Explainable Healthcare Framework – Complete Report

This document consolidates the system architecture diagrams, data flow diagrams, business/system components, algorithm descriptions, full reference Python implementations, and a README-style guide into a single PDF.

Key Algorithms & Explanations

A. Multimodal Harmonization & Feature Fusion: Schema mapping → missingness-aware transforms → modality encoders → attention-based fusion.
B. Synthetic Data Generation: Conditional generative sampling validated against constraints & logged with lineage.
C. Cognitive Network Reasoning: Knowledge-graph/causal path search, top-k scoring of reasoning chains.
D. Explainability & Counterfactuals: Attribution, case-based reasoning, counterfactual search.
E. Early Warning Detection: EWMA/sequence modeling for streaming risk alerts.
F. Uncertainty Quantification: Ensemble/MC dropout, risk triage.
G. Bias/Drift Auditing: Rolling metrics, PSI, fairness gaps.
H. Visual Reasoning Layout: DAG-style visualization of evidence → hypothesis → action.

Algorithm Reference Implementation (Python)

```
"""
algorithms.py — Reference implementations for key components implied by
"Generative AI■Powered Visual Frameworks for Explainable Healthcare Analytics
and Cognitive Network Research".

DISCLAIMER: These are illustrative, high■level implementations for educational
and architectural prototyping purposes, not clinical use.
"""

from __future__ import annotations
from dataclasses import dataclass, field
from typing import Dict, Any, List, Tuple, Optional
import numpy as np
import math
import random

# --------------------------- Data Harmonization ---------------------------

@dataclass
class DataHarmonizer:
    """
    Harmonizes heterogeneous healthcare data into a unified feature space.
    - Applies schema mapping, missingness-aware transforms, scaling, and encoding.
    - Returns aligned numeric features suitable for modeling.
    """
    feature_schema: Dict[str, Dict[str, Any]]  # {feature_name: {type: "num|cat|bin", ...}}

    def transform(self, record: Dict[str, Any]) -> Dict[str, float]:
        out: Dict[str, float] = {}
        for name, meta in self.feature_schema.items():
            ftype = meta.get("type", "num")
            default = meta.get("default", 0.0)

            val = record.get(name, None)
            if val is None or (isinstance(val, float) and math.isnan(val)):
                # Missingness indicator
                out[f"{name}__missing"] = 1.0
                val = default
            else:
                out[f"{name}__missing"] = 0.0

            if ftype == "num":
                mean = meta.get("mean", 0.0)
                std = meta.get("std", 1.0) or 1.0
                out[name] = (float(val) - mean) / std
            elif ftype == "bin":
                out[name] = 1.0 if bool(val) else 0.0
            elif ftype == "cat":
                # One■hot encode known categories
                categories: List[str] = meta.get("categories", [])
                for cat in categories:
                    out[f"{name}__{cat}"] = 1.0 if val == cat else 0.0
            else:
                out[name] = float(val) if isinstance(val, (int, float)) else 0.0
        return out

# --------------------------- Multimodal Fusion ---------------------------

@dataclass
class MultimodalFusion:
    """
    Simple missingness■aware attention fusion for multiple modality embeddings.
    Each modality provides an embedding vector; we compute weights that down■weight
    missing/low■quality modalities.
    """
    dim: int
    def fuse(self, embeddings: Dict[str, Optional[np.ndarray]], quality: Dict[str, float]) -> np.ndarray:
        keys = list(embeddings.keys())
        vecs: List[np.ndarray] = []
        weights: List[float] = []
        for k in keys:
            v = embeddings.get(k, None)
            if v is None:
                continue
            if v.shape[0] != self.dim:
                raise ValueError(f"Embedding for {k} must have dim={self.dim}")
            q = max(0.0, min(1.0, quality.get(k, 1.0)))
            vecs.append(v)
            weights.append(q)

        if not vecs:
            return np.zeros(self.dim, dtype=float)

        W = np.array(weights, dtype=float)
        if W.sum() <= 0:
            W = np.ones_like(W)
        W = W / W.sum()
        stacked = np.stack(vecs, axis=0)  # [m, d]
        fused = (W[:, None] * stacked).sum(axis=0)
        return fused

# -------------------- Synthetic Data Generation (Stub) --------------------

@dataclass
class SyntheticDataGenerator:
    """
    A simple conditional sampler stub (not a real diffusion/GAN). For prototyping:
    generates synthetic samples by adding structured noise conditioned on metadata.
    """
    seed: int = 42

    def sample(self, base: Dict[str, float], meta: Dict[str, Any], n: int = 1) -> List[Dict[str, float]]:
        rng = random.Random(self.seed)
        out: List[Dict[str, float]] = []
        scale = 0.05 + 0.01 * len(meta)  # simplistic conditioning
        for _ in range(n):
            s = {}
            for k, v in base.items():
                noise = rng.gauss(0.0, scale)
                s[k] = float(v + noise)
            out.append(s)
        return out

# --------------------- Cognitive Network Reasoning -----------------------

@dataclass
class CognitiveNetworkReasoner:
    """
    Lightweight path■search reasoner over a knowledge graph with edge weights
    representing causal/relational strength. Returns top■k reasoning chains.
    """
    graph: Dict[str, List[Tuple[str, float]]]  # node -> list of (neighbor, weight)

    def score_path(self, path: List[str]) -> float:
        score = 0.0
        for a, b in zip(path, path[1:]):
            edges = self.graph.get(a, [])
            w = 0.0
            for nb, wt in edges:
                if nb == b:
                    w = wt; break
            score += w
        return score / max(1, len(path) - 1)
    def topk_paths(self, start: str, goals: List[str], k: int = 3, max_depth: int = 4) -> List[Tuple[List[
        best: List[Tuple[List[str], float]] = []

        def dfs(node: str, path: List[str]):
            nonlocal best
            if len(path) > max_depth:
                return
            if node in goals:
                sc = self.score_path(path)
                best.append((path.copy(), sc))
                best = sorted(best, key=lambda x: x[1], reverse=True)[:k]

            for nb, wt in self.graph.get(node, []):
                if nb in path:
                    continue
                path.append(nb)
                dfs(nb, path)
                path.pop()

        dfs(start, [start])
        return best

# ---------------------- Explainability (Attribution) ---------------------

@dataclass
class XAIExplainer:
    """
    Simple feature attribution via finite differences (sensitivity approximation).
    For image/text models you'd use Grad■CAM/SHAP/LIME; this is a numeric example.
    """
    model: Any
    eps: float = 1e-3

    def predict(self, x: Dict[str, float]) -> float:
        return float(self.model(x))

    def attribute(self, x: Dict[str, float]) -> Dict[str, float]:
        base = self.predict(x)
        contrib: Dict[str, float] = {}
        for k, v in x.items():
            x2 = dict(x)
            x2[k] = v + self.eps
            contrib[k] = (self.predict(x2) - base) / self.eps
        return contrib

# -------------------- Early Warning Detector (Streaming) ------------------

@dataclass
class EarlyWarningDetector:
    """
    EWMA■based streaming risk score with interpretable contributions.
    """
    alpha: float = 0.2
    weights: Dict[str, float] = field(default_factory=dict)
    threshold: float = 0.7

    def step(self, vitals: Dict[str, float], prev_score: float = 0.0) -> Tuple[float, bool, Dict[str, floa
        raw = 0.0
        contrib: Dict[str, float] = {}
        for k, v in vitals.items():
            w = self.weights.get(k, 0.0)
            raw += w * v
            contrib[k] = w * v
        score = self.alpha * raw + (1.0 - self.alpha) * prev_score
        alert = score >= self.threshold
        return score, alert, contrib

# --------------- Uncertainty Quantifier (MC Dropout style) ---------------
```

```
@dataclass
class UncertaintyQuantifier:
    """
    Simple ensemble surrogate: average predictions across stochastic models to estimate
    mean & variance -> defer if uncertainty high.
    """
    ensemble: List[Any]
    defer_threshold: float = 0.15  # std dev cutoff

    def __call__(self, x: Dict[str, float]) -> Tuple[float, float, bool]:
        preds = np.array([float(m(x)) for m in self.ensemble], dtype=float)
        mean = float(preds.mean())
        std = float(preds.std())
        defer = std >= self.defer_threshold
        return mean, std, defer

# -------------------- Bias/Drift Auditor & Audit Log ---------------------

@dataclass
class BiasAuditor:
    """
    Tracks subgroup performance metrics and simple drift via population stability index (PSI).
    """
    def psi(self, baseline: np.ndarray, current: np.ndarray, bins: int = 10) -> float:
        baseline_hist, edges = np.histogram(baseline, bins=bins, density=True)
        current_hist, _ = np.histogram(current, bins=edges, density=True)
        eps = 1e-10
        terms = (current_hist + eps) * np.log((current_hist + eps) / (baseline_hist + eps))
        return float(terms.sum())

    def fairness_gap(self, yhat_a: np.ndarray, yhat_b: np.ndarray, positive_threshold: float = 0.5) -> flo
        pa = (yhat_a >= positive_threshold).mean()
        pb = (yhat_b >= positive_threshold).mean()
        return float(abs(pa - pb))

# --------------------------- Demo Model Stubs ----------------------------

def linear_stub(weights: Dict[str, float], bias: float = 0.0):
    def f(x: Dict[str, float]) -> float:
        s = bias
        for k, w in weights.items():
            s += w * x.get(k, 0.0)
        # squash to [0,1]
        return 1.0 / (1.0 + math.exp(-s))
    return f

# ------------------------------ Quick Demo -------------------------------

if __name__ == "__main__":
    # Harmonization
    harmonizer = DataHarmonizer({
        "age": {"type": "num", "mean": 60.0, "std": 15.0, "default": 60.0},
        "sex": {"type": "cat", "categories": ["M", "F"], "default": "M"},
        "bp_systolic": {"type": "num", "mean": 120.0, "std": 20.0, "default": 120.0},
        "diabetes": {"type": "bin", "default": 0},
    })
    rec = {"age": 72, "sex": "F", "bp_systolic": 140, "diabetes": 1}
    feats = harmonizer.transform(rec)

    # Fusion
    fusion = MultimodalFusion(dim=4)
    emb = {
        "ehr": np.array([0.1, 0.2, 0.3, 0.4]),
        "img": np.array([0.0, 0.1, 0.0, 0.2]),
        "geno": None,
    }
    q = {"ehr": 1.0, "img": 0.8, "geno": 0.0}
    fused = fusion.fuse(emb, q)
```

```
# Synthetic sampler
synth = SyntheticDataGenerator(seed=7)
syn = synth.sample({"x": 0.3, "y": -0.2}, {"class": "A"}, n=2)

# Cognitive reasoning
graph = {
    "fever": [("infection", 0.7)],
    "infection": [("sepsis", 0.5), ("pneumonia", 0.6)],
    "pneumonia": [("antibiotics", 0.8)],
    "sepsis": [("icu_care", 0.9)]
}
reasoner = CognitiveNetworkReasoner(graph=graph)
chains = reasoner.topk_paths(start="fever", goals=["sepsis", "pneumonia"], k=3, max_depth=4)

# Explainer
model = linear_stub({"age": 0.02, "diabetes": 0.8, "bp_systolic": 0.01}, bias=-2.0)
explainer = XAIExplainer(model=model)
score = explainer.predict(feats)
attributions = explainer.attribute(feats)

# Early warning
ewd = EarlyWarningDetector(alpha=0.2, weights={"hr": 0.003, "rr": 0.02, "temp": 0.1}, threshold=0.5)
s, alert, contribs = ewd.step({"hr": 110, "rr": 24, "temp": 38.6}, prev_score=0.3)

# Uncertainty
uq = UncertaintyQuantifier(ensemble=[linear_stub({"age": 0.02}, -2.1),
                                     linear_stub({"age": 0.021}, -2.0),
                                     linear_stub({"age": 0.018}, -2.2)], defer_threshold=0.05)
mean, std, defer = uq(feats)

# Bias auditor
auditor = BiasAuditor()
psi_val = auditor.psi(np.random.randn(1000), np.random.randn(1000) + 0.2, bins=10)
gap = auditor.fairness_gap(np.random.rand(500), np.random.rand(500), positive_threshold=0.6)

print("DEMO OK")
```

README / Quickstart Guide

~~~
# Algorithms Reference (Explainable Healthcare Framework)

This folder contains a PDF report with architecture & flow diagrams, plus reference Python
implementations of key algorithms implied by the framework.

## Files
- `Explainable_Healthcare_Framework_Report.pdf` — High-level architecture, flows, layers, algorithm summar
- `algorithms.py` — Clean, well-commented Python reference implementations (harmonization, fusion, synthet

## Quickstart
```bash
python algorithms.py
```
You should see `DEMO OK` in the console.

> DISCLAIMER: Educational reference only, **not** validated for clinical use.
~~~