SSAFY 특강: Router 와 Reward Model을 통해 배우는 효과적인 Reasoning

# Day #3: 다양한 Router 학습하기

Day #3: 다양한 Router 학습하기

# TF-IDF와 Router를 활용한 분류기 학습

## TF-IDF 란?

t를 단어(token), d를 문서(document) 라고 할때

$$TF - IDF(t, d) = TF(t, d) \times IDF(t)$$

이때,
- TF 는 특정 문서 (d) 에서 단어(t)의 상대적 빈도를 의미함.
(문서 d 안에서 단어 t가 얼마나 자주 나오는가)

- IDF 는 전체 말뭉치에서 단어 (t)의 희귀성을 의미함.
(전체 문서들 가운데 단어 t가 얼마나 희귀한가)
(희귀할 수록 값이 커짐)

TF-IDF 는 문서 내부 빈도와 전체 희귀성의 곱으로 “문서 d에서 t의 중요도” 를 의미함.

## TF 란?

$$TF(t, d) = count(t, d)$$

예) d = “time flies like an arrow time”

|d| = 6, count(“time”, d) = TF(“time”, d) = 2

$$TF(t, d) = count(t, d) \div |d| \qquad TF(t, d) = 1 + ln(count(t, d))$$

예) d = “time flies like an arrow time”

$$TF_{len} = 2 \div 6 = 0.3333... \quad , \quad TF_{log} = 1 + ln2 = 1.693$$

## IDF 란?

$$IDF(t) = ln(N \div df(t))$$

이때
- df 는 단어 t가 등장한(한 번이라도 등장한) 문서의 개수를 의미함
- N은 전체 문서의 수를 의미함

N = 3, df(“the”) = 3, df(“time”) = 1, IDF(“the”) = 0 IDF(“time”) = 1.09

# TF-IDF 를 활용한 분류기 학습

```python
import pandas as pd

# Hugging Face 데이터셋 주소"
url = 'https://huggingface.co/datasets/Blpeng/nsmc/resolve/main/ratings.csv'
df = pd.read_csv(url)
# NSMC 데이터는 id, document, label 세 컬럼을 포함합니다.
df = df[['document', 'label']].rename(columns={'document': 'sentence'})
df =df.dropna()
```

# TF-IDF 를 활용한 분류기 학습

```python
# 간단한 전처리: 문장을 공백 기준으로 토큰화
# 한국어 형태소 분석기가 없기 때문에 공백 기준으로 나눕니다.
import re

def tokenize(sentence):
    # 특수 문자 제거하고 공백으로 분리
    sentence = re.sub(r"[^가-힣0-9\s]", "", sentence)
    tokens = sentence.strip().split()
    return tokens

# 토큰화 적용

df['tokens'] = df['sentence'].apply(tokenize)
df
```

다양한 토크나이저들: khaiii, konlpy, pymecab-ko, tiktoken

## TF-IDF 를 활용한 분류기 학습

khaiii (Kakao 형태소분석기): CNN 기반 한국어 형태소/품사 분석.

konlpy : Okt, Mecab, Komoran, Kkma, Hannanum 을 제공하는 한국어 형태소 분석

pymecab-ko : Mecab의 파이썬/한국어 버전

tiktoken : 형태소 분석 X, GPT-4o가 사용하는 토크나이저

# TF-IDF 를 활용한 분류기 학습

```python
# TF-IDF 벡터화를 이용한 분류기 학습
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# 학습용과 테스트용 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(df['sentence'], df['label'], test_size=0.3, random_state=42)

# TF-IDF 벡터 생성기
vectorizer = TfidfVectorizer(tokenizer=lambda text: text.split(), min_df=1)

# 학습 데이터에 대해 벡터화 수행
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

# TF-IDF 를 활용한 분류기 학습

```python
# 로지스틱 회귀 분류기 학습
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train_tfidf, y_train)

# 테스트 데이터 평가
pred = clf.predict(X_test_tfidf)
print("TF-IDF 분류 결과:")
print(classification_report(y_test, pred))
```

# TF-IDF 를 활용한 분류기 학습

TF-IDF 분류 결과:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.77 | 0.82 | 0.80 | 30076 |
| 1 | 0.81 | 0.75 | 0.78 | 29922 |
| accuracy | | | 0.79 | 59998 |
| macro avg | 0.79 | 0.79 | 0.79 | 59998 |
| weighted avg | 0.79 | 0.79 | 0.79 | 59998 |

## Word2Vec 이란?

Word2Vec은 단어를 저차원 벡터로 임베딩하여 의미와 문법적 유사성을 공간상의 근접도로 표현함. 분포 가설(비슷한 문맥→비슷한 의미)을 학습으로 구현합니다.

- 각 단어 ↔ 하나의 dense vector (보통 50–300차원)
- 로컬 문맥 윈도우를 이용한 자기지도 학습
- 유사 단어는 가깝게, 관계는 벡터 연산으로 드러남(예: king – man + woman ≈ queen).

- sparse vector 인 TF-IDF와 달리 dense vector 임

## Word2Vec 이란?

학습 방식으로는 CBOW 와 skip-gram 이 존재함.
- **CBOW**: 주변 단어(문맥)로 중심 단어를 예측. 빠르고 빈도 높은 단어에 강함.
- **Skip-gram**: 중심 단어로 주변 단어를 예측. 희귀 단어 학습에 유리.
- *window* m: 문맥 크기(보통 2–10). 학습 시 무작위 가변 윈도우를 쓰기도 함.

# Word2Vec을 활용한 분류기 학습

```python
# 2. Word2Vec 학습
# sg=0는 CBOW, sg=1은 Skip-gram
w2v = Word2Vec(
    sentences=sentences,
    vector_size=100,
    window=8,
    min_count=5,
    workers=4,
    sg=0,
    epochs=5  # 필요 시 늘리세요
)

# 3. vocabulary 확인
vocab = list(w2v.wv.key_to_index.keys())
print(f"단어 수: {len(vocab)}")
print("예시 단어 20개:", vocab[:20])
```

# Word2Vec을 활용한 분류기 학습

```python
# 5. 문서 임베딩 만들기
# 각 문서의 토큰 중 vocab에 존재하는 단어 벡터 평균
def doc_vector(tokens, model):
    vecs = [model.wv[w] for w in tokens if w in model.wv]
    if len(vecs) == 0:
        return np.zeros(model.vector_size, dtype=np.float32)
    return np.mean(vecs, axis=0)

X = np.vstack([doc_vector(toks, w2v) for toks in sentences])

# 레이블 y 준비. 이미 df['label']이 0,1 형태라면 그대로 사용
if 'label' in df.columns:
    y = df['label'].values
else:
    # 레이블이 없으면 모두 0으로 세팅. 실제 실험에서는 반드시 레이블을 제공하세요.
    y = np.zeros(len(df), dtype=int)
```

# Word2Vec을 활용한 분류기 학습

```python
# 6. 분류기 학습과 평가
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=SEED, stratify=y if len(np.unique(y)) > 1 else None
)

clf = LogisticRegression(
    max_iter=200,
    n_jobs=None  # 최신 sklearn에서는 n_jobs가 제거된 경우가 있어 옵션 생략 권장
)
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, pred))
print(classification_report(y_test, pred))
```

# Word2Vec을 활용한 분류기 학습

Accuracy: 0.6863171579289482

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.67 | 0.74 | 0.70 | 20000 |
| 1 | 0.71 | 0.63 | 0.67 | 19999 |
| accuracy | | | 0.69 | 39999 |
| macro avg | 0.69 | 0.69 | 0.69 | 39999 |
| weighted avg | 0.69 | 0.69 | 0.69 | 39999 |

Day #3: 다양한 Router 학습하기

# TF-IDF와 Router를 활용한 Router 학습

# TF-IDF을 활용한 Router 학습

```python
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report

# 1) char ngram 기반
tfidf_char = Pipeline([
    ("tfidf", TfidfVectorizer(analyzer="char", ngram_range=(2,5), min_df=3)),
    ("clf", LinearSVC())
])
tfidf_char.fit(X_train, y_train)
pred_char = tfidf_char.predict(X_test)
acc_char = accuracy_score(y_test, pred_char)
print("TF-IDF char Accuracy:", acc_char)
print(classification_report(y_test, pred_char))
```

# TF-IDF을 활용한 Router 학습

```
# 2) word 기반
tfidf_word = Pipeline([
    ("tfidf", TfidfVectorizer(analyzer="word", token_pattern=r"(?u)\b\w+\b", min_df=3)),
    ("clf", LinearSVC())
])
tfidf_word.fit(X_train, y_train)
pred_word = tfidf_word.predict(X_test)
acc_word = accuracy_score(y_test, pred_word)
print("TF-IDF word Accuracy:", acc_word)
print(classification_report(y_test, pred_word))
```

# TF-IDF을 활용한 Router 학습

```
TF-IDF char Accuracy: 0.99081944916695
              precision    recall  f1-score   support

         B13       0.99      0.92      0.95       254
          K7       1.00      1.00      1.00       572
          M7       0.99      0.99      0.99       159
         WB7       0.99      1.00      0.99      1956

    accuracy                           0.99      2941
   macro avg       0.99      0.98      0.98      2941
weighted avg       0.99      0.99      0.99      2941

TF-IDF word Accuracy: 0.9806188371302278
              precision    recall  f1-score   support

         B13       0.96      0.83      0.89       254
          K7       1.00      0.99      1.00       572
          M7       0.99      0.97      0.98       159
         WB7       0.98      1.00      0.99      1956

    accuracy                           0.98      2941
   macro avg       0.98      0.95      0.96      2941
weighted avg       0.98      0.98      0.98      2941
```

# Word2Vec을 활용한 Router 학습

```python
import re
import numpy as np

from gensim.models import Word2Vec

def simple_tokenize(text: str):
    text = re.sub(r"[^가-힣A-Za-z0-9\s]", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text.split()


train_tokens = [simple_tokenize(t) for t in X_train]
test_tokens  = [simple_tokenize(t) for t in X_test]

w2v = Word2Vec(
    sentences=train_tokens,
    vector_size=100,
    window=8,
    min_count=5,
    workers=4,
    sg=0,
    epochs=5
)
```

```
model = Word2Vec(
    sentences,
    sg=1,
    negative=10,
    sample=1e-5,
    window=5,
    vector_size=200,
    min_count=2,
    epochs=5,
    workers=4
)
```

# Word2Vec을 활용한 Router 학습

```python
def doc_vec(tokens, model):
    vecs = [model.wv[w] for w in tokens if w in model.wv]
    if not vecs:
        return np.zeros(model.vector_size, dtype=np.float32)
    return np.mean(vecs, axis=0)

Xtr_w2v = np.vstack([doc_vec(t, w2v) for t in train_tokens])
Xte_w2v = np.vstack([doc_vec(t, w2v) for t in test_tokens])

from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report
clf_w2v = LinearSVC()
clf_w2v.fit(Xtr_w2v, y_train)
pred_w2v = clf_w2v.predict(Xte_w2v)
acc_w2v = accuracy_score(y_test, pred_w2v)
print("Word2Vec LinearSVC Accuracy:", acc_w2v)
print(classification_report(y_test, pred_w2v))
```

# Word2Vec을 활용한 Router 학습

```
Word2Vec LinearSVC Accuracy: 0.9666780006800408
              precision    recall  f1-score   support

         B13       0.90      0.78      0.83       254
          K7       0.99      0.98      0.99       572
          M7       0.96      0.95      0.96       159
         WB7       0.97      0.99      0.98      1956

    accuracy                           0.97      2941
   macro avg       0.95      0.92      0.94      2941
weighted avg       0.97      0.97      0.97      2941
```

# Test 셋에서 평가하기

```
y_pred_char = tfidf_char.predict(test['question'].astype(str).values)
print("\n[TF-IDF char] Accuracy:", accuracy_score(test['route_rule'], y_pred_char))
print(classification_report(test['route_rule'], y_pred_char))
```

```
[TF-IDF char] Accuracy: 0.9852
              precision    recall  f1-score   support

         B13       0.97      0.91      0.94       255
          K7       0.99      1.00      1.00       555
          M7       1.00      0.95      0.98       190
         WB7       0.98      1.00      0.99      1500

    accuracy                           0.99      2500
   macro avg       0.99      0.96      0.97      2500
weighted avg       0.99      0.99      0.98      2500
```

# LiteLLM 과 연결하기

```
mmap = {
    "B13": "openrouter/qwen/qwen3-14b:free",
    "K7": "openrouter/google/gemma-3-12b-it:free",
    "M7": "openrouter/meta-llama/llama-3.3-8b-instruct:free",
    "WB7": "openrouter/google/gemma-3n-e4b-it:free"
}

question = "씨 감자를 심으려 한다. 씨감자의 무게가 80g이라면 몇 조각으로 자르는 것이 가장 좋은가?"
pred_model = tfidf_word.predict([question])[0]

or_model = mmap[pred_model]
```

# LiteLLM 과 연결하기

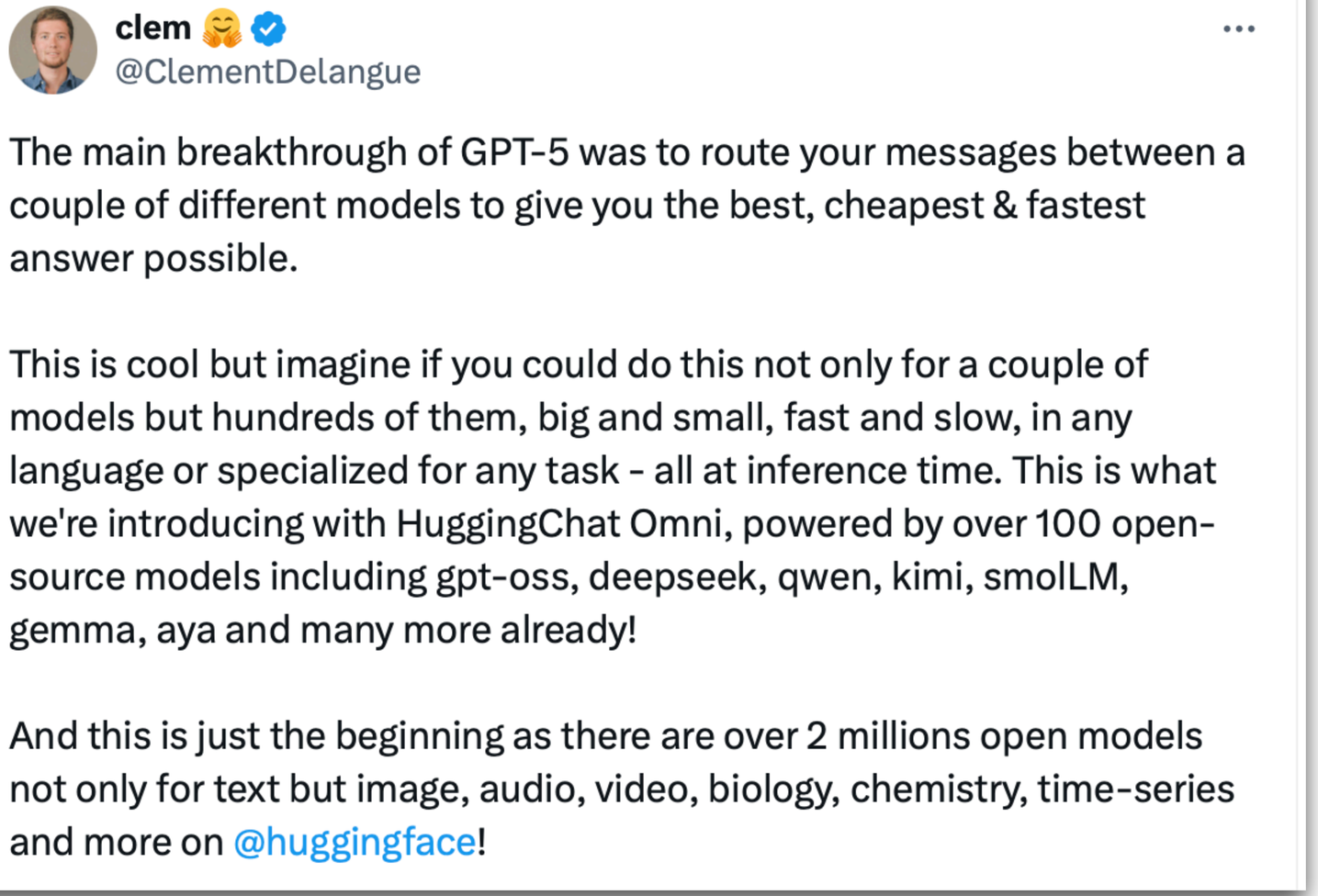
```
from litellm import completion
import os

## set ENV variables
os.environ["OPENROUTER_API_KEY"] = "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"

response = completion(
  model=or_model,
  messages=[{ "content": question,"role": "user"}]
)
```

```
print(response.choices[0].message.content)
```

Day #3: 다양한 Router 학습하기

# Routers in Real Life

# Router를 적용한 채팅 서비스를 선보인 HuggingFace



# HuggingFace 가 사용한 Arch-Router

With the rapid proliferation of large language models (LLMs) – each optimized for different strengths, style, or latency/cost profile – routing has become an essential technique to operationalize the use of different models. However, existing LLM routing approaches are limited in two key ways: they evaluate performance using benchmarks that often fail to capture human preferences driven by subjective evaluation criteria, and they typically select from a limited pool of models. In this work, we propose a preference-aligned routing framework that guides model selection by matching queries to user-defined domains (e.g., travel) or action types (e.g., image editing) – offering a practical mechanism to encode preferences in routing decisions. Specifically, we introduce **Arch-Router**, a compact 1.5B model that learns to map queries to domain-action preferences for model routing decisions. Our approach also supports seamlessly adding new models for routing without requiring retraining or architectural modifications. Experiments on conversational datasets demonstrate that our approach achieves state-of-the-art (SOTA) results in matching queries with human preferences, outperforming top proprietary models. Our approach captures subjective evaluation criteria and makes routing decisions more transparent and flexible. Our model is available at: `https://huggingface.co/katanemo/Arch-Router-1.5B`.

# HuggingFace 가 사용한 Arch-Router

**Domain–Action Taxonomy** In this work, we focus on modeling LLM routing to align with human preferences driven by subjective evaluation criteria. To incorporate human preferences as routing objectives, we adopt a Domain–Action taxonomy, a two-level hierarchy that mirrors how people typically describe tasks—starting with a general topic and narrowing to a specific action. ***Domain*** (e.g., legal and finance) captures the high-level thematic context of a query while ***Action*** (e.g., summarization and code generation) denotes the specific operation requested. This taxonomy serves as a mental model to help users define clear and structured routing policies. Separating Domain and Action strikes a balance between expressiveness and simplicity: it avoids an unwieldy flat list of composite labels (e.g., finance summarization and legal advice) and introduces a natural fallback. If a query is too vague to match an Action, the router can still resolve the Domain—maintaining robustness and reducing semantic ambiguity.

# HuggingFace 가 사용한 Arch-Router

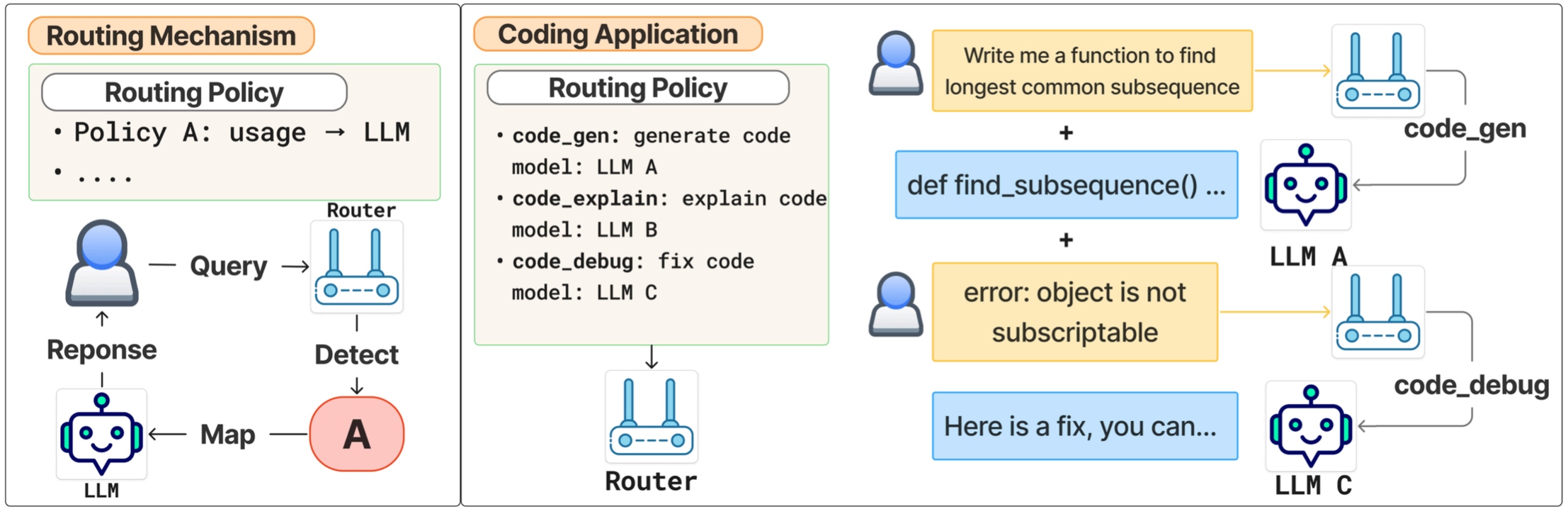



Figure 1: Preference-Aligned Routing Mechanism. The routes policies and user conversation is provided to the router to select the appropriate policy and corresponding LLM. Example of usage in a coding application is shown in the right.

# HuggingFace 가 사용한 Arch-Router

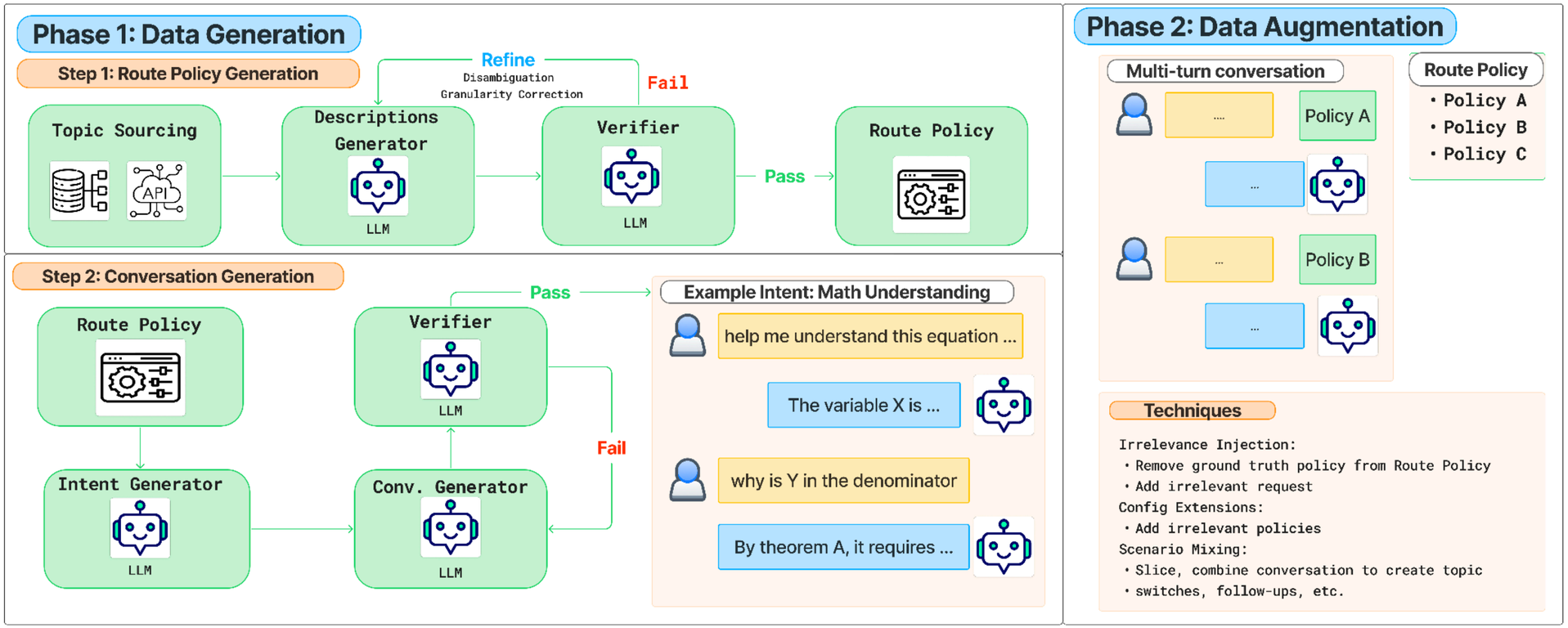



Figure 2: Overview of the data creation for Arch-Router framework. Phase 1 generates route configurations through an LLM process with feedback loops. Phase 2 generates conversations from generated intent. Phase 3 augments the conversations to get diverse scenarios and irrelevance.

# HuggingFace 가 사용한 Arch-Router

**Data Generation.** Phase 1 produces clean, labeled conversations through a structured two-step process. First, we generate route policies by constructing a diverse topic pool from industry classifications [20], academic benchmarks such as MMLU [8], and real-world API documentation [19]. An LLM is used to generate candidate route policies from this pool. These candidates are then validated and refined by a second LLM to ensure clarity, appropriate granularity (as defined by the Domain-Action Taxonomy 3.2), and semantic coherence. Second, we synthesize conversations using the curated set of route policies. For each policy, an LLM generates a specific conversational intent, which is then passed to another LLM to produce a full dialogue. To ensure data quality, a final LLM verifies the alignment between the conversation and the intended routing policy. Conversations that fail this check are regenerated.

# HuggingFace 가 사용한 Arch-Router

**Data Augmentation.** Phase 2 enhances the dataset by systematically incorporating real-world complexity to improve the robustness of the routing model. We apply three augmentation techniques to diversify conversational patterns: 1) Irrelevance injection introduces noise by adding off-topic user messages or removing the ground-truth route policy from a sample, simulating ambiguity in user intent, 2) Policy modification alters the candidate set of route policies by including irrelevant or misleading options, creating more challenging decision boundaries, and 3) Scenario mixing enriches the data by combining segments from different conversations to create longer dialogues with abrupt topic shifts, follow-up questions, and abandoned intents. These augmentations yield a more representative and challenging dataset, essential for training a routing model that generalizes well to real-world usage.

# HuggingFace 가 사용한 Arch-Router

**Training Details.** We adopt Supervised fine-tuning (SFT) [25], carried out with the Llama-Factory library [46], using full-parameter updates in bfloat16 precision, the AdamW optimizer, and a maximum of four epochs on a single NVIDIA A100 GPU. We train over 43,000 pairs of samples with training/validation split as 90/10. Full training data details are shown in Table 4.

# HuggingFace 가 사용한 Arch-Router

| Models | Turn | Span | Conv. | Overall |
|---|---|---|---|---|
| Qwen2.5-1.5B | 34.37 | 16.09 | 11.61 | 20.69 |
| GPT-4o | 93.96 | 90.74 | 84.52 | 89.74 |
| GPT-4o-mini | 87.49 | 80.89 | 80.00 | 82.79 |
| Claude-sonnet-3.7 | **96.24** | 94.70 | 87.45 | 92.79 |
| Claude-haiku-3.5 | 88.21 | 81.46 | 83.72 | 84.96 |
| Gemini-2.0-flash | 89.19 | 85.05 | 85.79 | 85.63 |
| Gemini-2.0-flash-lite | 85.13 | 72.61 | 74.76 | 76.69 |
| **Arch-Router** | 96.05 | **94.98** | **88.48** | **93.17** |

# HuggingFace 가 사용한 Arch-Router

| Route Policy | Description | LLM |
|---|---|---|
| `code_generation` | Generate new code snippets, functions, or boiler-plate from user requirements. | Claude-sonnet-3.7 |
| `code_explanation` | Explain what a piece of code does, including logic and edge cases. | GPT-4o |
| `bug_fixing` | Identify and fix errors or bugs in user-supplied code. | GPT-4o |
| `performance_optimization` | Suggest changes to make code faster, more scalable, or more readable. | GPT-4o |
| `api_help` | Assist with understanding or integrating external APIs and libraries. | GPT-4o-mini |
| `programming_question` | Answer general programming theory or best-practice questions. | GPT-4o-mini |
| `default` | | Qwen2.5-4B |

Table 10: Route policy for the coding application.

# HuggingFace 가 사용한 Arch-Router

| Turn | User request | RouteLLM | Arch-Router |
|---|---|---|---|
| 1 | Hi | `weak` ✓ | `general` ✓ |
| 2 | Write a function to visualize an dataframe that has error column, the visualization the accuracy aggregation over all the rows | `strong` ✓ | `code_generation` ✓ |
| 3 | This doesnt work | `weak` ✗ | `bug_fixing`✓ |
| 4 | Okay, now try the same thing for a dataframe with a label, output column, please check the correctness of the output column compared to the label, note that there is different t incorrect types: relevance, irrelevance, and incorrect format | `strong` ✓ | `code_generation`✓ |
| 5 | The code runs too slow, any way u can simplify it to make it faster? | `weak` ✗ | `performance_optimization` ✓ |
| 6 | What are the functions that can be replaced from seaborn | `strong` ✓ | `api_help` ✓ |
| 7 | Any other ones? | `weak` ✗ | `api_help` ✓ |
| 8 | Thats all, thank you | `weak` ✓ | `general` ✓ |

Table 11: Comparison of routing predictions from RouteLLM and Arch-Router for each user turn in a coding conversation.

Day #3: 다양한 Router 학습하기

# Future of LLMs