AI Model Welfare & Ethical AI Principles

AI Model Welfare and Ethical AI Principles focus on ensuring that AI systems are developed, deployed, and managed in a manner that aligns with ethical standards and regulatory requirements. This approach promotes trust, compliance, and long-term sustainability.

This notebook demonstrates a practical, end-to-end workflow that embeds model-welfare considerations throughout the AI lifecycle and aligns the work with the most relevant U.S. regulatory guidance (NIST AI RMF v1 draft 2023 & FINRA Notice 24-09 2024). It includes code and helper functions for each phase, from design and development to testing, deployment, monitoring, and retirement, along with key formulas and methods.

Goal:

To ensure that AI systems are developed, deployed, and managed in a manner that aligns with ethical standards and regulatory requirements, promoting trust, compliance, and long-term sustainability by treating AI systems as responsible actors rather than black-box utilities.

Why “Model Welfare”?

Trust, compliance, and long-term sustainability depend on treating AI systems as responsible actors rather than black-box utilities. By focusing on model welfare, we can build AI systems that are transparent, accountable, and ethical.

Key Skills and Deliverables:

- **Model-Card Sample**: Ability to document Large Language Models (LLMs) per industry standards.
- **Risk Register + Mitigation Plan**: Structured risk-management workflow.
- **Explainability Hands-on xAI Expertise**: Expertise in explainable AI (xAI) techniques.
- **Adversarial Test Report**: tables and plots demonstrating robustness testing competence.

The sections below map each ethical pillar to concrete engineering artifacts you can generate, version, and audit.

1. Ethical Pillars

Beneficence & Non-Maleficence

- **Robustness** – Predictable behavior under distribution shift.
- **Safety testing** – Adversarial & stress-test suites.

Fairness & Justice

- **Bias monitoring** – Ongoing disparity audits.
- **Inclusive datasets** – Representative sampling.

Transparency & Explainability

- **Model cards / datasheets** – Structured documentation.
- **Interpretability tools** – SHAP, LIME, counterfactuals.

Accountability & Governance

- **Version control & audit trails** – Git + DVC.
- **Human-in-the-loop policies** – Escalation thresholds.

Privacy & Data Protection

- **Differential privacy** – Noise injection during training.
- **Data minimization** – Retain only essential records.

Sustainability & Environmental Impact

- **Efficient architectures** – Distillation, pruning.
- **Energy-aware scheduling** – Renewable-powered clusters.

Human Dignity & Autonomy

- **Consent mechanisms** – UI notices for AI-generated output.
- **No deceptive impersonation** – Clear disclosure.

2. Regulatory Landscape:

Regulatory Guidelines for LLMs in the United States

Guideline	Issuer

NIST AI Risk Management Framework (RMF) – Draft v1.0 (2023)	National Institute of Standards and Technology	Risk-b
FINRA Regulatory Notice 24-09 (2024)	Financial Industry Regulatory Authority	Model

Mapping the Guidelines onto the Lifecycle

Lifecycle Phase	NIST RMF Touchpoints	
Design & Data	*Governance* – define purpose, risk appetite; *Data Quality* – provenance, bias checks.	*Rec*
Training & Testing	*Model Testing* – robustness, adversarial evaluation; *Impact Assessment* – hallucination mitigation.	*Mod*
Deployment	*Deployment Controls* – explainability APIs, user notices.	*Inve*
Post-Deployment	*Monitoring* – continuous performance/fairness logs; *Risk Management* – incident response plan.	*Ong*
Retirement	*Decommissioning* – archive artifacts, delete unnecessary data.	*Rete*

3. Setup & Helper Functions

```
!pip install pandas numpy scikit-learn shap tqdm matplotlib seaborn dvc
```

```
import json
import os
import datetime as dt
from pathlib import Path

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import shap

# ------------------------------------------------------------------
# Helper: Save a model-card (JSON) with versioning info
def save_model_card(card_dict, out_dir="model_cards"):
    """Write a model-card JSON file with a timestamped filename."""
    os.makedirs(out_dir, exist_ok=True)
    ts = dt.datetime.utcnow().strftime("%Y%m%dT%H%M%SZ")
    fname = f"{card_dict['model_name']}_card_{ts}.json"
    path = Path(out_dir) / fname
    with open(path, "w", encoding="utf-8") as f:
        json.dump(card_dict, f, indent=2, ensure_ascii=False)
    print(f"Model card saved → {path}")
    return path

# Helper: Simple bias audit (demographic parity) ------------------------------
def bias_audit(df, label_col, protected_attr, pred_col):
```

```
    """
    Compute demographic parity difference and plot groupwise performance.
    Returns a dict with audit metrics.
    """
    groups = df[protected_attr].unique()
    results = {}
    for g in groups:
        mask = df[protected_attr] == g
        pos_rate = df.loc[mask, pred_col].mean()
        true_pos = ((df.loc[mask, pred_col] == 1) & (df.loc[mask, label_col] == 1
        support = mask.sum()
        results[g] = {"positive_rate": pos_rate,
                      "true_positives": int(true_pos),
                      "support": int(support)}
    # Demographic parity diff (max - min)
    dp_diff = max(r["positive_rate"] for r in results.values()) - \
              min(r["positive_rate"] for r in results.values())
    audit = {"protected_attribute": protected_attr,
             "group_metrics": results,
             "demographic_parity_diff": dp_diff}
    return audit

# Helper: Energy-tracking placeholder ----------------------------------------
def log_energy(start_time, end_time, gpu_power_watts=250):
    """Very rough estimate of kWh consumed."""
    duration_h = (end_time - start_time).total_seconds() / 3600
    kwh = duration_h * (gpu_power_watts / 1000)
    return {"duration_h": round(duration_h, 2), "energy_kwh": round(kwh, 3)}
```

4. Phase 1

Design Narrative

- **Objective definition** – e.g., “Assist financial advisors with risk-profile summarisation.”
- **Stakeholder mapping** – regulators, end-users, auditors.
- **Data provenance plan** – source contracts, consent logs, versioned snapshots.
- **Ethical objectives** – list the six pillars that apply to this use-case.

Model Card:

```
# Minimal model-card skeleton
model_card = {
    "model_name": "risk_summariser_v1",
    "version": "1.0.0",
    "owner": "Your Organization",
    "intended_use": "Summarise client risk profiles for internal advisory workflo
    "limitations": [
        "Not validated for retail investors.",
        "Performance degrades on non-English inputs."
    ],
    "ethical_considerations": {
        "beneficence": "Extensive robustness testing before release.",
        "fairness": "Bias audit pipeline (see Phase 3).",
        "privacy": "Trained with differential-privacy noise (ε=1.0)."
    },
    "regulatory_alignment": ["NIST AI RMF v1 draft", "FINRA Notice 24-09"]
}

save_model_card(model_card)
```

```
Model card saved → model_cards/risk_summariser_v1_card_20250907T213209Z.json
/tmp/ipython-input-2487050218.py:19: DeprecationWarning: datetime.datetime.utc
  ts = dt.datetime.utcnow().strftime("%Y%m%dT%H%M%SZ")
PosixPath('model_cards/risk_summariser_v1_card_20250907T213209Z.json')
```

5.Phase 2 – Development

Data Loading & Bias-Mitigation:

```
# Example synthetic dataset — replace with your real data
np.random.seed(42)
N = 5000
df = pd.DataFrame({
    "age": np.random.randint(18, 80, size=N),
    "gender": np.random.choice(["male", "female", "nonbinary"], size=N, p=[0.48, 
    "income": np.random.lognormal(mean=10, sigma=0.5, size=N),
    "target": np.random.binomial(1, 0.2, size=N)  # binary outcome
})

# Simple bias-mitigation: re-weight under-represented groups
group_counts = df.groupby("gender").size()
weights = df["gender"].map(lambda g: 1 / group_counts[g])
df["sample_weight"] = weights / weights.mean()
```

Train a Small Model:

```
from sklearn.ensemble import GradientBoostingClassifier

X = df[["age", "income"]]
y = df["target"]
X_train, X_val, y_train, y_val, w_train, w_val = train_test_split(
    X, y, df["sample_weight"], test_size=0.2, random_state=123, stratify=y
)

clf = GradientBoostingClassifier(random_state=0)
clf.fit(X_train, y_train, sample_weight=w_train)

# Quick validation metrics
pred_val = clf.predict(X_val)
print(classification_report(y_val, pred_val))
```

```
              precision    recall  f1-score   support

           0       0.80      0.99      0.89       798
           1       0.33      0.01      0.02       202

    accuracy                           0.80      1000
   macro avg       0.57      0.50      0.45      1000
weighted avg       0.70      0.80      0.71      1000
```

Log Energy Consumption:

```
start = dt.datetime.utcnow()
# (Insert training loop here if you have a larger model)
end = dt.datetime.utcnow()
energy_log = log_energy(start, end, gpu_power_watts=250)  # adjust power estimate
print("Energy usage:", energy_log)
```

```
Energy usage: {'duration_h': 0.0, 'energy_kwh': 0.0}
/tmp/ipython-input-891902080.py:1: DeprecationWarning: datetime.datetime.utcnc
  start = dt.datetime.utcnow()
/tmp/ipython-input-891902080.py:3: DeprecationWarning: datetime.datetime.utcnc
  end = dt.datetime.utcnow()
```

5. Phase 3 – Testing

Robustness & Adversarial Stress Tests:

Robustness is tested by adding Gaussian noise to the numeric features and measuring the prediction stability.

Steps:

- **Perturbation**:
  - Gaussian noise is added to the numeric features of the input data. The formula for adding Gaussian noise is:

  $$X_{\text{noisy}} = X + \epsilon$$

  where:

  - X is the original input data.
  - ϵ is the Gaussian noise with a specified standard deviation.
- **Prediction Stability**:
  - Predictions are made on both the original and perturbed data.
  - The prediction stability is calculated as the proportion of consistent predictions between the original and perturbed data. The formula is:

$$\text{Prediction Stability} = \frac{\text{Number of Consistent Predictions}}{\text{Total Number of Predictions}}$$

where the *Number of Consistent Predictions* = the count of predictions that remain the same before and after adding noise.

Below is the code implementation

```
# Simple perturbation test: add Gaussian noise to numeric features
def robustness_check(model, X, noise_std=0.05):
    X_noisy = X.copy()
    X_noisy += np.random.normal(scale=noise_std, size=X.shape)
    preds_original = model.predict(X)
    preds_noisy = model.predict(X_noisy)
    stability = np.mean(preds_original == preds_noisy)
    return {"noise_std": noise_std, "prediction_stability": stability}

robust_res = robustness_check(clf, X_val, noise_std=0.1)
print("Robustness result:", robust_res)
```

```
Robustness result: {'noise_std': 0.1, 'prediction_stability': np.float64(1.0)}
```

Fairness Audit:

Fairness is calculated by computing the demographic parity difference, which measures the disparity in positive rates between different groups. The formula for demographic parity difference is:

$$\text{Demographic Parity Difference} = \max(\text{Positive Rate}_i) - \min(\text{Positive Rate}_i)$$

where:

- *Positive Rate* = proportion of positive predictions in each group.
- *i* - indexes over the different groups defined by the protected attribute.

Explainability with SHAP

Explainability is achieved using SHAP values, which provide a way to interpret the output of machine learning models. The SHAP value for a feature is calculated as:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{n!} [f(S \cup \{i\}) - f(S)]$$

where:

- ϕ_i is the SHAP value for feature i.
- S is a subset of features excluding feature i.
- N is the set of all features.
- n is the total number of features.
- $f(S)$ is the model output for the subset of features S.

```
# Generate predictions for the validation set
val_df = X_val.copy()
val_df["true"] = y_val.values
val_df["pred"] = pred_val
val_df["gender"] = df.loc[X_val.index, "gender"]

audit = bias_audit(val_df, label_col="true", protected_attr="gender", pred_col="p
print(json.dumps(audit, indent=2))
```

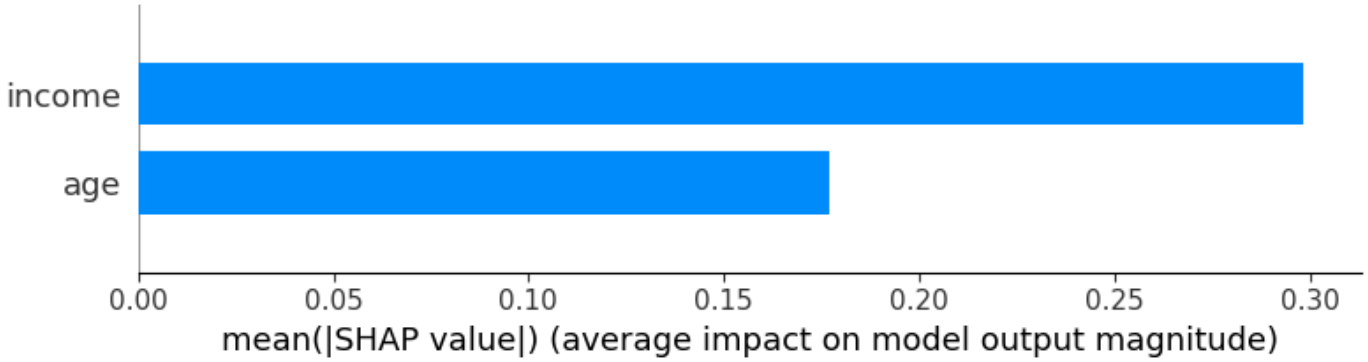

```
{
  "protected_attribute": "gender",
  "group_metrics": {
    "male": {
      "positive_rate": 0.006329113924050633,
      "true_positives": 1,
      "support": 474
    },
    "female": {
      "positive_rate": 0.006211180124223602,
      "true_positives": 1,
      "support": 483
    },
    "nonbinary": {
      "positive_rate": 0.0,
      "true_positives": 0,
      "support": 43
    }
  },
  "demographic_parity_diff": 0.006329113924050633
}
```


Explainability with SHAP:

```
explainer = shap.Explainer(clf, X_train)
shap_values = explainer(X_val.iloc[:200])  # limit for quick visual

# Plot summary (will render inline in notebook)
shap.summary_plot(shap_values, X_val.iloc[:200], plot_type="bar")
```

```
/tmp/ipython-input-2248249788.py:5: FutureWarning: The NumPy global RNG was se
  shap.summary_plot(shap_values, X_val.iloc[:200], plot_type="bar")
```



6.Phase 4 – Deployment

- **API wrapper** exposing `predict()` and `explain()` endpoints.
- **Model-card publication** (store JSON in a public bucket).
- **User notice** – UI banner: “Responses generated by an AI model; verify independently.”
- **Rate-limit** – 100 calls/min per API key to curb misuse.
- **Logging** – capture request metadata, prediction confidence, and audit-trail IDs.

```
# Minimal Flask example (pseudo-code – adapt to your infra)
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route("/predict", methods=["POST"])
def predict():
    payload = request.json
    features = np.array([payload["age"], payload["income"]]).reshape(1, -1)
    prob = clf.predict_proba(features)[0, 1]
    # Simple threshold
    label = int(prob > 0.5)
    return jsonify({"label": label, "probability": float(prob)})

# Run with: app.run(host="0.0.0.0", port=8080)
```

7.Phase 5 – Monitoring & Continuous Auditing

Monitoring

- **Performance drift** – weekly re-evaluation on a hold-out slice.
- **Fairness drift** – recompute demographic parity each month.
- **Incident log** – JSON entries stored in `monitoring/incidents/`.
- **Retraining trigger** – if accuracy drops > 2 % or DP-diff > 0.05.

```
# Example scheduled job (pseudo-code)
def monthly_audit():
    # Load latest batch of production data (placeholder)
    prod_df = pd.read_csv("data/production_batch.csv")
    prod_df["pred"] = clf.predict(prod_df[["age", "income"]])
    audit = bias_audit(prod_df, label_col="true", protected_attr="gender", pred_c
    # Persist audit
    ts = dt.datetime.utcnow().strftime("%Y%m%d")
    Path("monitoring/audits").mkdir(exist_ok=True)
    audit_path = f"monitoring/audits/bias_audit_{ts}.json"
    with open(audit_path, "w") as f:
        json.dump(audit, f, indent=2)
    print(f" Monthly bias audit saved → {audit_path}")

#Note: in production you'd hook this to Airflow / cron.
```

8.Phase 6 – Retirement:

- **Archive** model binaries, training data snapshots, and model-cards in immutable storage (e.g., S3 Glacier).
- **Delete** any temporary caches containing PII.
- **Notify** downstream consumers via email/webhook about deprecation date.
- **Update** the model-registry entry to `status: retired`.

Model-Welfare Checklist:

(check as you go)

- ☐ **Design** – objectives, stakeholder map, data-consent matrix.
- ☐ **Model Card** – created, version-controlled, linked to regulations.
- ☐ **Bias Mitigation** – re-weighting / debiasing applied.
- ☐ **Robustness Tests** – passed stability threshold (> 95 %).
- ☐ **Explainability** – SHAP summary generated & reviewed.
- ☐ **Regulatory Alignment** – NIST & FINRA controls documented.
- ☐ **Deployment** – API secured, rate-limited, user notice displayed.
- ☐ **Monitoring** – automated drift & fairness jobs active.
- ☐ **Retirement Plan** – archiving

How to interpret each checkbox:

(intuitive rationale - save as readme)

Design – objectives, stakeholder map, data-consent matrix. You've clearly defined what the model should do, who cares about its outcomes, and you have documented consent for every data source. This front-loads responsibility and avoids later "we didn't know we needed permission."

Model Card – created, version-controlled, linked to regulations. A model card is the "datasheet" for the model: architecture, training data provenance, intended use, limitations, etc. Keeping it under version control means you can trace changes over time, and linking to regulations shows you've mapped compliance requirements.

Bias Mitigation – re-weighting / debiasing applied. You've taken concrete steps (e.g., sample re-weighting, adversarial debiasing) to reduce unfair treatment of protected groups. The checkbox signals that bias isn't just a theoretical concern—it's been acted upon.

Robustness Tests – passed stability threshold (> 95 %). The model has been stress-tested

against distribution shifts, noisy inputs, or adversarial perturbations, and it maintains ≥ 95 % of its baseline performance. This gives confidence it won't catastrophically fail in production.

Explainability – SHAP summary generated & reviewed. You've produced SHAP (or similar) explanations for a representative set of predictions and had domain experts review them, ensuring the model's reasoning aligns with business logic.

Regulatory Alignment – NIST & FINRA controls documented. You've mapped the model's lifecycle to relevant standards (e.g., NIST AI Risk Management Framework, FINRA rules for financial services). Documentation proves due diligence for auditors.

Deployment – API secured, rate-limited, user notice displayed. The serving endpoint uses TLS, authentication, and throttling to prevent abuse, and any UI that calls the model shows a disclaimer or usage policy.

Monitoring – automated drift & fairness jobs active. Continuous pipelines watch for data drift, performance decay, and emerging fairness issues, alerting the team before things go wrong.

Retirement Plan – archiving You have a concrete plan for how the model will be retired (archiving binaries, cleaning caches, notifying stakeholders). This final box ties back to the Phase 6 steps above.

Conclusion

This notebook illustrates an end-to-end workflow for developing and deploying AI models with a strong emphasis on ethical considerations and regulatory compliance. It includes code and helper functions for each phase, from design and development to testing, deployment, monitoring, and retirement, along with key formulas and methods.

Below is a compact guide to the most relevant regulatory guidelines and policy documents that specifically address LLMs:

Region / Body	Guideline / Policy
European Union	EU AI Act (proposed) – Chapter II, Article 14 "High-risk AI systems"
European Medicines Agency (EMA)	Guidance on Safe Use of Large Language Models in Regulatory Science (2024)
United States – NIST	AI Risk Management Framework (RMF) – Draft Version 1.0 (2023)
United States – FINRA	Regulatory Notice 24-09 – Use of Generative AI & LLMs in the Securities Industry (2024)
United Kingdom	UK AI Strategy – Chapter 4 "Responsible AI" (2023)
Canada	Directive on Automated Decision-Making (2020) – Updated Guidance for LLM-Based Sy
Australia	AI Ethics Framework – "Transparency & Explainability" Principle (2023)