FRIS — STEP 2: FEATURE INFERENCE CONTRACT (FROZEN)

Status: FINAL — this is the single source of truth for raw → engineered inference

This document defines the exact behavior, boundaries, and guarantees of the **feature inference layer**. No model logic. No explainability. No shortcuts.

PURPOSE (ONE SENTENCE)

Convert **one raw transaction dict** into the **exact engineered feature DataFrame** used during training.

Nothing more.

FILE OWNERSHIP

File: `src/feature_inference.py`

Why a separate file: - `features.py` defines transformations - `feature_inference.py` applies them in inference context - `models.py` must never know how features are created

This file is the **only legal entry point** for inference feature creation.

PUBLIC API

```
def prepare_features(raw_input: dict) -> pd.DataFrame
```

This function is called by: - `models.predict()` - `models.predict_proba()` - `explain.explain_transaction()` - sanity / verification scripts

---

INPUT CONTRACT

raw_input

- Type: `dict`
- Represents **one transaction only**
- Must contain the raw columns expected by training

Minimum required keys (current system): - `Time` - `Amount`

Notes: - No batch support - No list of dicts - No default filling here

---

OUTPUT CONTRACT

- Type: `pd.DataFrame`
- Shape: `(1, N)`
- Columns: **exactly the same** as `feature_pipeline(..., fit=True)` produced during training
- Order: not required, but column set must match

This DataFrame is **model-ready** but contains **no model outputs**.

INTERNAL EXECUTION STEPS (MANDATORY ORDER)

1. **Convert raw dict → single-row DataFrame**

```
df_raw = pd.DataFrame([raw_input])
```

2. **Load preprocessors from disk**

3. scaler
4. encoders
5. PCA

6. must be the exact objects fit during training

7. **Call feature pipeline in inference mode**

```
df_eng, _ = feature_pipeline(
    df_raw,
    fit=False,
    preprocessors=preprocessors
)
```

8. **Column contract assertion (HARD FAIL)**

9. Load `training_feature_columns.json`
10. Compare against `df_eng.columns`

11. Missing or extra columns → raise error

12. **Return engineered DataFrame**

WHAT prepare_features MUST NOT DO

- No model loading
- No model inference
- No probability computation
- No anomaly scoring
- No latent extraction
- No clustering
- No SHAP
- No thresholding
- No try/except swallowing errors

If something breaks here, the system **must crash loudly**.

WHAT prepare_features GUARANTEES

If this function returns successfully:

- Feature engineering is identical to training
- All downstream models can safely consume the output
- Explainability will operate on the same features as prediction

This is the **single choke point** for correctness.

RELATIONSHIP TO OTHER FILES

features.py

- Defines how features are created
- Stateless, reusable, inference-agnostic

feature_inference.py (this file)

- Owns raw → engineered transition
- Loads state (preprocessors)
- Enforces training contract

models.py

- Consumes engineered features
- Computes model signals
- Performs stacking and labeling

explain.py

- Explains XGBoost using engineered features
- Must never re-engineer features

INFERENCE PIPELINE (FINAL, CANONICAL)

```
raw_input (dict)
  ↓
prepare_features()
  ↓
engineered_df
  ↓
models.py (base models + stacker)
  ↓
score + label
```

Explainability uses the **same engineered_df**.

STEP 2 OUTPUT (WHAT IS NOW FROZEN)

- Feature inference responsibility is isolated
- Ownership boundaries are clean
- Raw → engineered path is unambiguous
- No file overlaps responsibility

READY FOR STEP 3

Next step: **Refactor** `models.py` **to consume** `prepare_features()` **output and delete** `validate_input()`**.**