**🚀 Cortex AI Feature Generation & GenAI Integration Plan**

**🧱 1. Structured Feature Experiment Using Cortex AI**

Use Cortex AI's AutoML & feature engineering pipeline for:

* **Auto-discovery of new variables**
* **Interaction detection** (e.g., peril × region, KRI score × delay in reporting)

**🔧 Steps:**

1. **Dataset Prep:**
   * Remove liability-related records/features
   * Include claim\_id, ultimate\_severity, existing features, and doc references
2. **Upload to Cortex:**
   * Upload structured dataset to Snowflake with Cortex integration
3. **Run Feature Synthesis:**
   * Use Cortex “Auto Feature Discovery”:
     + Time-based aggregations (e.g., historical claim patterns)
     + Claim-level behavioral features (e.g., frequency of interactions, updates)
     + External enrichments (weather, location)
4. **Evaluate Feature Importance:**
   * Let Cortex identify top features by SHAP importance
   * Prioritize features with non-linear interactions with ultimate\_severity

**📄 2. GenAI for Loss Description & Document Feature Engineering**

**✨ Objective:**

Transform free-text fields & documents into structured severity indicators using GenAI in Cortex AI.

**🔧 Strategy:**

Use GenAI text processing in Cortex to create **new NLP-based features**:

**📌 Prompt-Based GenAI Features**

Use *prompt templates* in Cortex to enrich data fields.

**📄 Example Prompt 1:**

**“Summarize the likely severity of this claim based on the description: ‘{loss\_description}’. Focus on extent of damage, risk indicators, and recovery time.”**

| **Output** | **Use as Feature** |
| --- | --- |
| High severity due to total property loss and fire spread | Severity keyword flag |
| Moderate loss with expected repairs within 2 weeks | Recovery duration |
| Potential for escalation due to legal factors | Escalation risk |

**📄 Example Prompt 2:**

**“List all damage types mentioned in the following claim notes: ‘{claim\_notes}’. Return in structured format.”**

| **Output** | **Use as Feature** |
| --- | --- |
| [“Fire”, “Roof Collapse”, “Smoke Damage”] | Binary flags for each type |
| [] | Indicates vague/empty notes |

**📄 Example Prompt 3:**

**“Does this document suggest that the loss is among the top 5% most severe historical claims? Answer Yes/No and provide reasoning.”**

| **Output** | **Use as Feature** |
| --- | --- |
| Yes (Extensive structural damage + commercial interruption > 6 months) | GenAI Top 5% Indicator |

**🧪 3. Build Embeddings for Semantic Similarity**

Use **SentenceTransformer** or **OpenAI embeddings in Cortex** to:

* Vectorize full loss descriptions, claim notes, and adjuster summaries
* Apply **clustering** or **t-SNE** to explore claim types
* Feed embeddings (or their cluster labels) into LightGBM

**🔁 4. Integrate with LightGBM**

After generating these features:

* Merge Cortex GenAI outputs with structured data
* Retrain LightGBM model (optimize for **recall**)
* Compare performance with previous model

**✅ Deliverables**

Here’s what you can prepare or ask Cortex to produce:

| **Deliverable** | **Purpose** |
| --- | --- |
| 🔍 **Feature importance list (SHAP)** | Understand top new features |
| 🧠 **GenAI-based labels** | Structured fields from documents |
| 📊 **Severity flags / escalation signals** | Improve early intervention |
| 🔢 **Embedded clusters** | Segment high-severity claim patterns |
| 📈 **Updated LightGBM performance** | Recall, Precision, AUC comparison |

**1. Prompt Template Set**

These are prompts (with expected JSON outputs) that you can use to generate structured features from free‑text descriptions / loss notes / documents. You’ll call them via snowflake.cortex.complete(...), or other Cortex LLM functions.

| **Prompt Name** | **Purpose / Feature(s) Extracted** | **Prompt Template** | **Output Schema (JSON)** | **Example Use** |
| --- | --- | --- | --- | --- |
| **Damage Types Extractor** | Flags for types of damage mentioned; counts etc. | ``` |  |  |
| You are an expert claims adjuster. Given the following loss description, list all damage types mentioned (e.g. fire, water, smoke, structural collapse, hail, theft, vandalism), and estimate severity level for each damage type from “minor / moderate / major / catastrophic”. Return JSON. |  |  |  |  |

Loss description: “{loss\_description}”  
|json  
{  
"damage\_types": [  
{ "type": "Fire", "severity": "major" },  
{ "type": "Smoke", "severity": "moderate" }  
],  
"primary\_damage": "Fire",  
"highest\_damage\_severity": "major"  
}  
| Use flag features like `has\_fire\_damage`, `has\_smoke\_damage`, also severity mapping: `primary\_damage\_severity` etc. | | \*\*Escalation & Litigation Risk\*\* | Identify whether description suggests likely escalation risk, e.g. legal, reinstatement, delay, dispute |  
Given the following loss notes, determine if the claim is likely to escalate (yes/no). Identify cues (keywords or phrases) that suggest escalation, such as “legal”, “dispute”, “ongoing”, “multiple parties”, “suspected fraud”, “injury” etc.

Return JSON with keys: escalate\_flag, escalation\_reasons[].

Loss notes: “{loss\_notes}”  
|json  
{  
"escalate\_flag": true,  
"escalation\_reasons": ["legal dispute", "multiple parties involved", "severe injury mentioned"]  
}  
| Use feature `escalate\_flag` (binary), and one‐hot or count of reasons; maybe severity of escalation risk. | | \*\*Estimated Recovery Time\*\* | Estimate how long the repairs / claim resolution may take as mentioned / implied in description => short vs long delay indicator |  
You are an insurance severity estimator. Based on this loss description, estimate the expected recovery / repair time in days, or if not explicitly given, approximate based on context (words like “months”, “weeks”, “long term disruption”).

Return JSON: { "estimated\_repair\_days": integer or null, "repair\_time\_category": "short" / "medium" / "long" }

Loss description: “{loss\_description}”  
|json  
{  
"estimated\_repair\_days": 120,  
"repair\_time\_category": "long"  
}

**SQL Code:**

| \*\*Severity Signal\*\* | Overall severity signal from text: is this likely among top 5% by severity? Yes/no + reason | ```

Given this loss description, would you rate this claim among our top 5% of severe claims historically (Yes/No)? Provide the top 3 reasons from the description for that assessment.

Loss description: “{loss\_description}”

``` | ```json

{

"text\_severity\_flag": true,

"text\_severity\_reasons": ["structural collapse", "fire spread", "business interruption"]

}

``` | Use `text\_severity\_flag` as a label / feature; use count of reasons etc. |

| \*\*Entity & Quantitative Extraction\*\* | Extract any numbers, measurements, amounts, injuries, extent etc. | ```

Read the description below and extract all measurable entities: number of persons injured, amount of damage (if stated), area impacted (e.g. square meters), number of properties/flats involved.

Return JSON, include keys like injured\_count, damage\_amount, area\_impacted, number\_of\_units. If not mentioned, null.

Loss description: “{loss\_description}”

``` | ```json

{

"injured\_count": 2,

"damage\_amount": null,

"area\_impacted\_sqm": 150,

"number\_of\_units": 3

}

``` | Useful numeric features. |

---

You may also combine some prompts (for efficiency) or run them in batch.

### Prompt Template Runner / Parameterization

You can parameterize these using \*\*Cortex Prompt Template Runner\*\*. According to Snowflake documentation, you can build template files (YAML/SQL configs) where prompt variables map to table columns, literal vars, etc. :contentReference[oaicite:0]{index=0}

For example:

```yaml

prompt:

name: "Damage\_Types\_Extractor\_v1"

version: "1.0"

messages:

- role: system

content: |

You are an expert claims adjuster. Given the following loss description, list all damage types mentioned ...

- role: user

content: |

Loss description: "{loss\_description}"

column\_variables:

loss\_description: "loss\_description\_col"

literal\_variables:

# nothing or maybe some instruction modifiers

operators:

model: "claude-4-sonnet"

max\_tokens: 200

temperature: 0.3

origin\_table: "claims.loss\_notes"

**2. Example Integration Pipeline**

Here’s a simplified end‑to‑end pipeline you can use:

-- 1. Create a view/table with the existing structured data + loss descriptions

CREATE OR REPLACE VIEW claims.for\_model\_input AS

SELECT

claim\_id,

ultimate\_severity, -- target

/\* existing features: day\_number, days\_between\_date\_of\_loss\_and\_reporting, fe\_count\_of\_involved\_in\_claim, d\_cumu\_total\_kri\_score, peril, subperil, etc. \*/

day\_number,

days\_between\_date\_of\_loss\_and\_reporting,

fe\_count\_of\_involved\_in\_claim,

d\_cumu\_total\_kri\_score,

peril\_group,

subperil\_group,

loss\_description,

loss\_notes

FROM claims.claims\_data

WHERE liability\_flag = FALSE; -- remove liability cases if applying removal

-- 2. Use Cortex COMPLETE + PROMPT (or classify\_text etc.) to extract new features

CREATE OR REPLACE TABLE claims.text\_features AS

SELECT

claim\_id,

(snowflake.cortex.complete(

'claude-4-sonnet',

PROMPT(

'You are an expert claims adjuster. Given the description: "{0}", list all damage types and estimate severity level for each damage type in JSON. Loss description: ',

loss\_description

)

)::VARIANT) AS damage\_types\_info,

(snowflake.cortex.complete(

'claude-4-sonnet',

PROMPT(

'Based on the following loss notes: "{0}", determine whether this claim is likely to escalate. Return JSON with escalate\_flag and escalation\_reasons[]. Loss notes: ',

loss\_notes

)

)::VARIANT) AS escalation\_info

FROM claims.for\_model\_input

;

-- 3. Parse JSON outputs to structured columns

CREATE OR REPLACE TABLE claims.text\_features\_parsed AS

SELECT

claim\_id,

damage\_types\_info:"primary\_damage"::STRING AS primary\_damage,

damage\_types\_info:"highest\_damage\_severity"::STRING AS highest\_damage\_severity,

-- for damage types, maybe flatten into flags: has\_fire = 1 if damage\_types\_info array contains type “Fire”

CASE WHEN ARRAY\_CONTAINS(damage\_types\_info:"damage\_types"[\*] :> "type", 'Fire') THEN 1 ELSE 0 END AS has\_fire\_damage,

CASE WHEN ARRAY\_CONTAINS(damage\_types\_info:"damage\_types"[\*] :> "type", 'Water') THEN 1 ELSE 0 END AS has\_water\_damage,

escalation\_info:"escalate\_flag"::BOOLEAN AS escalate\_flag,

-- maybe number of escalation reasons

ARRAY\_SIZE(escalation\_info:"escalation\_reasons")::INT AS escalation\_reason\_count

FROM claims.text\_features

;

-- 4. Join back into model input and export for training

CREATE OR REPLACE VIEW claims.model\_augmented\_input AS

SELECT

m.\*,

tfp.primary\_damage,

tfp.highest\_damage\_severity,

tfp.has\_fire\_damage,

tfp.has\_water\_damage,

tfp.escalate\_flag,

tfp.escalation\_reason\_count

FROM claims.for\_model\_input m

LEFT JOIN claims.text\_features\_parsed tfp

ON m.claim\_id = tfp.claim\_id

;

And then in Python (Snowflake Snowpark or external notebook):

import snowflake.snowpark as sf

from lightgbm import LGBMClassifier

import pandas as pd

# setup session

session = sf.Session.builder.configs({...}).create()

# Load data

df = session.table("claims.model\_augmented\_input").to\_pandas()

# Feature engineering: convert categorical text fields (primary\_damage, highest\_damage\_severity) to numerical or one‑hot

df = pd.get\_dummies(df, columns=["primary\_damage","highest\_damage\_severity"], dummy\_na=True)

# Label: top 5% severity

threshold = df['ultimate\_severity'].quantile(0.95)

df['severe\_top5'] = (df['ultimate\_severity'] >= threshold).astype(int)

# Split train/test

from sklearn.model\_selection import train\_test\_split

X = df.drop(columns=['claim\_id','ultimate\_severity','severe\_top5'])

y = df['severe\_top5']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, test\_size=0.2)

# Train LightGBM

model = LGBMClassifier(

objective='binary',

boosting\_type='gbdt',

is\_unbalance=True,

# set parameters to favour recall

metric='auc',

# you might tweak for recall via class\_weight or scale\_pos\_weight etc.

)

model.fit(X\_train, y\_train)

# Evaluate recall

from sklearn.metrics import classification\_report

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred, digits=4))