1. 현황 분석

1.1 문제 현상

• 채용공고 데이터에서 심각한 중복 문제 발생

• 동일 내용의 반복적 기술 및 동의어 혼재

• HTML 태그, JavaScript 코드, SEO 키워드 등 노이즈 데이터 포함

1.2 근본 원인

구조적 문제

• 데이터 수집 단계: 크롤링 시 불필요한 요소까지 모두 수집

• 전처리 부재: 수집 데이터를 정제 없이 저장

• 중복 기준 모호: 단순 문자열 비교로 의미적 중복 감지 불가

기술적 한계

• 정규식 기반 정제의 한계

• 해시 기반 중복 제거는 완전 동일 텍스트만 감지

• 동의어 처리 로직 부재

1.3 비즈니스 영향

• 검색 품질 저하

• 사용자 이탈률 증가

• 프리미엄 고객 클레임 지속 발생

2. GenAI 기반 해결 전략 및 구현

2.1 핵심 접근 방식

LLM을 도메인 전문가의 사고 과정을 모방하는 파트너로 활용하여 채용 담당자가 공고를 읽듯이 데이터를 이해하고 정제하는 시스템 구축

2.2 구현 코드

2.2.1 구조화 정제 모듈

import json

from typing import Dict, List, Any

from openai import OpenAI

import re

from datetime import datetime

class JobPostingRefiner:

def \_\_init\_\_(self, api\_key: str):

self.client = OpenAI(api\_key=api\_key)

def structure\_and\_clean(self, raw\_text: str) -> Dict[str, Any]:

STRUCTURING\_PROMPT = """

당신은 10년 경력의 채용 전문가입니다. 수천 개의 채용공고를 분석한 경험을 바탕으로

다음 채용공고에서 구직자에게 꼭 필요한 정보만 추출해주세요.

[원본 텍스트]

{raw\_text}

[추출할 정보]

- 직무명: 실제 수행할 업무를 나타내는 명확한 직무명

- 필수 요구사항: 지원자가 반드시 갖춰야 할 조건

- 주요 업무: 입사 후 실제로 하게 될 일

- 우대 사항: 있으면 좋은 추가 역량

- 회사 소개: 회사의 핵심 가치와 비전

[처리 규칙]

1. 과장된 표현은 사실 위주로 정제 ("최고의" → 구체적 성과로)

2. 중복된 내용은 한 번만 기술

3. 동의어는 업계 표준 용어로 통일

4. SEO를 위한 키워드 나열은 제거

[출력 형식]

JSON 형식으로 구조화하여 반환

"""

try:

response = self.client.chat.completions.create(

model="gpt-4",

messages=[

{"role": "system", "content": "You are a recruitment expert."},

{"role": "user", "content": STRUCTURING\_PROMPT.format(raw\_text=raw\_text)}

],

temperature=0.3

)

result = response.choices[0].message.content

return json.loads(result)

except Exception as e:

print(f"Error in structuring: {e}")

return {}

제가 이 프롬프트를 이렇게 설계한 이유는 LLM에게 단순히 "데이터를 정제하라"고 지시하는 것보다, 구체적인 역할과 맥락을 부여하는 것이 훨씬 효과적이기 때문입니다. "10년 경력의 채용 전문가"라는 페르소나를 설정함으로써, LLM이 단순한 텍스트 처리가 아닌 도메인 전문성을 발휘하도록 유도하였습니다. 실제로 채용 담당자가 공고를 검토할 때 주목하는 포인트들을 명시하여, LLM이 그들의 사고 과정을 따라가도록 하였습니다. 특히 "과장된 표현을 사실 위주로 정제"라는 지침은 현업에서 자주 마주치는 문제를 직접적으로 해결합니다. "업계 최고 대우"같은 모호한 표현보다는 "연봉 상위 10%"같은 구체적 정보가 구직자에게 더 유용하기 때문입니다.

2.2.2 의미 기반 중복 제거 모듈

python

class SemanticDeduplicator:

def \_\_init\_\_(self, api\_key: str):

self.client = OpenAI(api\_key=api\_key)

def find\_semantic\_duplicates(self, sentences: List[str]) -> Dict[str, List[str]]:

SEMANTIC\_DEDUPLICATION\_PROMPT = """

당신은 자연어 처리 전문가입니다.

다음 문장들을 읽고, 실질적으로 같은 의미를 전달하는 문장들을 찾아주세요.

중요한 것은 '표현'이 아니라 '의미'입니다.

[문장 리스트]

{sentences}

[판단 과정]

먼저 각 문장이 전달하려는 핵심 메시지를 파악합니다.

그 다음, 다음 기준으로 중복 여부를 판단합니다:

1. 정보의 포함 관계

예: "Python 필수" ⊂ "Python과 Django 필수"

2. 동의어 관계

예: "3년 이상 경력" = "3년차 이상"

3. 의도의 동일성

예: "우대합니다" = "환영합니다" = "좋습니다"

[사고 과정 보여주기]

Chain-of-Thought 방식으로 어떻게 판단했는지 단계별로 설명해주세요.

이렇게 하면 결과를 신뢰할 수 있고, 필요시 수정할 수 있습니다.

[최종 출력]

각 중복 그룹별로:

- 왜 중복인지 설명

- 어떤 문장을 대표로 선택했는지

- 선택 이유 (더 구체적, 더 포괄적 등)

JSON 형식으로 반환:

{

"groups": [

{

"representative": "대표 문장",

"duplicates": ["중복 문장1", "중복 문장2"],

"reasoning": "선택 이유"

}

]

}

"""

try:

response = self.client.chat.completions.create(

model="gpt-4",

messages=[

{"role": "system", "content": "You are an NLP expert."},

{"role": "user", "content": SEMANTIC\_DEDUPLICATION\_PROMPT.format(

sentences=json.dumps(sentences, ensure\_ascii=False)

)}

],

temperature=0.2

)

result = json.loads(response.choices[0].message.content)

return result

except Exception as e:

print(f"Error in deduplication: {e}")

return {"groups": []}

def deduplicate\_text(self, text: str) -> str:

sentences = [s.strip() for s in text.split('.') if s.strip()]

duplicate\_groups = self.find\_semantic\_duplicates(sentences)

cleaned\_sentences = []

processed = set()

for group in duplicate\_groups.get('groups', []):

cleaned\_sentences.append(group['representative'])

processed.add(group['representative'])

for dup in group['duplicates']:

processed.add(dup)

for sentence in sentences:

if sentence not in processed:

cleaned\_sentences.append(sentence)

return '. '.join(cleaned\_sentences)

설계의도: 특히 채용공고처럼 민감한 데이터를 다룰 때는 틀리면 안된다고 생각합니다. Chain-of-Thought을 사용한 이유는 LLM이 스스로의 추론 과정을 검증하도록 하기 위함입니다. 연구에 따르면 LLM에게 "생각하는 과정을 보여달라"고 요청하면 정확도가 평균적으로 매우 향상된다고 합니다. 또한 "정보의 포함 관계"같은 구체적인 판단 기준을 제시한 것은 실제 개발 현장에서 자주 마주치는 패턴들을 반영한 것입니다. 예를 들어 "Python 필수"와 "Python과 Django 필수"가 있을 때, 후자가 전자를 포함하므로 후자만 남기는 것이 정보 보존 측면에서 유리합니다.

2.2.3 쿼리 의도 파악 및 확장 모듈

python

class QueryIntentAnalyzer:

def \_\_init\_\_(self, api\_key: str):

self.client = OpenAI(api\_key=api\_key)

def expand\_query(self, user\_query: str) -> Dict[str, Any]:

QUERY\_UNDERSTANDING\_PROMPT = """

당신은 구직자의 마음을 읽는 검색 전문가입니다.

사용자가 입력한 짧은 검색어 뒤에 숨겨진 진짜 의도를 파악해주세요.

[검색어]

{user\_query}

[분석 관점]

1. 명시적 요구사항: 직접적으로 표현된 것

2. 암묵적 기대사항: 말하지 않았지만 당연히 기대하는 것

3. 관련 확장 개념: 함께 검색하면 좋을 연관 키워드

[실제 사례 기반 추론]

예를 들어 "스타트업 개발자"를 검색한 사람은:

- 명시적: 스타트업에서 일하는 개발자 포지션

- 암묵적: 스톡옵션, 자율적인 문화, 빠른 성장 기회

- 확장: 시리즈 A, B 투자 단계, 기술 스택 다양성

[도메인 지식 활용]

최신 IT 채용 트렌드를 반영하세요:

- "풀스택" 검색 → Node.js, React, Next.js 포함

- "AI 엔지니어" 검색 → ML, 딥러닝, PyTorch, LLM 포함

- "데브옵스" 검색 → 쿠버네티스, CI/CD, 클라우드 포함

[출력]

JSON 형식으로 반환:

{

"original\_query": "원본 검색어",

"explicit\_intent": ["명시적 의도 리스트"],

"implicit\_expectations": ["암묵적 기대사항"],

"expanded\_keywords": ["확장 키워드"],

"reasoning": "확장 이유 설명"

}

"""

try:

response = self.client.chat.completions.create(

model="gpt-4",

messages=[

{"role": "system", "content": "You are a search intent expert."},

{"role": "user", "content": QUERY\_UNDERSTANDING\_PROMPT.format(

user\_query=user\_query

)}

],

temperature=0.4

)

result = json.loads(response.choices[0].message.content)

return result

except Exception as e:

print(f"Error in query expansion: {e}")

return {

"original\_query": user\_query,

"expanded\_keywords": [user\_query]

}

def build\_search\_query(self, expanded\_query: Dict[str, Any]) -> str:

all\_keywords = (

[expanded\_query['original\_query']] +

expanded\_query.get('explicit\_intent', []) +

expanded\_query.get('implicit\_expectations', []) +

expanded\_query.get('expanded\_keywords', [])

)

unique\_keywords = list(set(all\_keywords))

return ' OR '.join(unique\_keywords)

설계의도: 검색의 핵심은 ‘지원자가 진짜 원하는 것’을 찾아주는 데 있습니다. 많은 지원자가 니즈를 정확히 말로 표현하지 못하기에, 저는 사용자(채용담당자·현업)의 숨은 의도를 읽어내는 태도를 중요하게 봅니다. 그래서 LLM을 쓸 때도 단순 키워드 매칭이 아니라 공감 맥락을 프롬프트에 반영하고, 제가 겪은 암묵적 기대사항(예: “스타트업 개발자”가 원하는 성장·도전·자율 문화)을 명시합니다. 또한 추상적 지시 대신 \*\*구체 사례(Few-shot)\*\*를 제공해 모델이 패턴을 더 잘 잡도록 설계합니다.

2.2.4 품질 자동 평가 모듈

python

class QualityAssessor:

def \_\_init\_\_(self, api\_key: str):

self.client = OpenAI(api\_key=api\_key)

def assess\_quality(self, original: str, cleaned: str) -> Dict[str, Any]:

QUALITY\_ASSESSMENT\_PROMPT = """

당신은 까다로운 품질 관리 전문가입니다.

정제된 데이터가 실제로 사용자에게 도움이 될지 냉정하게 평가해주세요.

[원본 데이터]

{original}

[정제된 데이터]

{cleaned}

[평가 기준 - 사용자 관점]

1. 핵심 정보 보존: 지원 결정에 필요한 정보가 다 있는가?

2. 가독성: 한눈에 이해되는가?

3. 신뢰성: 과장이나 모순은 없는가?

4. 차별성: 다른 공고와 구별되는 특징이 살아있는가?

[비판적 검토]

특히 다음 사항을 엄격하게 체크하세요:

- 중요한 정보가 실수로 삭제되지 않았는지

- 과도한 요약으로 뉘앙스가 왜곡되지 않았는지

- 동의어 통일 과정에서 의미 변질이 없었는지

[점수 산정]

각 항목을 10점 만점으로 평가하되,

"대충 7-8점"이 아니라 명확한 근거와 함께 점수를 매겨주세요.

예시:

"가독성 4/10 - 한 문단이 너무 길어 모바일에서 읽기 어려움.

3-4개 문장으로 단락을 나누면 8/10까지 개선 가능"

[개선 제안]

낮은 점수를 받은 항목에 대해 구체적인 개선 방안을 제시하세요.

"더 잘해야 한다"가 아니라 "이렇게 하면 된다"를 알려주세요.

[출력]

JSON 형식으로 반환:

{

"scores": {

"information\_preservation": {"score": 0, "reasoning": ""},

"readability": {"score": 0, "reasoning": ""},

"reliability": {"score": 0, "reasoning": ""},

"differentiation": {"score": 0, "reasoning": ""}

},

"total\_score": 0,

"improvements": ["개선 제안 1", "개선 제안 2"],

"is\_acceptable": true/false

}

"""

try:

response = self.client.chat.completions.create(

model="gpt-4",

messages=[

{"role": "system", "content": "You are a critical quality assessor."},

{"role": "user", "content": QUALITY\_ASSESSMENT\_PROMPT.format(

original=original,

cleaned=cleaned

)}

],

temperature=0.3

)

result = json.loads(response.choices[0].message.content)

return result

except Exception as e:

print(f"Error in quality assessment: {e}")

return {"is\_acceptable": False, "total\_score": 0}

설계의도: 저는 LLM을 \*\*‘까다로운 품질 관리 전문가’\*\*로 설정해, 단순한 칭찬 대신 근거 중심의 비판적 평가를 끌어내도록 설계했습니다. LLM은 자동 평가에서 점수 관대/중앙값(7–8점) 쏠림이 나타나기 쉬워, 이를 깨기 위해 명확한 감점·가점 기준과 증거 제시를 의무화했습니다. 또한 데이터 품질을 사용자(구직자) 관점—‘쓸모(사용 적합성)’으로 보게 하여, 기술적 완성도만 아니라 실제 지원자가 이해·결정에 쓰기 좋은가를 최우선으로 평가합니다. 마지막으로, 점수로 끝내지 않고 구체 개선안을 요구해 지속적인 피드백 루프를 형성합니다.

4 파이프라인 지연이 검색 품질에 미치는 도미노 효과

현재 시스템에서 파이프라인과 검색 품질은 밀접하게 연관되어 있습니다. 6시간마다 실행되는 배치 처리가 실패하거나 지연되면, 그 영향은 검색 품질 저하로 이어집니다.

실제 발생 시나리오 분석

케이스 1: 배치 처리 타이밍 문제

배치 처리 직후에 등록된 채용공고는 다음 배치까지 대기 상태에 머물러야 합니다. 배치 처리 자체가 수 시간 소요될 경우, 상당한 지연이 발생할 수 있습니다. 이러한 지연은 특히 "긴급 채용"이나 "선착순 모집"과 같은 시간에 민감한 공고에서 문제가 됩니다.

케이스 2: 파이프라인 부분 실패 시

데이터 처리 중 일부만 성공할 경우 검색 결과가 불완전한 상태로 제공됩니다. 예를 들어, 사용자가 "Python 개발자"를 검색했을 때 최신 공고는 누락되고 오래된 공고만 노출되는 상황이 발생할 수 있습니다.

케이스 3: 정제 실패로 인한 품질 저하

HTML 태그나 JavaScript 코드가 제대로 제거되지 않을 경우, 이러한 노이즈가 그대로 임베딩 과정에 포함됩니다. 결과적으로 검색 의도와 무관한 결과가 노출될 가능성이 있습니다.

현재 프로세스

데이터 수집 → 전체 정제 → 임베딩 생성 → 인덱싱 (각 단계가 순차적으로 진행되며, 하나라도 실패하면 전체 롤백)

확인된 문제점

• All or Nothing 방식: 부분 성공이 불가능한 구조

• 순차 처리 구조: 각 단계가 완료되어야 다음 단계 진행 가능

• 실패 시 복구: 처음부터 전체 프로세스를 재시작해야 함

6.3 파이프라인 문제가 만드는 연쇄 반응

파이프라인 지연이나 실패는 다음과 같은 연쇄적인 문제를 발생시킬 수 있다.

데이터 신선도 관련 이슈

• 최신 공고가 검색 결과에 반영되지 않는 현상

• 구직자가 업데이트되지 않은 정보에 노출

• 마감된 공고가 계속 노출되는 문제

중복 데이터 가능성

• 재처리 과정에서 동일 데이터의 중복 인덱싱 위험

• 검색 결과에 동일 공고가 여러 번 노출될 가능성

• 사용자 혼란 초래

임베딩 품질 일관성 문제

• 일부 데이터는 정제되고 일부는 정제되지 않은 상태로 처리

• 벡터 공간의 일관성 문제 발생 가능

• 검색 관련성 저하 우려

아래페이지는 아키텍처입니다.

