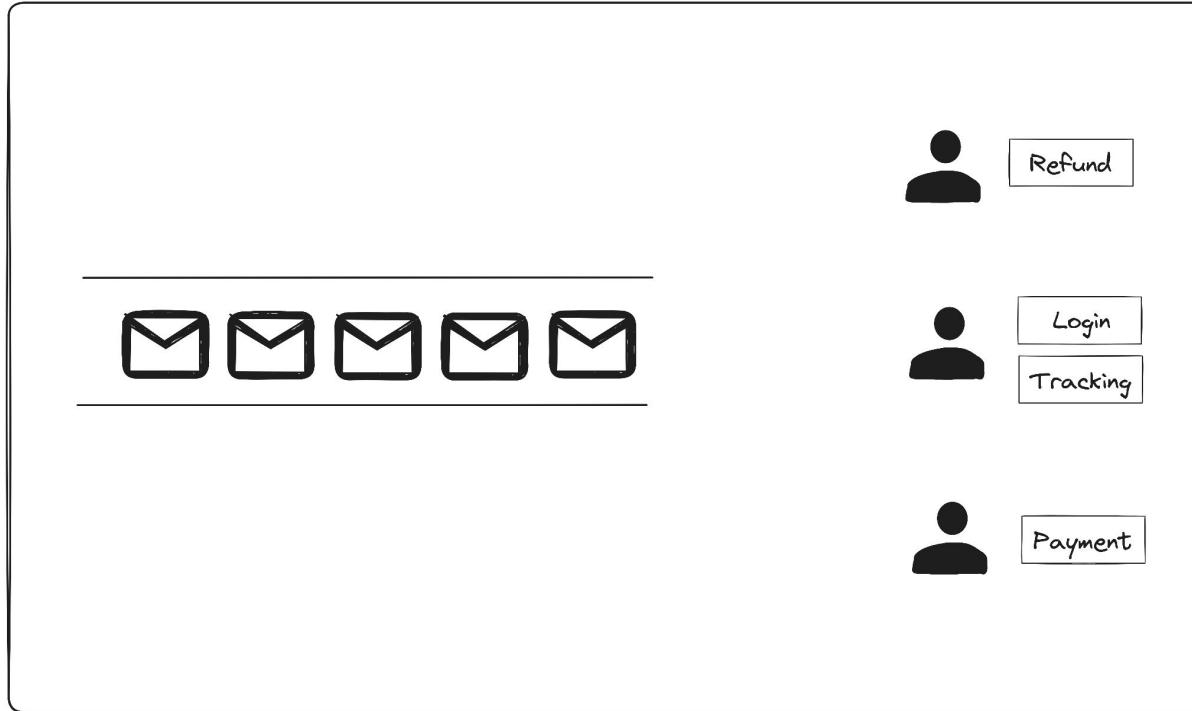
- Knowledge of handling all types of inquiries
- High onboarding cost for new agents
- Difficulty in changing guidelines for any type of inquiry



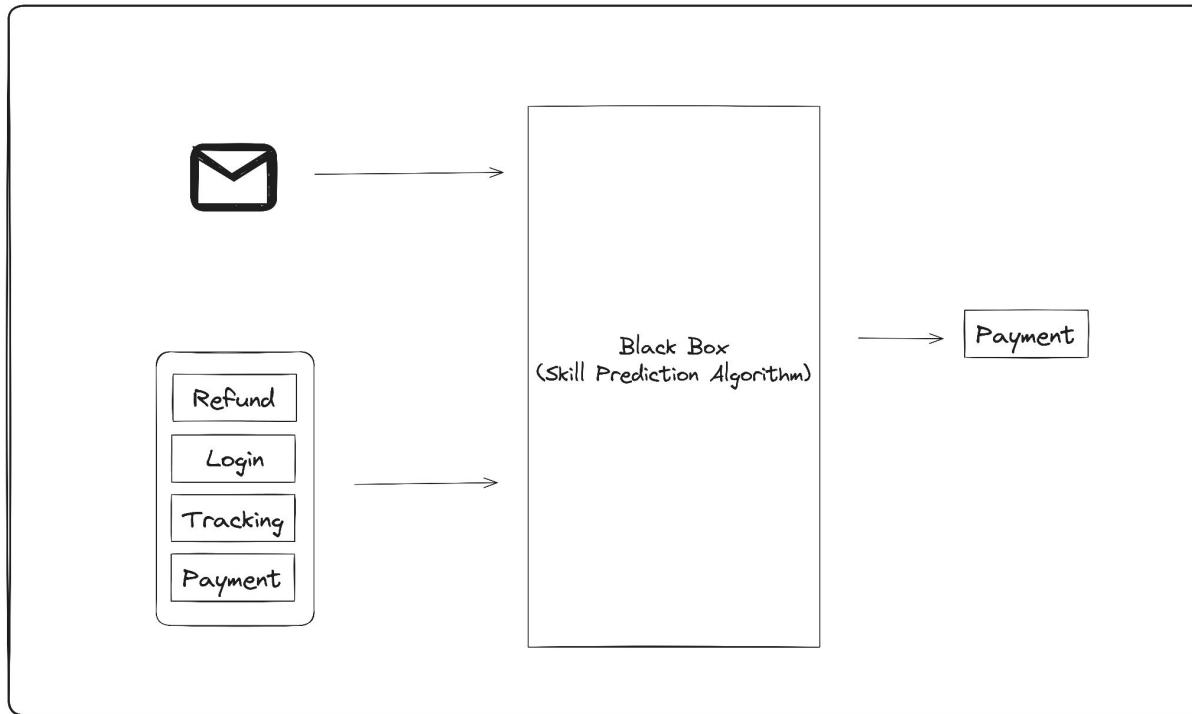
I Skill Based Routing



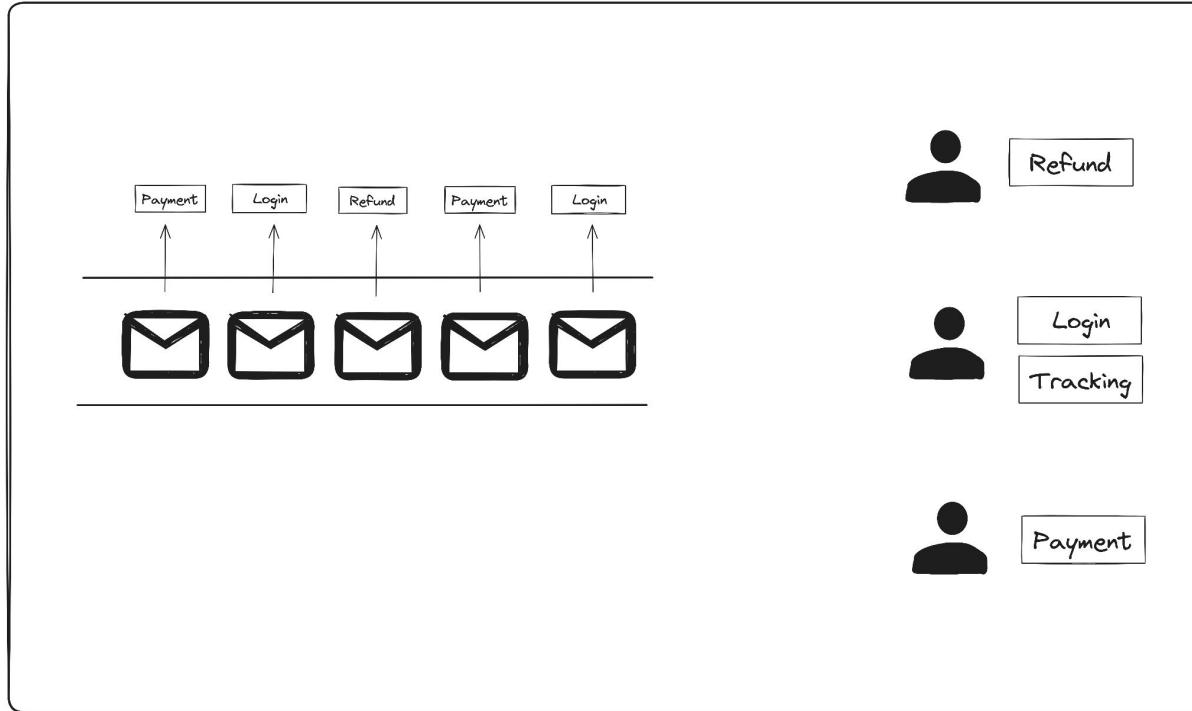
| Skill Based Routing



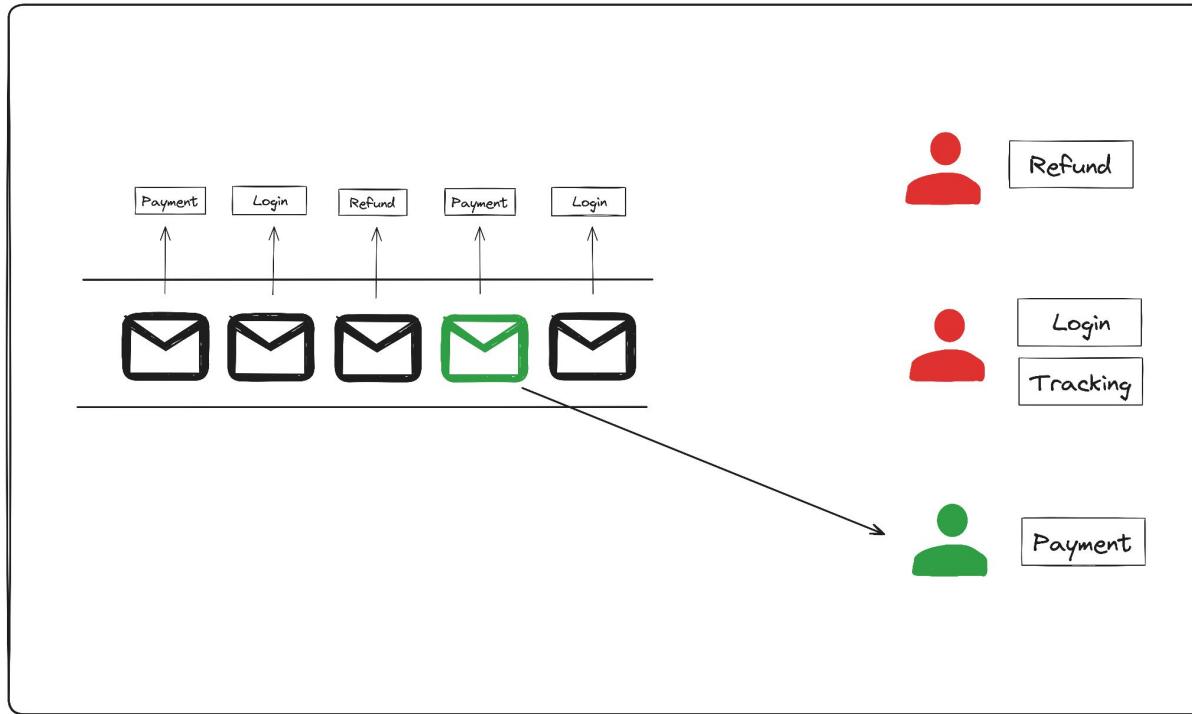
I Skill Based Routing



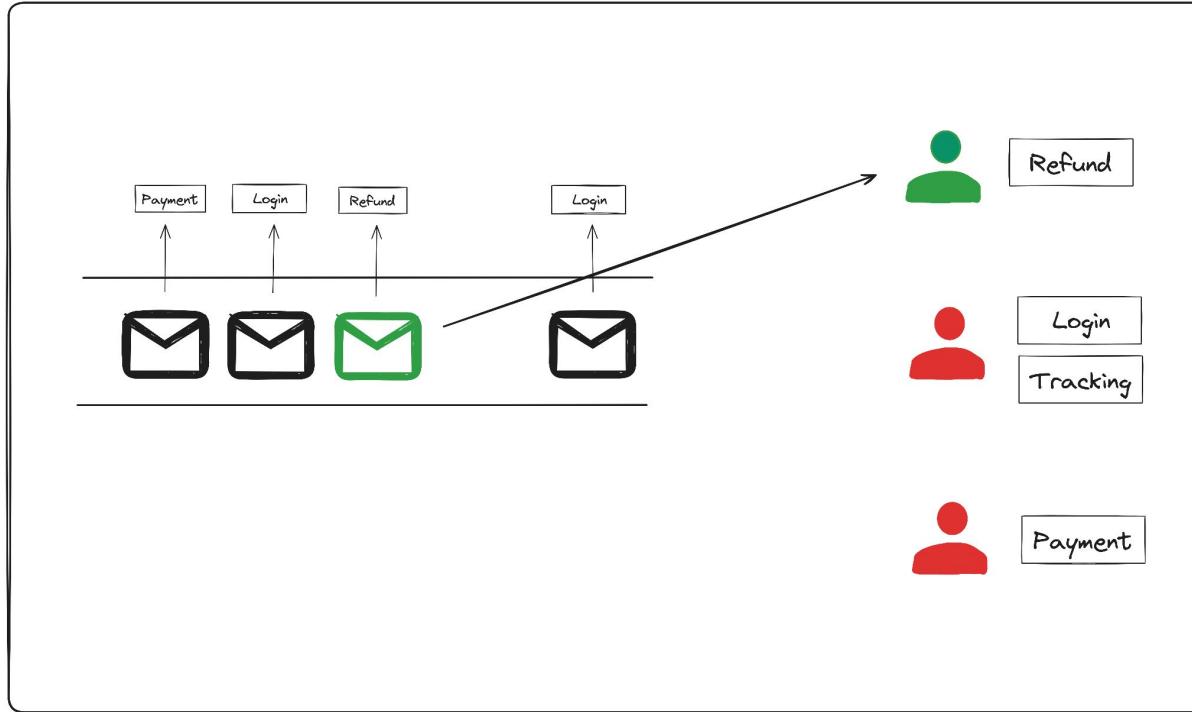
I Skill Based Routing



| Skill Based Routing



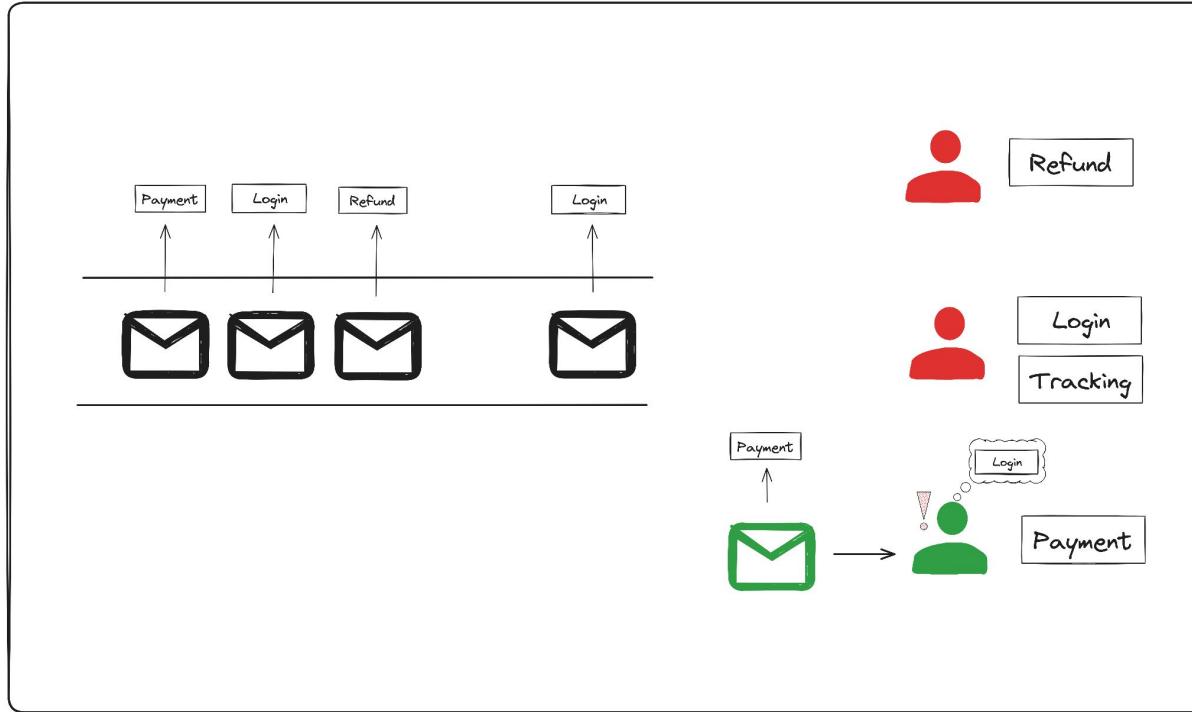
| Skill Based Routing



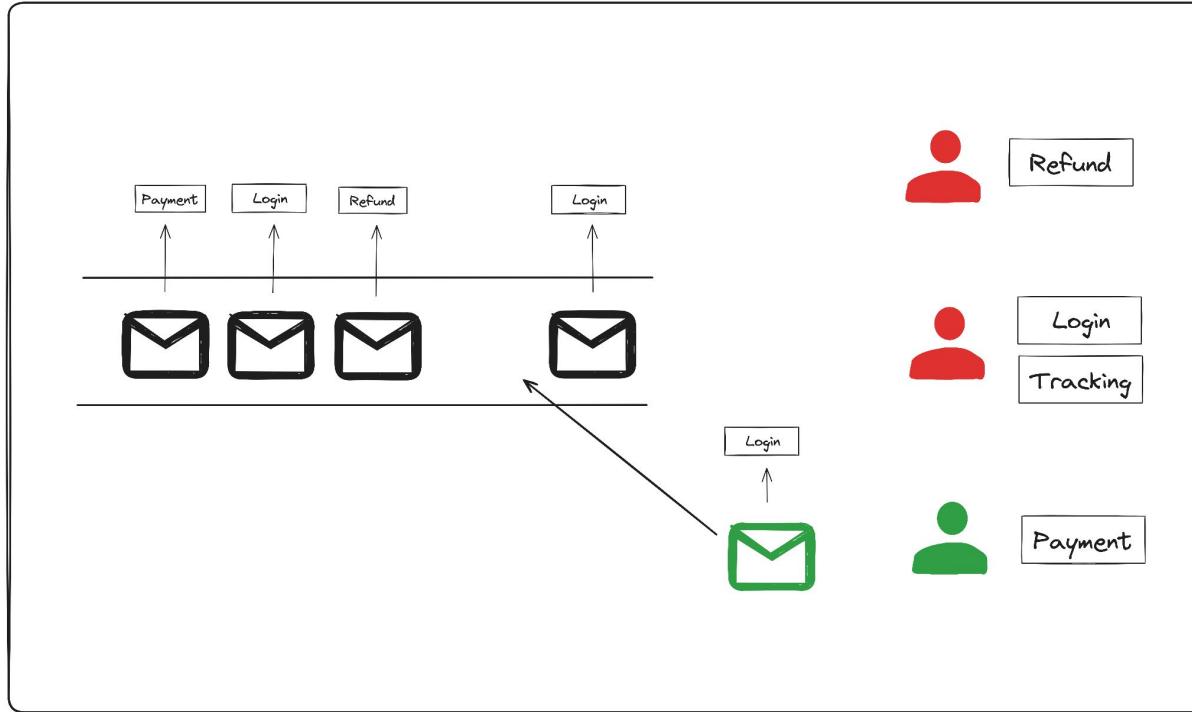
| Skill Based Routing

- Agents have one or more skills
- Assign one skill to each inquiry
- Routing algorithm
 - An agent becomes available
 - Find the first inquiry whose assigned skill is one of the skills of the agent
 - Route that inquiry to be handled by that available agent

| Transfers



| Transfers



| Business Problem

- Transfer: Manual change of skill assigned to a case by a CS agent
- Problem: High number of transfers
- Transfer Rate: Ratio of number of transfers and number of times inquiries sent to agents
- **Business problem: Reducing the Transfer Rate**

| Why reducing Transfer Rate is important?

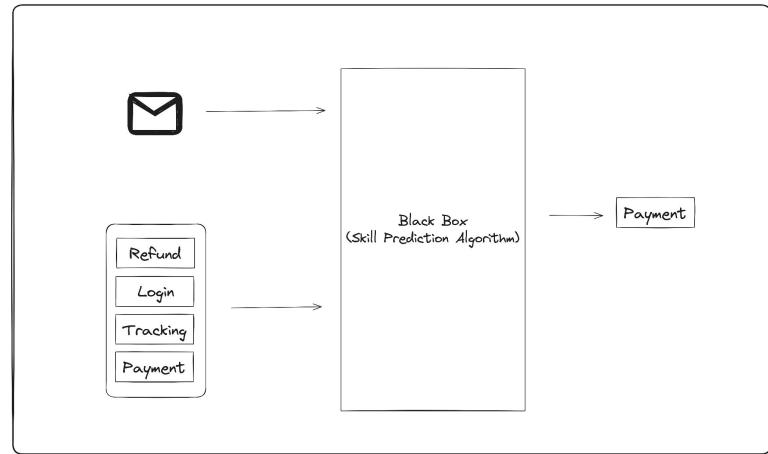
- Faster response
- Improved efficiency

02

Baseline solution

| How to reduce Transfer Rate?

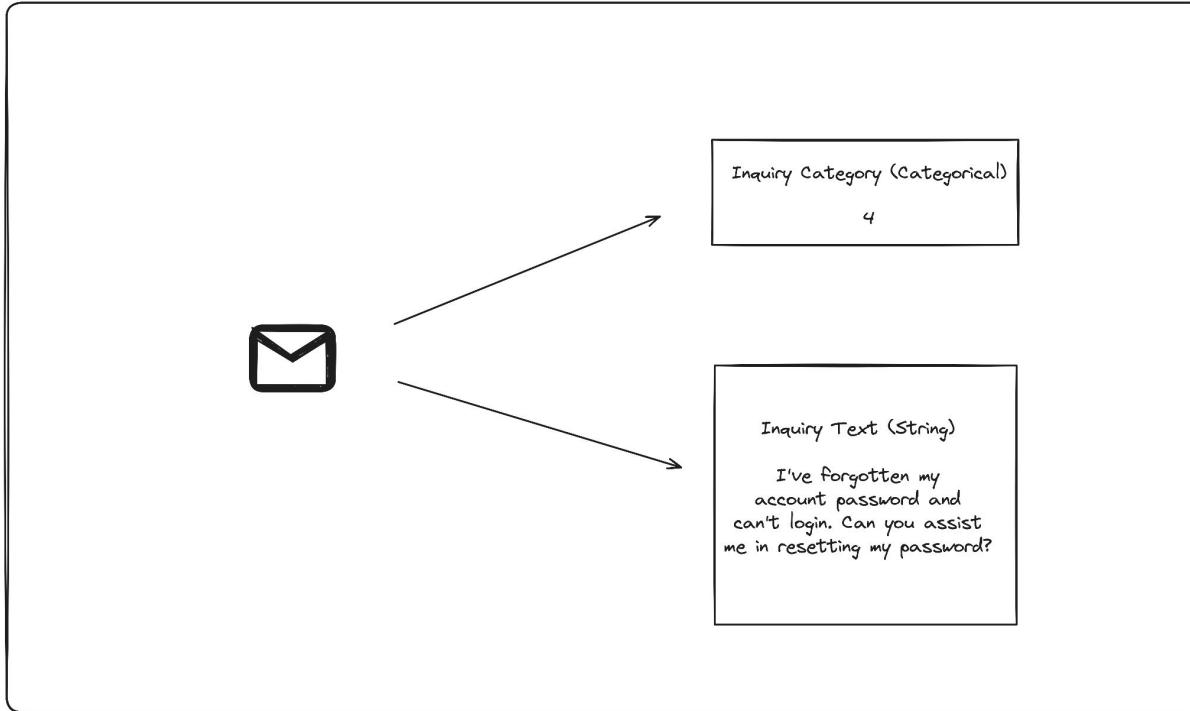
- If assigned skill to an inquiry is correct, no need for any transfer.
- Improvement in the accuracy of Skill Prediction Algorithm should lead to lesser number of transfers which will reduce transfer rate.



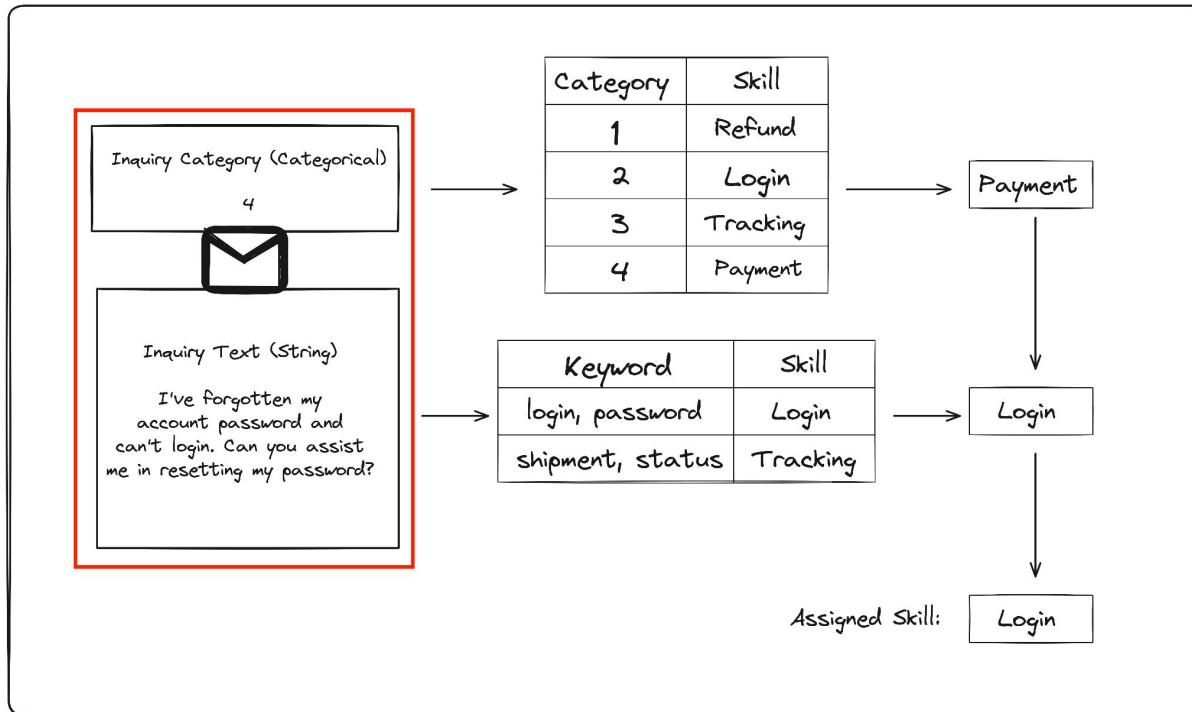
I Improving Skill Prediction Algorithm

- Target metric: Accuracy
- Baseline skill prediction algorithm
- Using NLP and ML for improvement

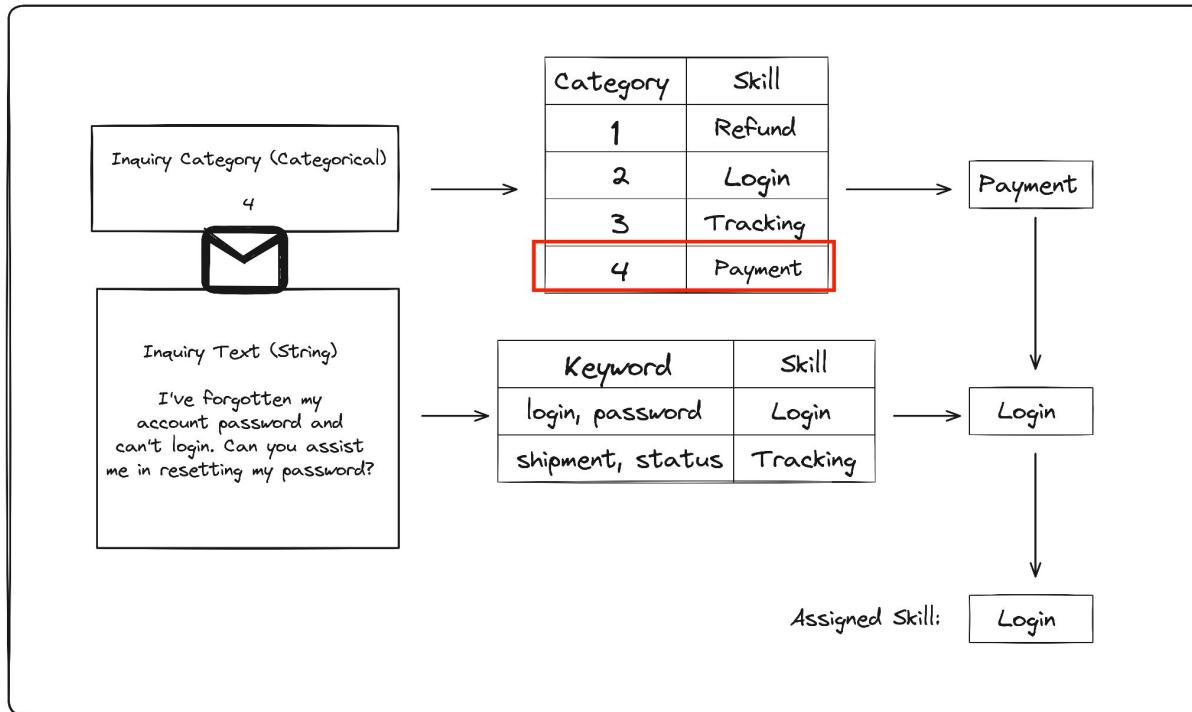
I Baseline Skill Prediction Algorithm



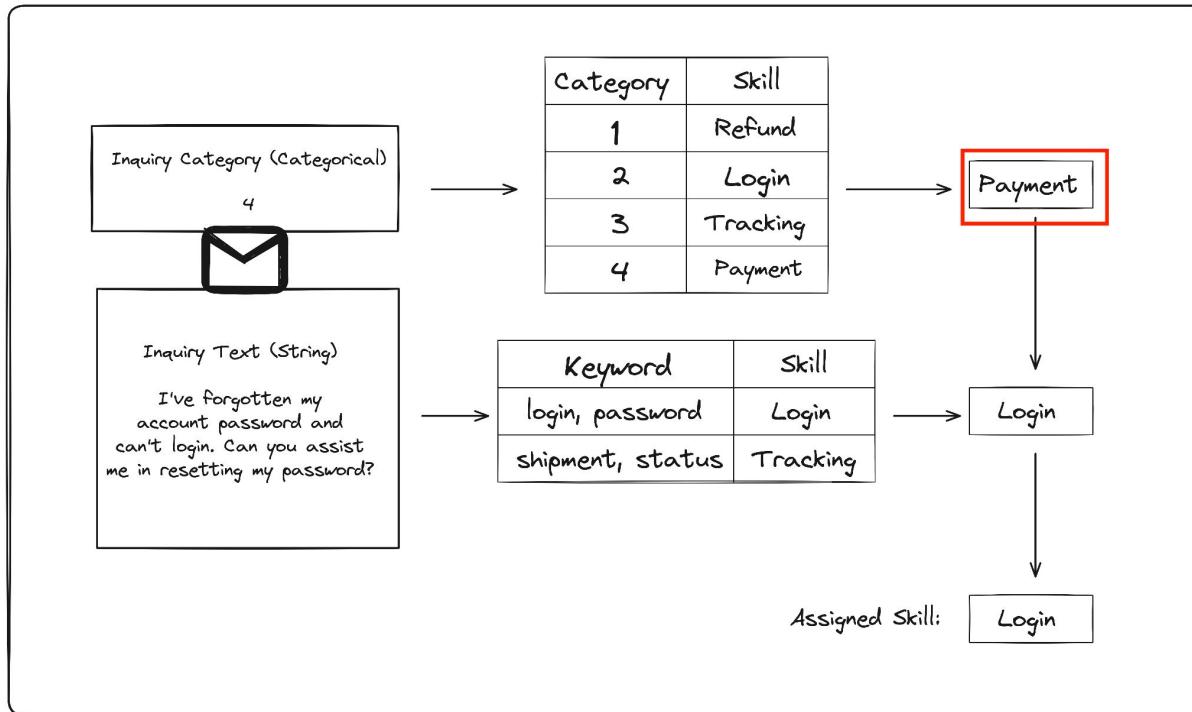
Baseline Skill Prediction Algorithm Example 1



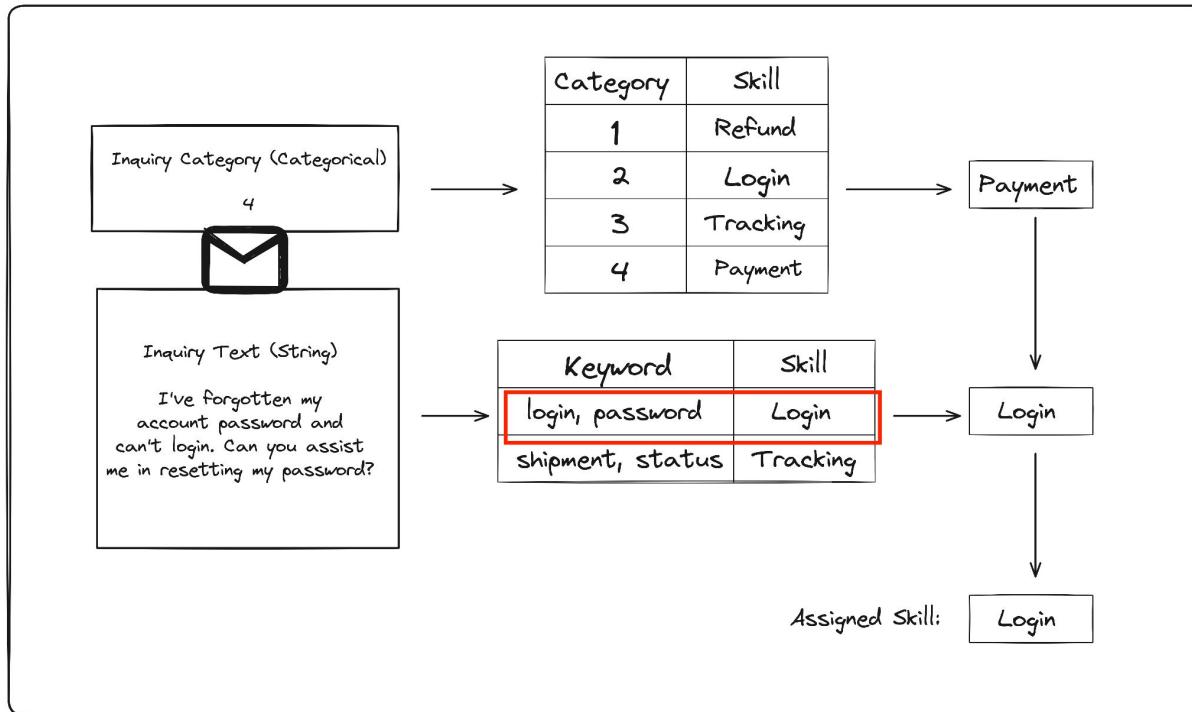
Baseline Skill Prediction Algorithm Example 1



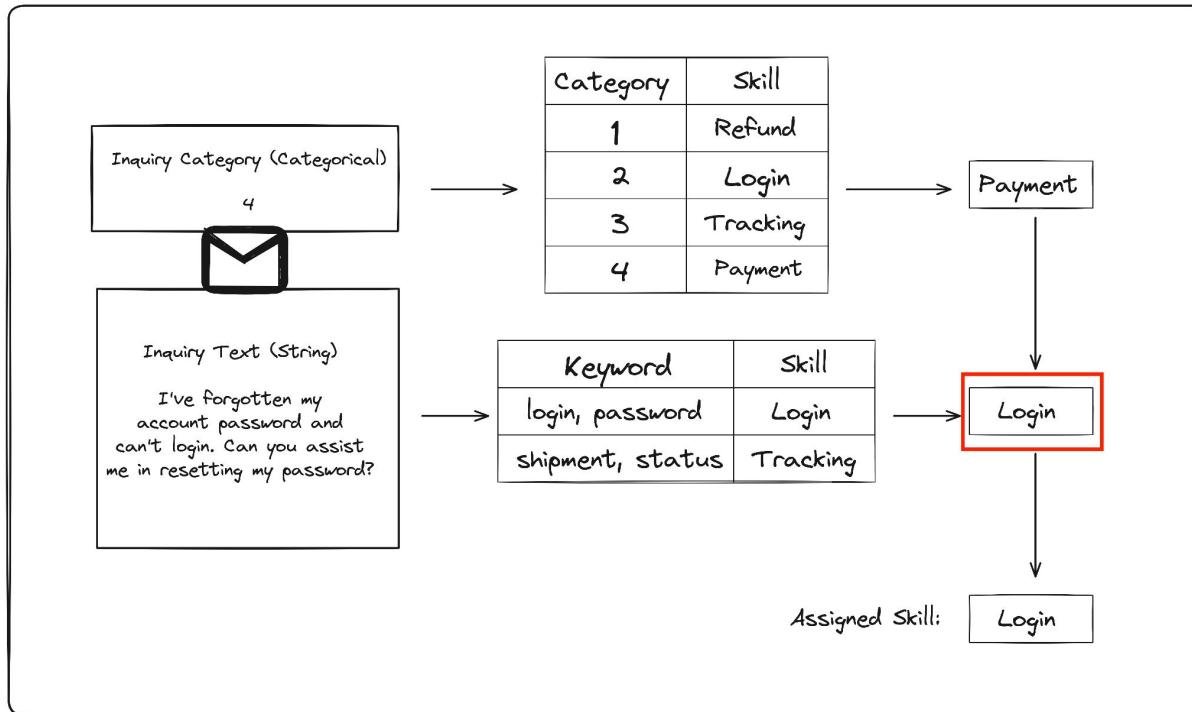
Baseline Skill Prediction Algorithm Example 1



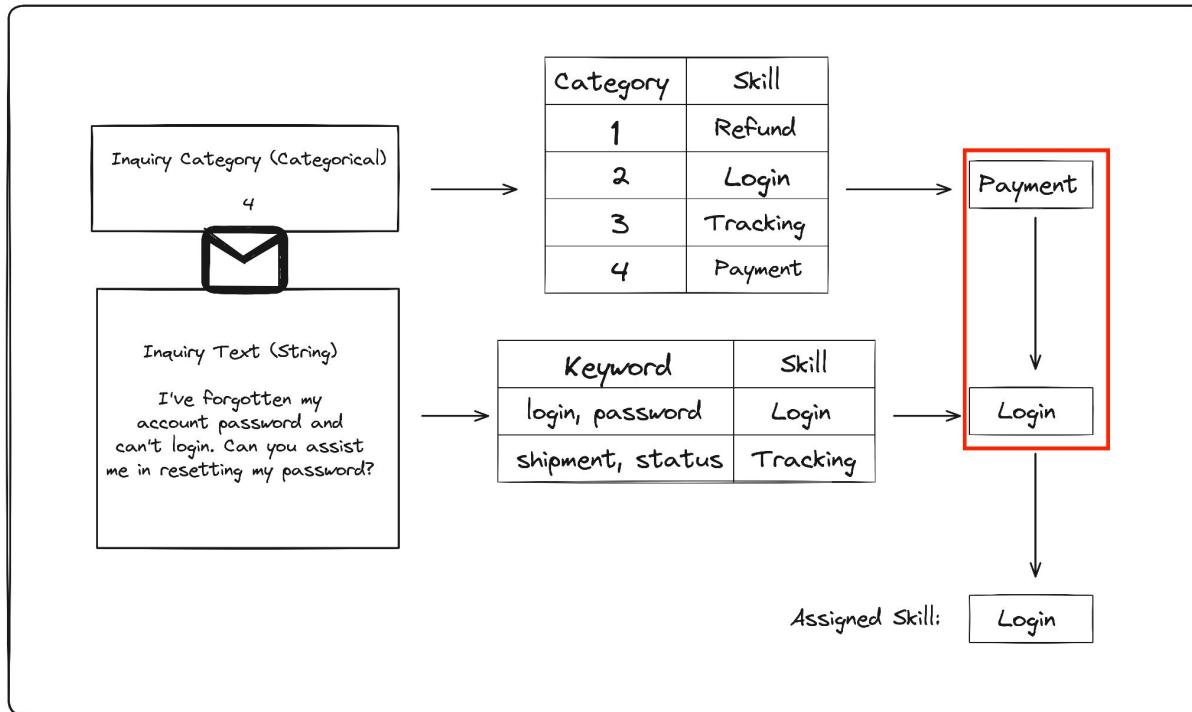
Baseline Skill Prediction Algorithm Example 1



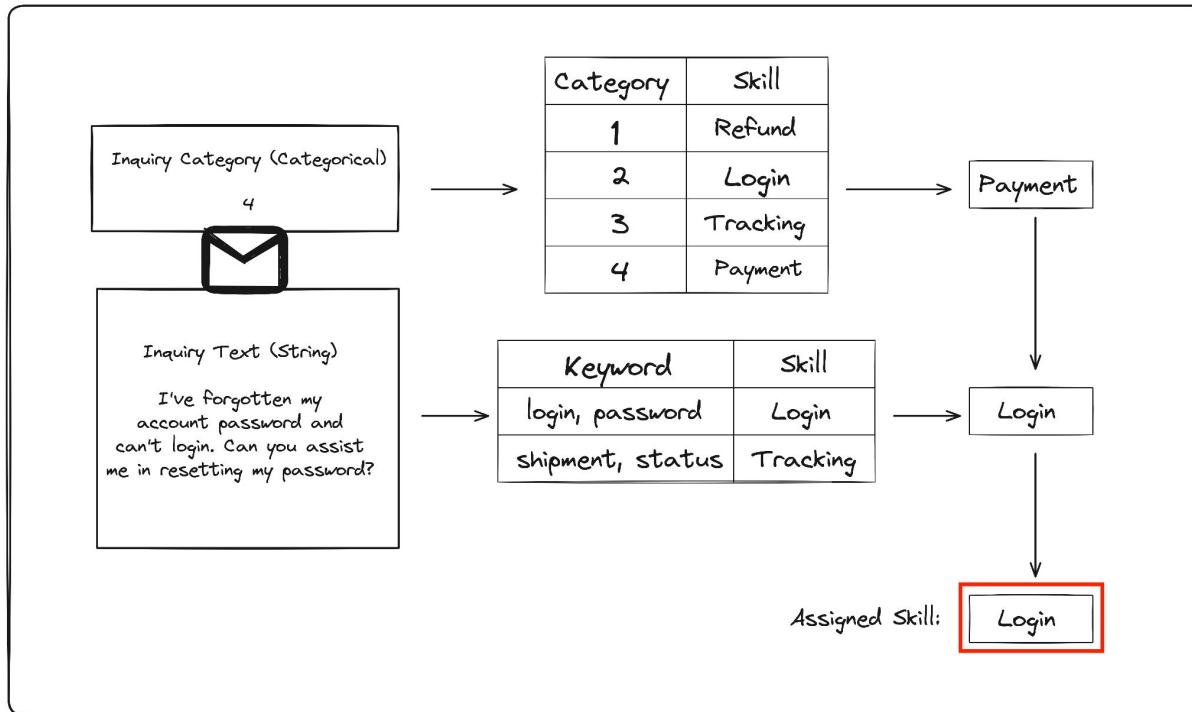
Baseline Skill Prediction Algorithm Example 1



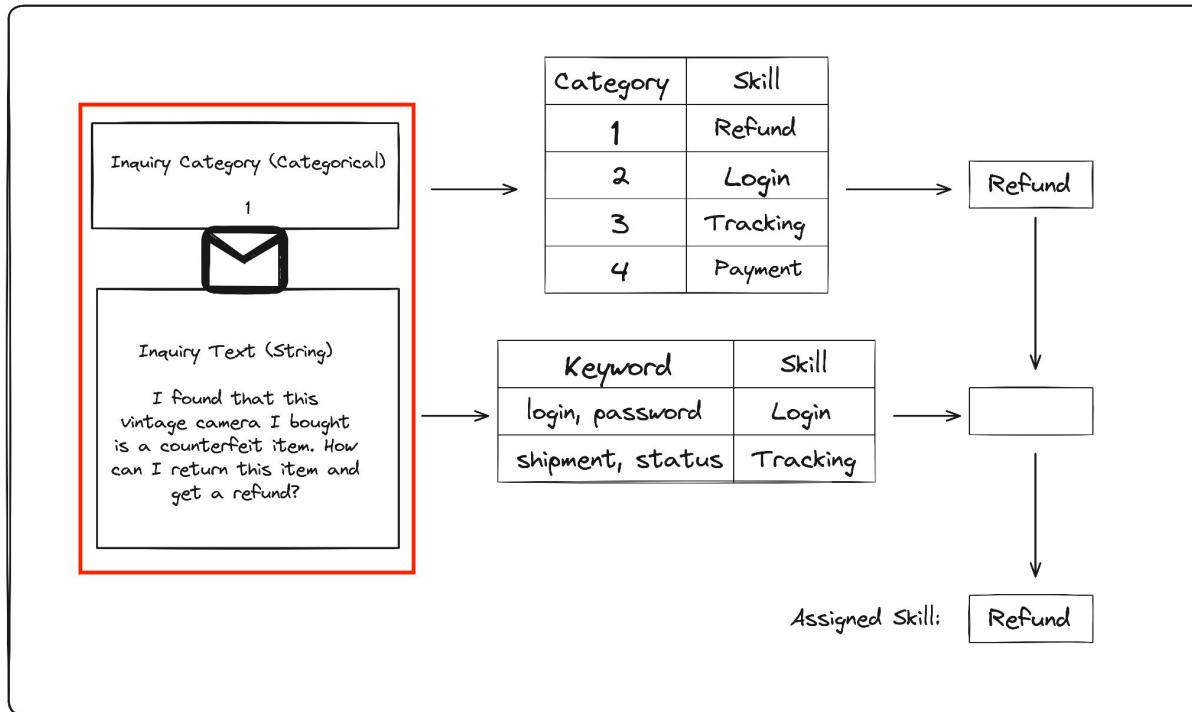
Baseline Skill Prediction Algorithm Example 1



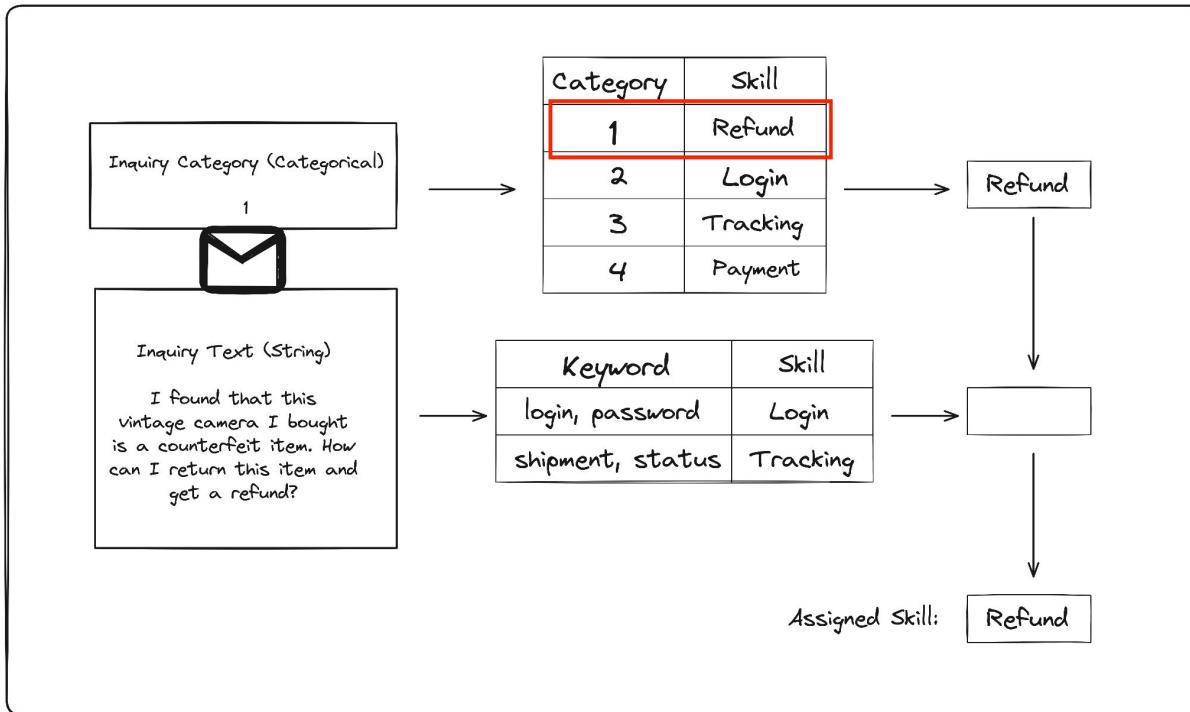
Baseline Skill Prediction Algorithm Example 1



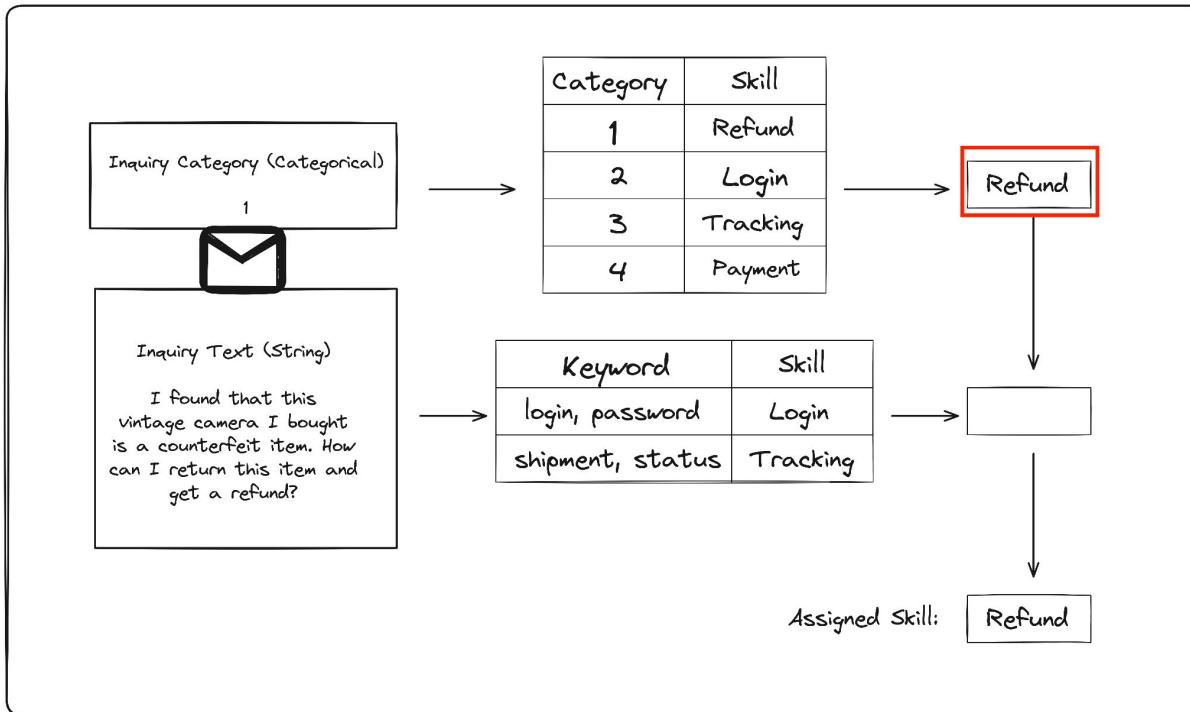
Baseline Skill Prediction Algorithm Example 2



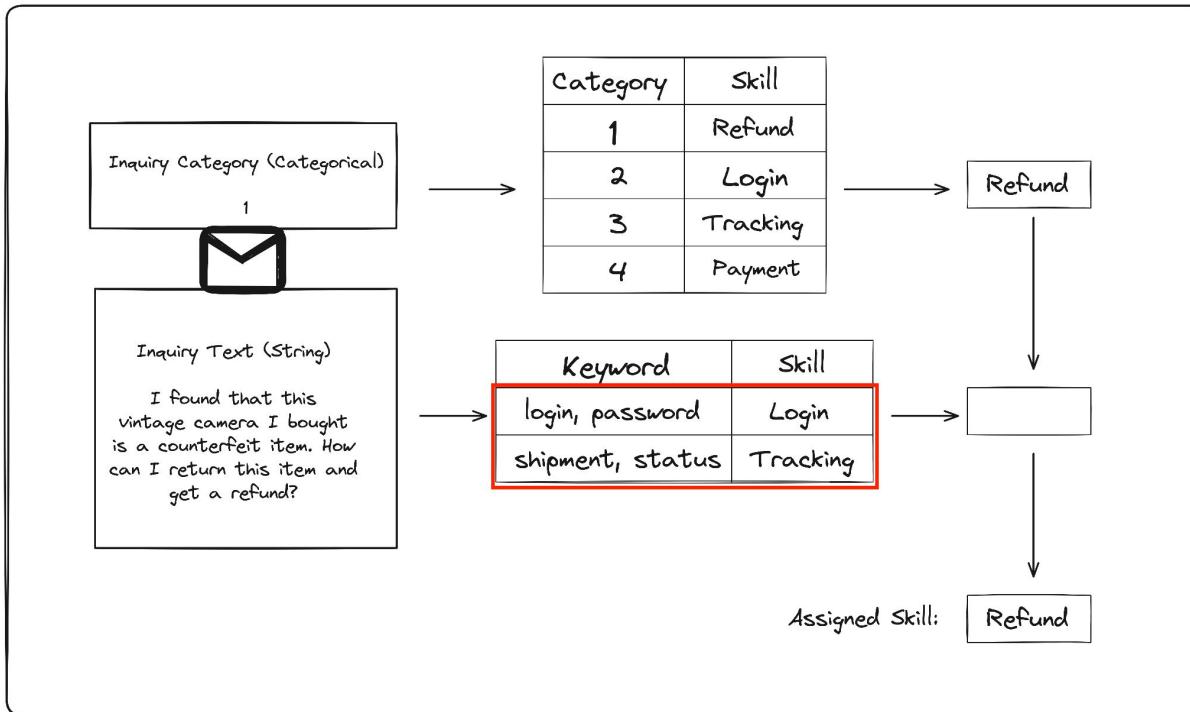
Baseline Skill Prediction Algorithm Example 2



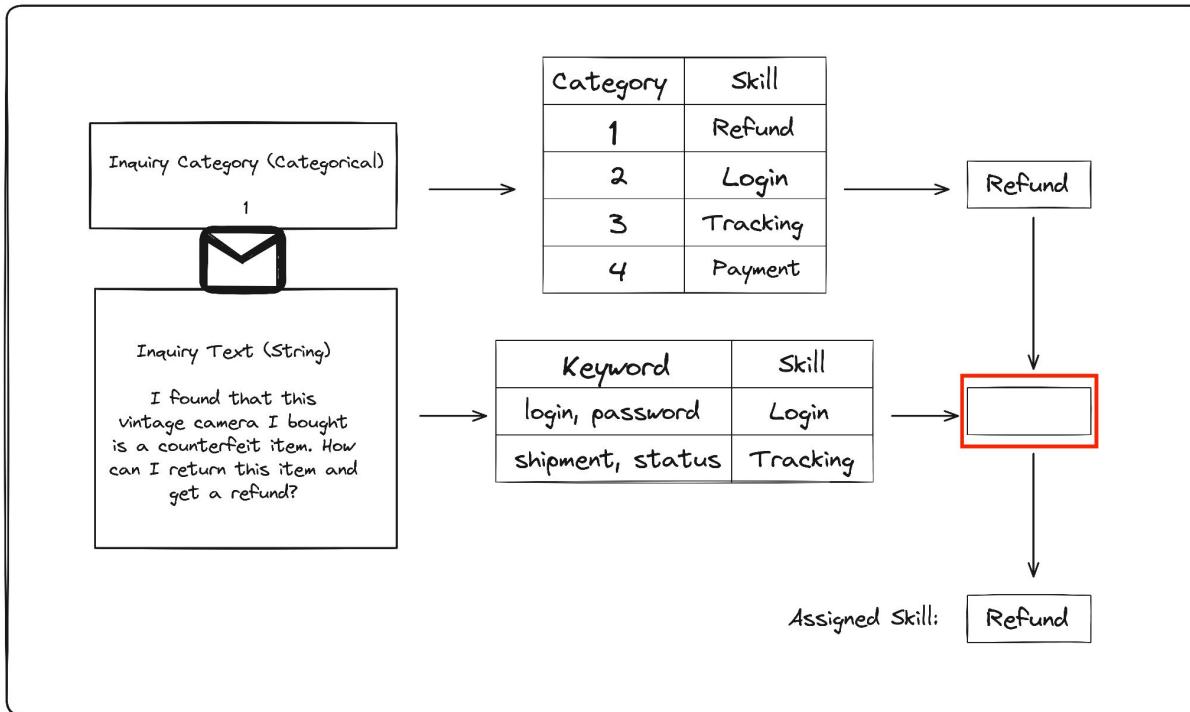
Baseline Skill Prediction Algorithm Example 2



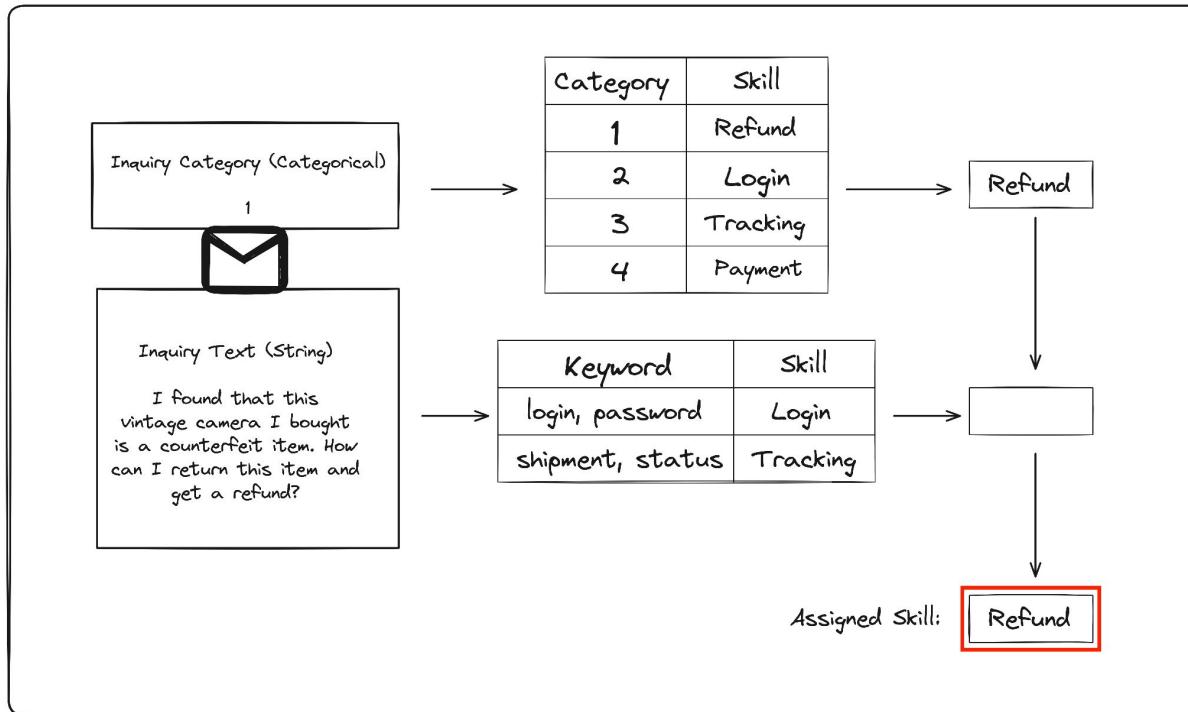
Baseline Skill Prediction Algorithm Example 2



Baseline Skill Prediction Algorithm Example 2



Baseline Skill Prediction Algorithm Example 2



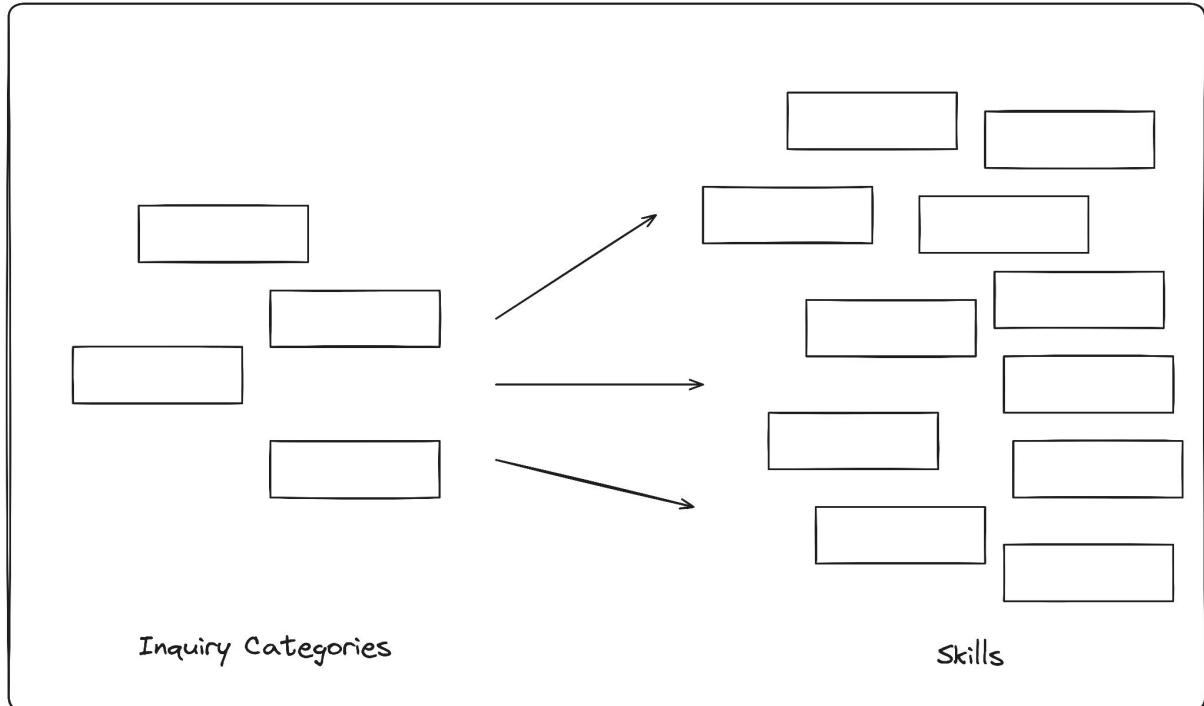
I Baseline Skill Prediction Algorithm

- All inquiry categories have associated skill
- Assign skill to the inquiry based on the inquiry category
- Any of the keyword rules match with the inquiry text?
- Update the skill of the inquiry based on the keyword rule

I Problems with the Baseline Algorithm

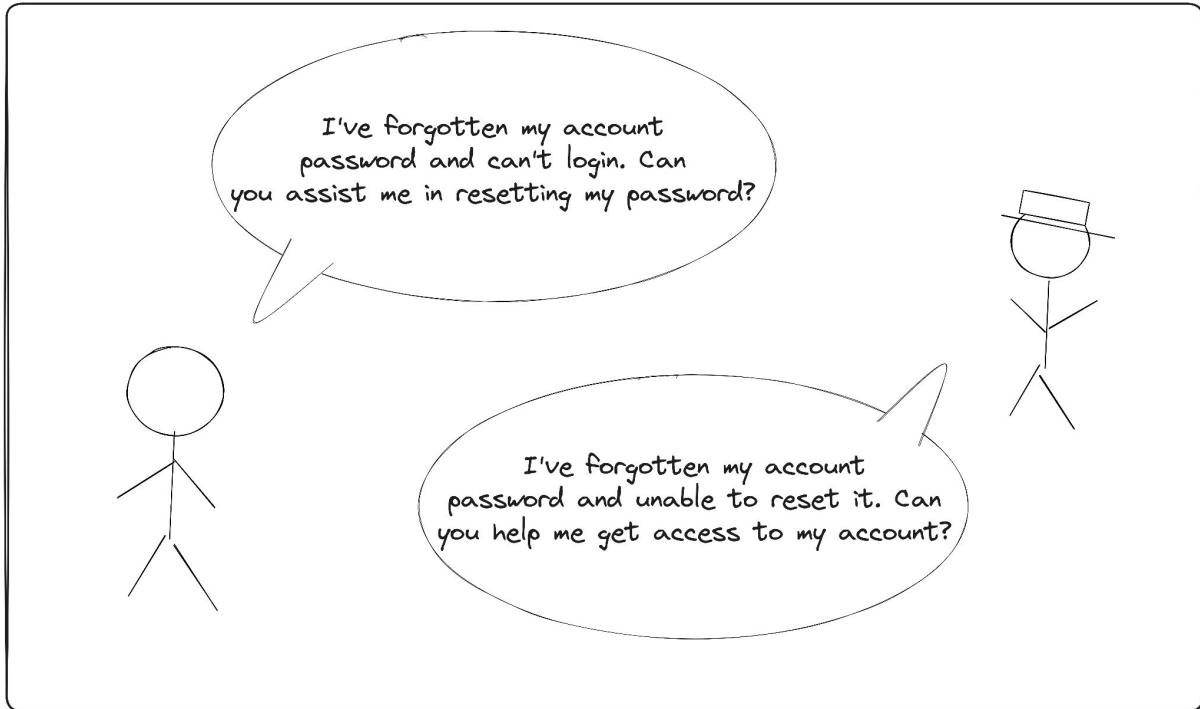
Few inquiry
categories

Lots of skills



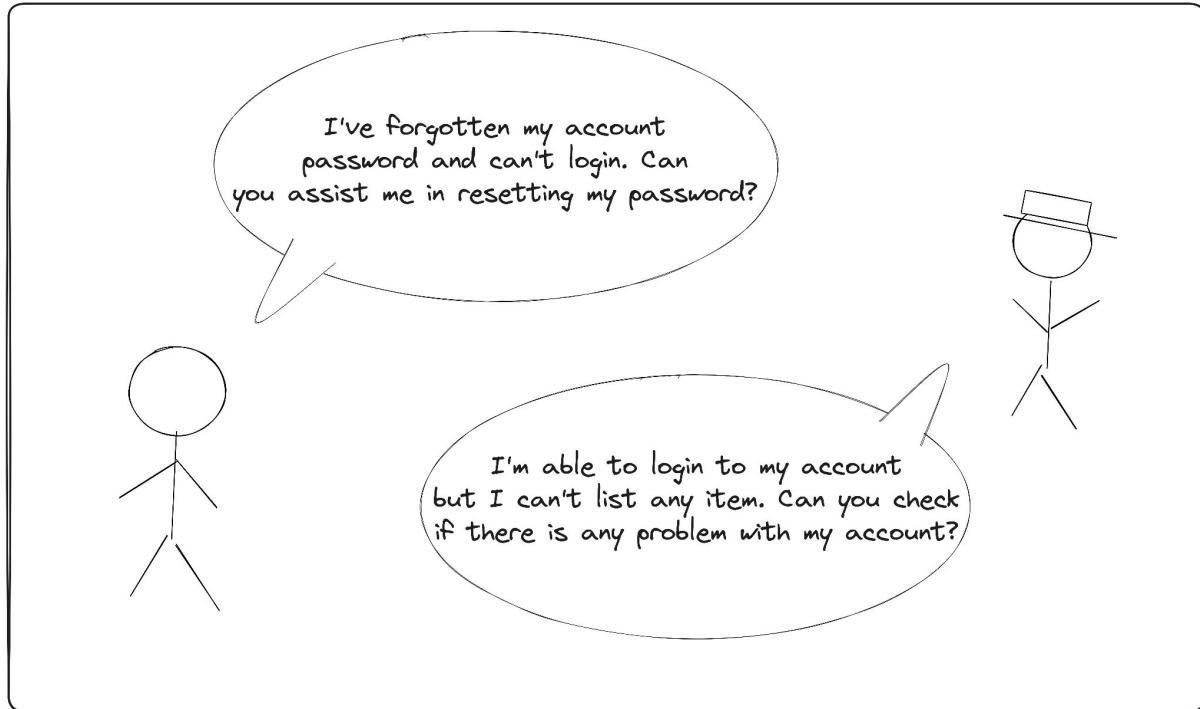
I Problems with the Baseline Algorithm

Handling
synonyms



I Problems with the Baseline Algorithm

Understanding context

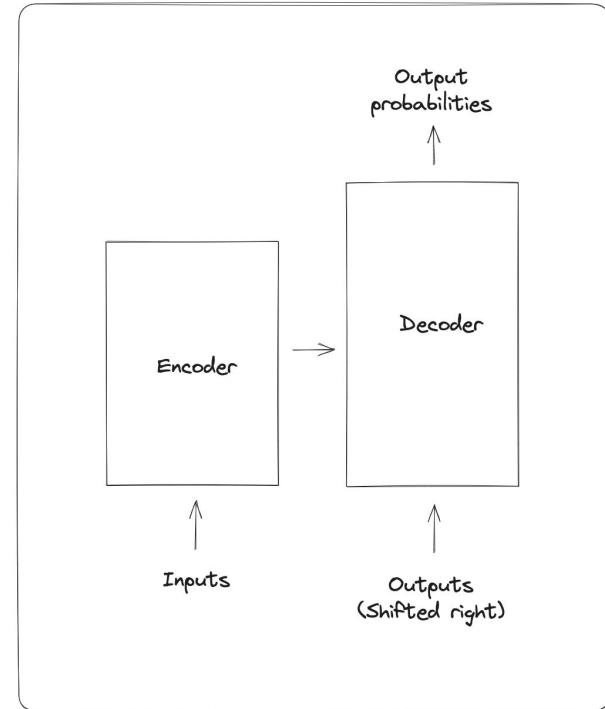


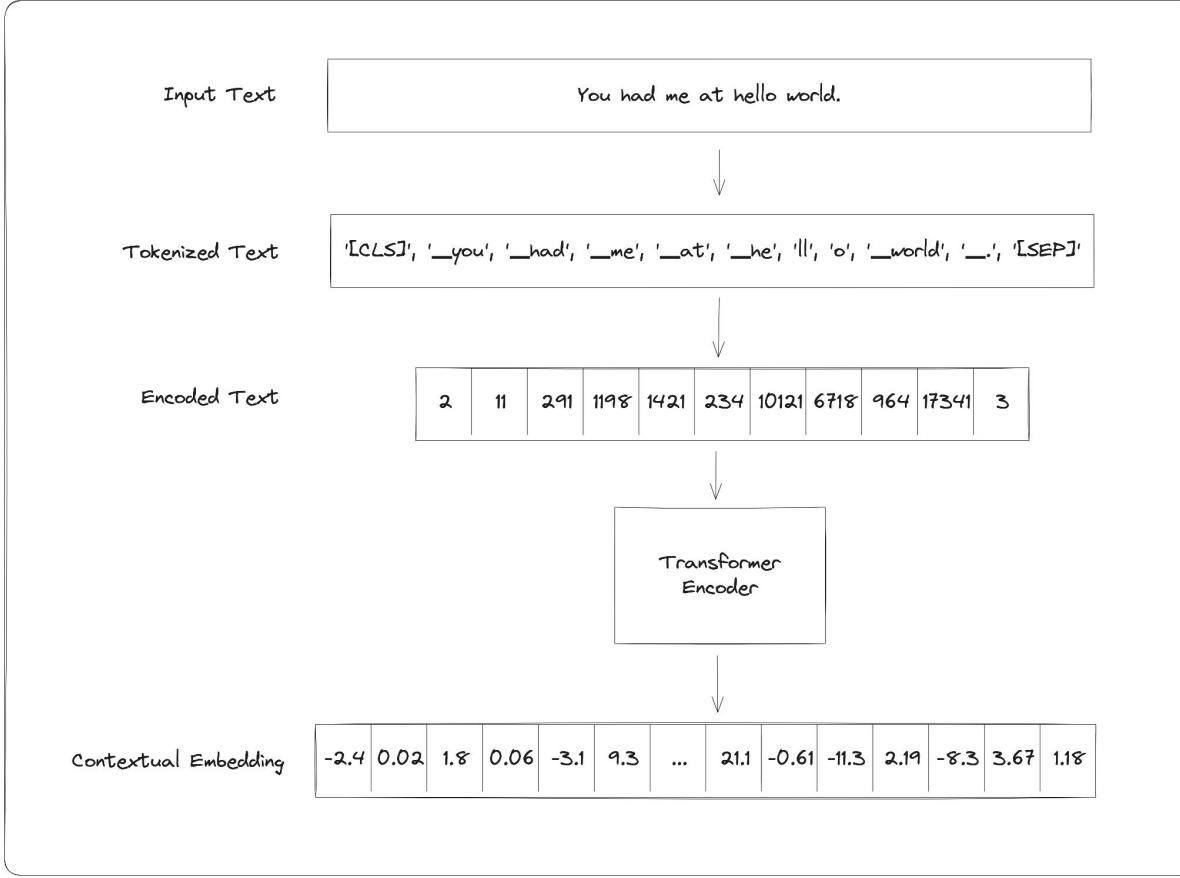
03

Developing the ML solution

Transformers Overview

- Introduced in 2017
- State-of-the-art architecture for NLP tasks
- Pre-trained on massive amount of data
- Contextual understanding





I Pre-trained Models

Model Name	Release Year	Link
Tohoku BERT base Japanese V3	2023	https://huggingface.co/cl-tohoku/bert-base-japanese-v3
LINE DistilBERT Japanese	2023	https://github.com/line/LINE-DistilBERT-Japanese
Rinna Japanese RoBERTa Base	2021	https://huggingface.co/rinna/japanese-roberta-base
Bandai Namco DistilBERT-base-jp	2020	https://github.com/BandaiNamcoResearchInc/DistilBERT-base-jp

```
from transformers import AutoTokenizer, AutoModel

sentence = "パスワードを忘れてしまいました。リセットの手立てを教えてください。"

tokenizer = AutoTokenizer.from_pretrained(
    "line-corporation/line-distilbert-base-japanese"
)
tokenizer_output = tokenizer(sentence, return_tensors="pt")

model = AutoModel.from_pretrained(
    "line-corporation/line-distilbert-base-japanese"
)
last_hidden_state = model(
    input_ids=tokenizer_output["input_ids"],
    attention_mask=tokenizer_output["attention_mask"],
)[0]
contextual_embedding = last_hidden_state[:, 0]

print(contextual_embedding.shape)
# torch.Size([1, 768])
```

```
from transformers import AutoTokenizer, AutoModel

sentence = "パスワードを忘れてしまいました。リセットの手立てを教えてください。"

tokenizer = AutoTokenizer.from_pretrained(
    "line-corporation/line-distilbert-base-japanese"
)
tokenizer_output = tokenizer(sentence, return_tensors="pt")

model = AutoModel.from_pretrained(
    "line-corporation/line-distilbert-base-japanese"
)
last_hidden_state = model(
    input_ids=tokenizer_output["input_ids"],
    attention_mask=tokenizer_output["attention_mask"],
)[0]
contextual_embedding = last_hidden_state[:, 0]

print(contextual_embedding.shape)
# torch.Size([1, 768])
```

```
from transformers import AutoTokenizer, AutoModel

sentence = "パスワードを忘れてしまいました。リセットの手立てを教えてください。"

tokenizer = AutoTokenizer.from_pretrained(
    "line-corporation/line-distilbert-base-japanese"
)
tokenizer_output = tokenizer(sentence, return_tensors="pt")

model = AutoModel.from_pretrained(
    "line-corporation/line-distilbert-base-japanese"
)
last_hidden_state = model(
    input_ids=tokenizer_output["input_ids"],
    attention_mask=tokenizer_output["attention_mask"],
)[0]
contextual_embedding = last_hidden_state[:, 0]

print(contextual_embedding.shape)
# torch.Size([1, 768])
```

```
from transformers import AutoTokenizer, AutoModel

sentence = "パスワードを忘れてしまいました。リセットの手立てを教えてください。"

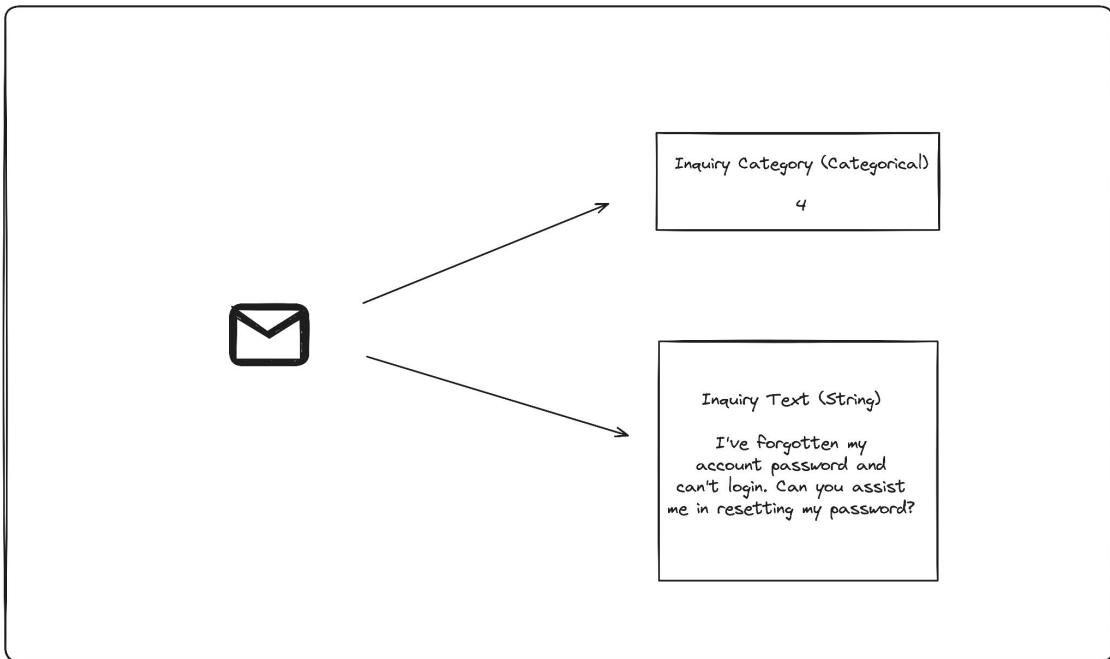
tokenizer = AutoTokenizer.from_pretrained(
    "line-corporation/line-distilbert-base-japanese"
)
tokenizer_output = tokenizer(sentence, return_tensors="pt")

model = AutoModel.from_pretrained(
    "line-corporation/line-distilbert-base-japanese"
)
last_hidden_state = model(
    input_ids=tokenizer_output["input_ids"],
    attention_mask=tokenizer_output["attention_mask"],
)[0]
contextual_embedding = last_hidden_state[:, 0]

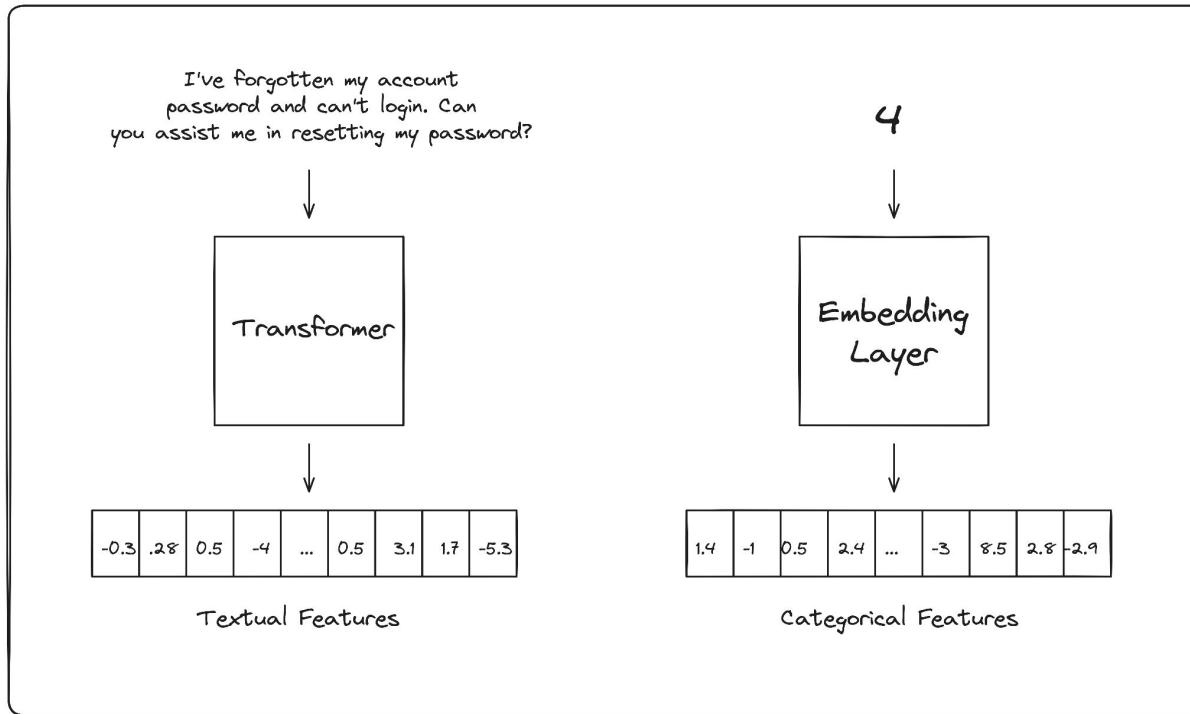
print(contextual_embedding.shape)
# torch.Size([1, 768])
```

I ML Model for Skill Prediction

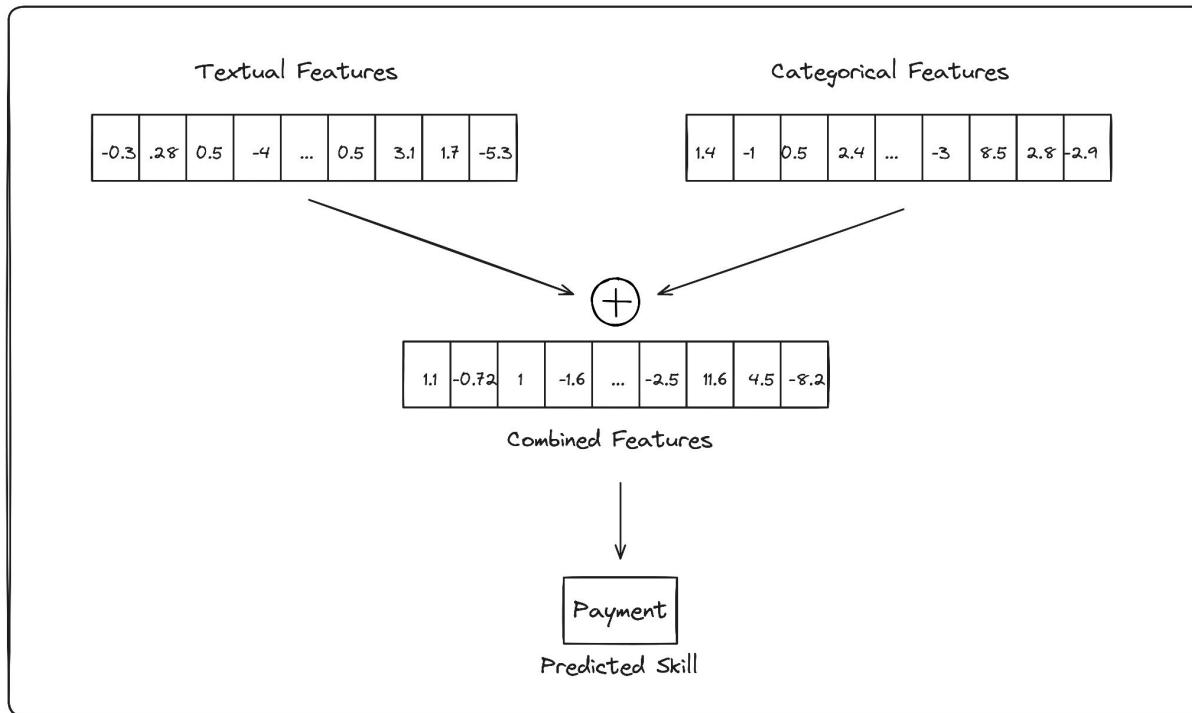
- Inputs:
 - Inquiry category
 - Inquiry text
- Outputs:
 - Skill for the inquiry



ML Model for Skill Prediction



ML Model for Skill Prediction



```
import torch
from transformers import DistilBertModel


class DistilBertWithCategorial(torch.nn.Module):
    def __init__(
        self,
        categorical_input_distinct_values: int,
        distilbert_pretrained_model_name: str,
        pre_classifier_units: int,
        num_labels: int,
    ):
        super(DistilBertWithCategorial, self).__init__()
        self.distilbert = DistilBertModel.from_pretrained(
            distilbert_pretrained_model_name
        )

        self.text_feat_linear = torch.nn.Linear(
            self.distilbert.config.dim, self.distilbert.config.dim
        )
```

```
import torch
from transformers import DistilBertModel


class DistilBertWithCategorial(torch.nn.Module):
    def __init__(
        self,
        categorical_input_distinct_values: int,
        distilbert_pretrained_model_name: str,
        pre_classifier_units: int,
        num_labels: int,
    ):
        super(DistilBertWithCategorial, self).__init__()
        self.distilbert = DistilBertModel.from_pretrained(
            distilbert_pretrained_model_name
        )

        self.text_feat_linear = torch.nn.Linear(
            self.distilbert.config.dim, self.distilbert.config.dim
        )
```

```
        self.distilbert = DistilBertModel.from_pretrained(  
            distilbert_pretrained_model_name  
        )  
  
        self.text_feat_linear = torch.nn.Linear(  
            self.distilbert.config.dim, self.distilbert.config.dim  
        )  
        self.text_feat_bn = torch.nn.BatchNorm1d(self.distilbert.config.dim)  
  
        self.cat_embedding = torch.nn.Embedding(  
            categorical_input_distinct_values, self.distilbert.config.dim  
        )  
        self.cat_bn = torch.nn.BatchNorm1d(self.distilbert.config.dim)  
  
        self.pre_classifier_linear = torch.nn.Linear(  
            self.distilbert.config.dim, pre_classifier_units  
        )  
        self.pre_classifier_bn = torch.nn.BatchNorm1d(pre_classifier_units)  
  
        self.classifier = torch.nn.Linear(pre_classifier_units, num_labels)  
  
    def forward(  
        ...  
    )
```

```
def forward(
    self,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    categorical_input: int,
):
    distilbert_output = self.distilbert(
        input_ids=input_ids, attention_mask=attention_mask
    )
    last_hidden_state = distilbert_output[0]
    text_features = last_hidden_state[:, 0]

    text_features = self.text_feat_linear(text_features)
    text_features = self.text_feat_bn(text_features)
    text_features = torch.nn.ReLU()(text_features)

    categorical_features = self.cat_embedding(categorical_input)
    categorical_features = self.cat_bn(categorical_features)
    categorical_features = torch.nn.ReLU()(categorical_features)

    combined_features = torch.add(text_features, categorical_features)

    combined_features = self.pre_classifier_linear(combined_features)
    combined_features = self.pre_classifier_bn(combined_features)
    combined_features = torch.nn.ReLU()(combined_features)

    logits = self.classifier(combined_features)
    return logits
```

```
def forward(
    self,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    categorical_input: int,
):
    distilbert_output = self.distilbert(
        input_ids=input_ids, attention_mask=attention_mask
    )
    last_hidden_state = distilbert_output[0]
    text_features = last_hidden_state[:, 0]

    text_features = self.text_feat_linear(text_features)
    text_features = self.text_feat_bn(text_features)
    text_features = torch.nn.ReLU()(text_features)

    categorical_features = self.cat_embedding(categorical_input)
    categorical_features = self.cat_bn(categorical_features)
    categorical_features = torch.nn.ReLU()(categorical_features)

    combined_features = torch.add(text_features, categorical_features)

    combined_features = self.pre_classifier_linear(combined_features)
    combined_features = self.pre_classifier_bn(combined_features)
    combined_features = torch.nn.ReLU()(combined_features)

    logits = self.classifier(combined_features)
    return logits
```

```
def forward(
    self,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    categorical_input: int,
):
    distilbert_output = self.distilbert(
        input_ids=input_ids, attention_mask=attention_mask
    )
    last_hidden_state = distilbert_output[0]
    text_features = last_hidden_state[:, 0]

    text_features = self.text_feat_linear(text_features)
    text_features = self.text_feat_bn(text_features)
    text_features = torch.nn.ReLU()(text_features)

    categorical_features = self.cat_embedding(categorical_input)
    categorical_features = self.cat_bn(categorical_features)
    categorical_features = torch.nn.ReLU()(categorical_features)

    combined_features = torch.add(text_features, categorical_features)

    combined_features = self.pre_classifier_linear(combined_features)
    combined_features = self.pre_classifier_bn(combined_features)
    combined_features = torch.nn.ReLU()(combined_features)

    logits = self.classifier(combined_features)
    return logits
```

```
def forward(
    self,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    categorical_input: int,
):
    distilbert_output = self.distilbert(
        input_ids=input_ids, attention_mask=attention_mask
    )
    last_hidden_state = distilbert_output[0]
    text_features = last_hidden_state[:, 0]

    text_features = self.text_feat_linear(text_features)
    text_features = self.text_feat_bn(text_features)
    text_features = torch.nn.ReLU()(text_features)

    categorical_features = self.cat_embedding(categorical_input)
    categorical_features = self.cat_bn(categorical_features)
    categorical_features = torch.nn.ReLU()(categorical_features)

    combined_features = torch.add(text_features, categorical_features)

    combined_features = self.pre_classifier_linear(combined_features)
    combined_features = self.pre_classifier_bn(combined_features)
    combined_features = torch.nn.ReLU()(combined_features)

    logits = self.classifier(combined_features)
    return logits
```

```
def forward(
    self,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    categorical_input: int,
):
    distilbert_output = self.distilbert(
        input_ids=input_ids, attention_mask=attention_mask
    )
    last_hidden_state = distilbert_output[0]
    text_features = last_hidden_state[:, 0]

    text_features = self.text_feat_linear(text_features)
    text_features = self.text_feat_bn(text_features)
    text_features = torch.nn.ReLU()(text_features)

    categorical_features = self.cat_embedding(categorical_input)
    categorical_features = self.cat_bn(categorical_features)
    categorical_features = torch.nn.ReLU()(categorical_features)

    combined_features = torch.add(text_features, categorical_features)

    combined_features = self.pre_classifier_linear(combined_features)
    combined_features = self.pre_classifier_bn(combined_features)
    combined_features = torch.nn.ReLU()(combined_features)

    logits = self.classifier(combined_features)
    return logits
```

```
def forward(
    self,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    categorical_input: int,
):
    distilbert_output = self.distilbert(
        input_ids=input_ids, attention_mask=attention_mask
    )
    last_hidden_state = distilbert_output[0]
    text_features = last_hidden_state[:, 0]

    text_features = self.text_feat_linear(text_features)
    text_features = self.text_feat_bn(text_features)
    text_features = torch.nn.ReLU()(text_features)

    categorical_features = self.cat_embedding(categorical_input)
    categorical_features = self.cat_bn(categorical_features)
    categorical_features = torch.nn.ReLU()(categorical_features)

    combined_features = torch.add(text_features, categorical_features)

    combined_features = self.pre_classifier_linear(combined_features)
    combined_features = self.pre_classifier_bn(combined_features)
    combined_features = torch.nn.ReLU()(combined_features)

    logits = self.classifier(combined_features)
    return logits
```

```
def forward(
    self,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    categorical_input: int,
):
    distilbert_output = self.distilbert(
        input_ids=input_ids, attention_mask=attention_mask
    )
    last_hidden_state = distilbert_output[0]
    text_features = last_hidden_state[:, 0]

    text_features = self.text_feat_linear(text_features)
    text_features = self.text_feat_bn(text_features)
    text_features = torch.nn.ReLU()(text_features)

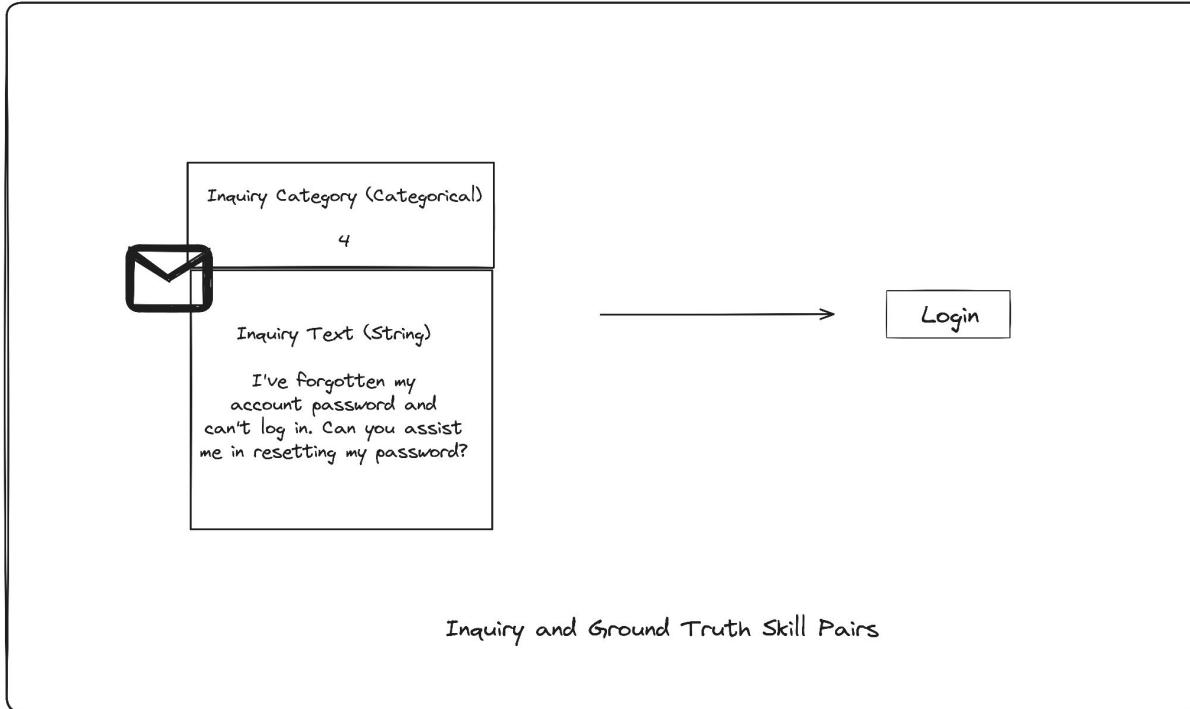
    categorical_features = self.cat_embedding(categorical_input)
    categorical_features = self.cat_bn(categorical_features)
    categorical_features = torch.nn.ReLU()(categorical_features)

    combined_features = torch.add(text_features, categorical_features)

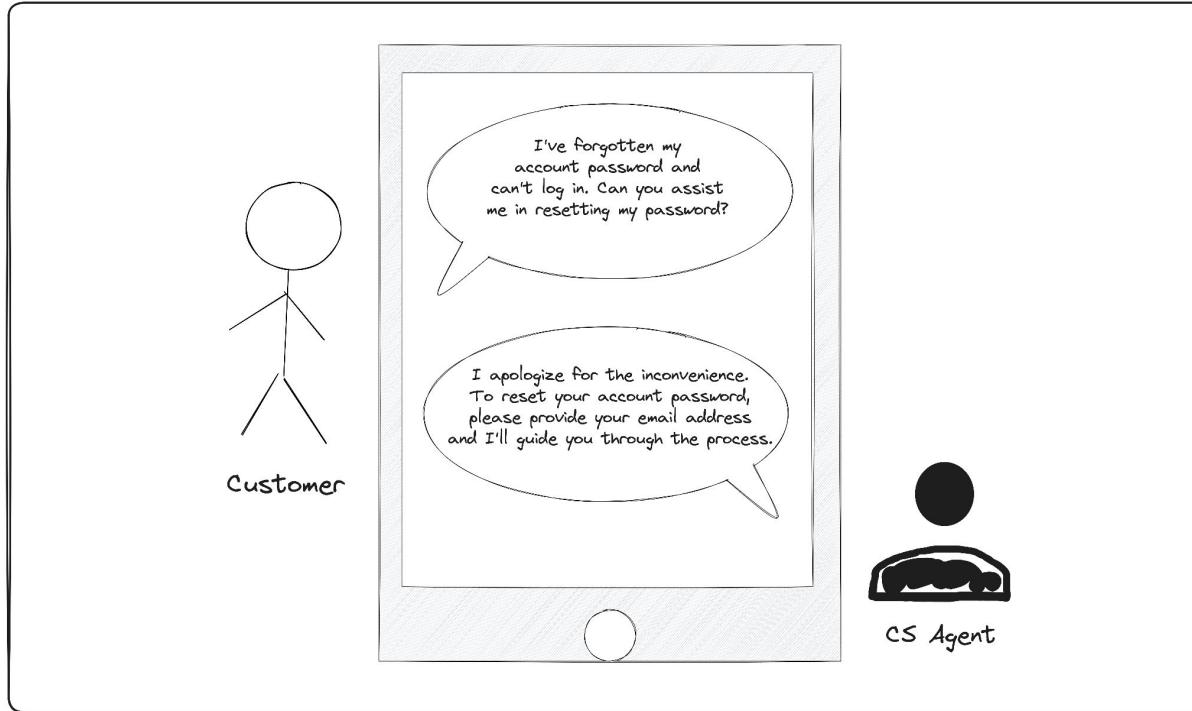
    combined_features = self.pre_classifier_linear(combined_features)
    combined_features = self.pre_classifier_bn(combined_features)
    combined_features = torch.nn.ReLU()(combined_features)

    logits = self.classifier(combined_features)
    return logits
```

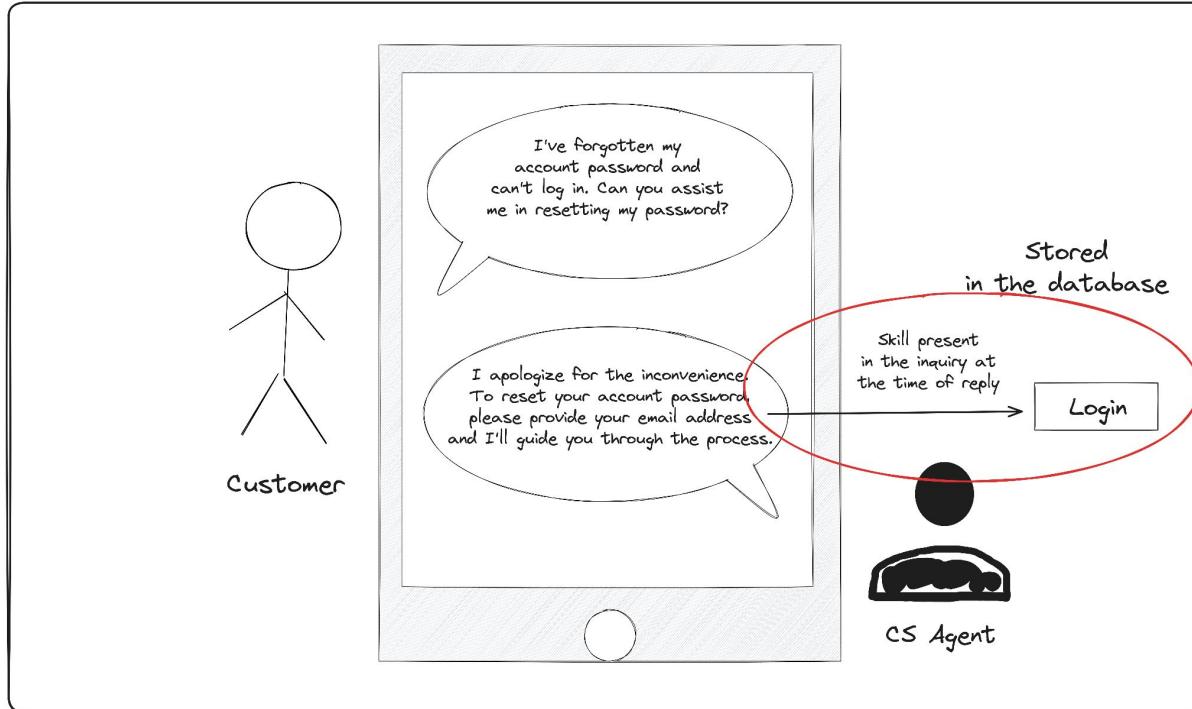
I Gathering Annotated Data



Gathering Annotated Data



Gathering Annotated Data



I Fine-Tuning Strategy

- Goal: Improve the accuracy of skill prediction
- Model:
 - DistilBert model to extract text features
 - Embedding layer to convert inquiry category to categorical features
 - Combine text and categorical features to predict the skill of the inquiry
- Data: Gather annotated data from last 1 year
 - 8 months of data for training
 - 1 month of data for validation
 - 3 months of data for testing

04

Business impact

| A/B Test

- Control: Use baseline algorithm to predict skill for inquiries
- Treatment: Use ML model to predict skill for inquiries
- Time period: 1 week

A/B Test Metrics

- Goal:
 - Accuracy of assigning skills
 - Transfer Rate
- Guardrail:
 - Average number of transfers per reply

| A/B Test Results

Metric	Change in Treatment compared to Control
Accuracy (of assigning skill)	+8.2%
Transfer Rate	-4.7%
Average number of transfers per reply	-26.4%

| What We Learnt!

- Routing customer inquiries to CS agents using skill of the inquiry
- Business problem (reducing transfer rate) =>
ML problem (improving the accuracy of skill prediction algorithm)
- Using NLP to build an ML model to predict the skill required to handle the inquiries
- Impact from using the ML model



I Appendix

- [The Illustrated Transformer](#)
 - Understanding transformer models
- [Multimodal-Toolkit](#)
 - Combining text features with numerical and categorical ones in different ways
- [primaprashant/ai-customer-support](#)
 - Collection of blogs and papers on using ML for customer support