Insurance Rebuttal Take Home

1) Overview

Build an end-to-end pipeline that ingests two restoration proposals for a **single residential job**—one from **JDR** (the contractor) and one from the **Insurance** provider—and produces a **marked-up, annotated proposal** highlighting agreement, discrepancies, missing items, and insurance-only "nuggets." The objective is to **maximize dollars recovered for JDR** by surfacing differences clearly and defensibly.

The output should closely replicate the look and feel of the provided reference markup (`ciridae_markup_proposal.pdf`).

---

2) Data & Reference Output

- **Folder with inputs & reference:** https://drive.google.com/drive/folders/1yTdL2F6fXPkW5yMhaixcWJiNg82C7zt3?usp=drive_link

    - `jdr_proposal.pdf` — JDR proposal (unannotated)
    - `insurance_proposal.pdf` — Insurance proposal
    - `ciridae_markup_proposal.pdf` — reference marked-up output (target style)

  Proposals originate from **Xactimate** and are typically exported to PDF.

---

3) Business Context

JDR is a Texas restoration firm working on **flooding, mold damage, and hail damage**. For each job, JDR and the insurer produce detailed **Xactimate proposals** listing line items (materials, labor, quantities, units, pricing). JDR's proposal is typically **50%–3×** larger in cost than the insurer's. Historically, a JDR leader spent **~30%** of time printing and **color-highlighting** proposals to prepare for negotiations. Your automation removes this bottleneck.

---

4) Definitions & Rules

Color semantics used in the marked-up output:

- **Green — Exact Match**: A JDR line item matches an Insurance line item **in the same room**; description is essentially the same (semantic match), and all key metadata (e.g., quantity, units, unit price) are within **±2%**.
- **Orange — Partial Match**: Same underlying task (semantic match) **but** one or more key metadata fields differ beyond **±2%**.
- **Blue — JDR-Only**: Present in JDR but **absent** in Insurance.
- **"Nuggets" — Insurance-Only**: Present in Insurance but **absent** in JDR. These should be surfaced clearly; JDR can often add them directly since the insurer already agreed.

Room constraint: Matches are valid **only within a room**. Identical descriptions attached to different rooms **do not** match. Room names may vary between documents (e.g., Insurance "Kitchen Area" ↔ JDR "Kitchen" + "Breakfast Area"; Insurance "Bedroom 1" ↔ JDR "Master Bedroom"). Your system must robustly handle splits/merges and naming variation.

---

5) Requirements

1. **Inputs:** Two proposal documents (JDR + Insurance). Assume PDFs; handle variations you encounter.
2. **Output:** A **marked-up proposal** that mirrors `ciridae_markup_proposal.pdf`, including:
    - Green/Orange/Blue highlights per line item on the **JDR proposal** pages.
    - **Per-line notes** explaining the rationale for each classification.
    - **Page-level callouts** for Insurance-only **nuggets**.
3. **Tolerance:** Exact matches require metadata agreement within **±2%**.
4. **Rooms:** Robust mapping across naming differences and room splits/merges; only compare items within mapped rooms.
5. **End-to-end automation:** One command/process should run the full pipeline from the two inputs to the final annotated output.
6. **Gateway usage:** Use **Ciridae's LLM Gateway** for any LLM calls.
7. **Frontend:** Minimal, customer facing UI for using the product.

---

6) Frontend

A minimal web UI that allows uploading the two documents and viewing/downloading the resulting marked-up proposal.

If you have extra time, feel free to add any features that you think a user would want or need. Note that extra frontend features are not required.

---

7) Deliverables

- A working solution that runs **end-to-end** on the provided inputs and produces the annotated proposal along with a customer facing frontend for using the product.
    - The solution should be aligned with our tech stack, outlined below.
- A link to a GitHub project with a concise **README** describing setup, how to run, and any assumptions.

---

8) Tech Stack

We'd like to see this built with our standard stack. Please use these tools or explain your substitutions:

- **Backend:** Python with uv, FastAPI
- **Database:** Postgres with SQLAlchemy (2.x)
- **Frontend:** TypeScript, React, Vite
- **State/Data:** React Query
  Styling/UI: TailwindCSS, shadcn/ui
- **Validation:** zod (frontend) / Pydantic (backend)

LLM Calls: See LLM Gateway documentation at bottom of document

---

9) Ciridae LLM Gateway

Use **Ciridae's LLM Gateway** for any LLM calls. See documentation below.

LLM Gateway Documentation

API Key:
6aogo3d2Vv_f7_M9NXBxRGhvYTY3aSoIaiYq9C6Fli4

Overview

The LLM Gateway is a lightweight API that allows you to interact with various LLM providers through an Open AI compatible API interface.

Quick Start

Python

```python
import os
from openai import OpenAI

client = OpenAI(
    api_key=os.getenv("GATEWAY_API_KEY"),
    base_url="https://llm-gateway-5q22j.ondigitalocean.app",
)

# Open AI Chat Completion
response = client.chat.completions.create(
    model="openai/gpt-4o-2024-08-06",
    messages=[{"role": "user", "content": "Hello, world!"}],
)

# Anthropic Chat Completion
response = client.chat.completions.create(
    model="anthropic/claude-3-5-sonnet-20240620",
    messages=[{"role": "user", "content": "Hello, world!"}],
)
```

Supported Models

All supported models can be found here

Model Routers

The Gateway has support for a number of "model routers". These are essentially load balancers or pre-defined fallbacks for a set of models.

For example, "claude-3-7-sonnet" is a router that is rate-limit aware and spreads requests between the Anthropic API, AWS Bedrock and Google Vertex. It also will fall back to another deployment of Sonnet if the first API fails.

"fast-production" and "strong-production" are routers that have a pre-defined fallback chain of models based on our internal ratings of the best overall and best cheap models.

The Gateway has support for other Routers based on budgets, latency and others.

Image Support

How to pass in images to the Gateway:

Python

```
import base64

def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode("utf-8")

image_path = "test_image.jpeg"
base64_image = encode_image(image_path)
messages = [
    {
        "content": [
            {"type": "text", "text": "What is in this image?"},
            {
                "type": "image_url",
                "image_url": {"url": "" + base64_image},
# providing an externally hosted image as a url is also supported
            },
        ],
        "role": "user",
    }
]

response = client.chat.completions.create(
    model="claude-3-5-sonnet",
    messages=messages,
)
```

Structured Outputs

Python

```python
from pydantic import BaseModel

class CalendarEvent(BaseModel):
    name: str
    date: str
    participants: list[str]

completion = client.chat.completions.parse(
    model="claude-3-5-sonnet",
    messages=[
        {"role": "system", "content": "Extract the event information."},
        {
            "role": "user",
            "content": "Alice and Bob are going to a science fair on Friday.",
        },
    ],
    response_format=CalendarEvent,
)

event = completion.choices[0].message.parsed
```