```python
# ============================================================
# Cell 1 — Environment, Invariants, and Global Assumptions
# ============================================================
# Purpose:
#   Establish a deterministic, auditable execution environment.
#   Define global invariants and explicit scope assumptions.
#   Perform zero analysis and load zero labels into memory.
#
# Non-negotiables:
#   - No stochasticity without explicit seeding
#   - No silent shape changes
#   - No downstream leakage from external labels
# ============================================================

# ------------------------
# Standard library
# ------------------------
import os
import sys
from pathlib import Path
from typing import Tuple

# ------------------------
# Numerical stack
# ------------------------
import numpy as np
import pandas as pd
import scipy as sp

# ------------------------
# Single-cell data
# ------------------------
import scanpy as sc
import anndata as ad

# ------------------------
# Reproducibility
# ------------------------
RANDOM_STATE: int = 42
np.random.seed(RANDOM_STATE)

# ------------------------
# Display / logging safety
# ------------------------
pd.set_option("display.max_rows", 20)
pd.set_option("display.max_columns", 20)
pd.set_option("display.width", 120)

# ============================================================
# File system layout (explicit, no magic paths)
# ============================================================
PROJECT_ROOT = Path.cwd()
DATA_DIR = PROJECT_ROOT / "data"

assert DATA_DIR.exists(), "Data directory does not exist."
assert DATA_DIR.is_dir(), "DATA_DIR is not a directory."

# Required files (existence only; not loaded here)
REQUIRED_FILES = [
    DATA_DIR / "obesity_challenge_1.h5ad",
    DATA_DIR / "program_proportion.csv",
]

for fp in REQUIRED_FILES:
    assert fp.exists(), f"Missing required file: {fp.name}"

# ============================================================
# Global assumptions (explicit, immutable)
# ============================================================
ASSUMPTIONS = {
    "expression_data": [
        "adata.X is preprocessed and normalized by the dataset
provider",
        "Raw counts are NOT treated as independent observations",
    ],
    "analysis_unit": [
        "All downstream analyses operate on perturbation-level means",
        "Individual-cell variability is not modeled",
    ],
    "label_usage": [
        "Cell-state program proportions are external-only",
        "They are NEVER used for representation construction",
    ],
    "scope": [
        "Analysis is descriptive and geometric",
        "No causal or mechanistic claims are made",
    ],
}

# ============================================================
# Sanity: environment fingerprint
# ============================================================
ENV_FINGERPRINT = {
    "python": sys.version.split()[0],
    "numpy": np.__version__,
    "pandas": pd.__version__,
    "scipy": sp.__version__,
    "scanpy": sc.__version__,
    "anndata": ad.__version__,
    "random_state": RANDOM_STATE,
}

for k, v in ENV_FINGERPRINT.items():
    print(f"{k:>12s}: {v}")

print("\nEnvironment initialized. No data loaded. No labels touched.")
```

```
      python: 3.11.9
       numpy: 2.2.0
      pandas: 2.3.2
       scipy: 1.16.1
      scanpy: 1.11.5
     anndata: 0.12.6
random_state: 42

Environment initialized. No data loaded. No labels touched.

C:\Users\Bryan\AppData\Local\Temp\ipykernel_21120\1206140970.py:97:
FutureWarning: `__version__` is deprecated, use
`importlib.metadata.version('scanpy')` instead
  "scanpy": sc.__version__,
C:\Users\Bryan\AppData\Local\Temp\ipykernel_21120\1206140970.py:98:
FutureWarning: `__version__` is deprecated, use
`importlib.metadata.version('anndata')` instead.
  "anndata": ad.__version__,
```

```
# ==============================================================
# Cell 2 — Data Loading and Structural Invariants
# ==============================================================
# Purpose:
#   Load the single-cell perturbation dataset exactly once.
#   Assert structural and semantic invariants required for all
#   downstream analysis.
#
# Scope:
#   - Expression data ONLY
#   - No aggregation
#   - No dimensionality reduction
#   - No external labels loaded
#
# Failure mode:
#   Any violation aborts execution immediately.
# ==============================================================

from importlib.metadata import version as pkg_version

# ------------------------
# Load AnnData object
# ------------------------
ADATA_PATH = DATA_DIR / "obesity_challenge_1.h5ad"
```

```
adata = sc.read_h5ad(ADATA_PATH)

# ------------------------
# Hard structural checks
# ------------------------
assert isinstance(adata, ad.AnnData), "adata is not an AnnData object"
assert adata.n_obs > 0, "adata contains zero cells"
assert adata.n_vars > 0, "adata contains zero genes"

# Expression matrix
assert adata.X is not None, "adata.X is None"
assert adata.X.shape == (adata.n_obs, adata.n_vars), "adata.X shape
mismatch"

# ------------------------
# Required annotations
# ------------------------
REQUIRED_OBS_COLS = ["gene"]
for col in REQUIRED_OBS_COLS:
    assert col in adata.obs.columns, f"Missing adata.obs column:
'{col}'"

# Control label must exist
assert "NC" in set(adata.obs["gene"]), "Control label 'NC' not found
in adata.obs['gene']"

# Gene identifiers
assert adata.var_names.is_unique, "Gene names are not unique"
assert adata.obs_names.is_unique, "Cell identifiers are not unique"

# ------------------------
# Dtype sanity (no silent object arrays)
# ------------------------
assert adata.obs["gene"].dtype == object or
pd.api.types.is_string_dtype(
    adata.obs["gene"]
), "adata.obs['gene'] must be string-like"

# ------------------------
# Provenance fingerprint (no deprecated access)
# ------------------------
DATA_FINGERPRINT = {
    "n_cells": adata.n_obs,
    "n_genes": adata.n_vars,
    "n_perturbations_total": adata.obs["gene"].nunique(),
    "scanpy_version": pkg_version("scanpy"),
    "anndata_version": pkg_version("anndata"),
}

for k, v in DATA_FINGERPRINT.items():
    print(f"{k:>22s}: {v}")

print("\nData loaded. Structural invariants satisfied.")
```

```
               n_cells: 88202
               n_genes: 21592
 n_perturbations_total: 123
        scanpy_version: 1.11.5
       anndata_version: 0.12.6

Data loaded. Structural invariants satisfied.
```

```
# ==============================================================
# Cell 3 — Control Separation and Perturbation Indexing
# ==============================================================
# Purpose:
#   Partition cells into control vs perturbed groups.
#   Construct a stable, auditable perturbation index.
#
# Scope:
#   - No expression aggregation
#   - No arithmetic on adata.X
#   - No dimensionality reduction
#
# Invariants established here are relied on downstream.
# ==============================================================

# ------------------------
# Identify control cells
# ------------------------
CONTROL_LABEL = "NC"

control_mask = adata.obs["gene"] == CONTROL_LABEL
perturb_mask = ~control_mask

# ------------------------
# Basic partition sanity
# ------------------------
n_control = int(control_mask.sum())
n_perturb = int(perturb_mask.sum())

assert n_control > 0, "No control cells found"
assert n_perturb > 0, "No perturbed cells found"
assert n_control + n_perturb == adata.n_obs, "Cell partition mismatch"

# ------------------------
# Slice AnnData objects (view-safe)
# ------------------------
adata_control = adata[control_mask].copy()
adata_perturb = adata[perturb_mask].copy()
```

```
# ------------------------
# Perturbation identity checks
# ------------------------
perturbation_labels = adata_perturb.obs["gene"]

assert perturbation_labels.nunique() > 1, "Only one perturbation
found"
assert CONTROL_LABEL not in set(perturbation_labels), "Control leaked
into perturbations"

# Stable, deterministic ordering of perturbations
PERTURBATION_GENES = tuple(sorted(perturbation_labels.unique()))

# Index mapping (gene → integer id)
PERTURBATION_INDEX = {g: i for i, g in enumerate(PERTURBATION_GENES)}

# ------------------------
# Consistency checks
# ------------------------
assert len(PERTURBATION_INDEX) == len(PERTURBATION_GENES)
assert set(PERTURBATION_INDEX.keys()) == set(PERTURBATION_GENES)

# ------------------------
# Summary fingerprint
# ------------------------
PARTITION_FINGERPRINT = {
    "n_control_cells": n_control,
    "n_perturbed_cells": n_perturb,
    "n_perturbations_excl_control": len(PERTURBATION_GENES),
}

for k, v in PARTITION_FINGERPRINT.items():
    print(f"{k:>30s}: {v}")

print("\nControl / perturbation partition established.")
```

```
               n_control_cells: 8705
             n_perturbed_cells: 79497
  n_perturbations_excl_control: 122

Control / perturbation partition established.
```

```
# ============================================================
# Cell 4 — Control Reference State (Mean Expression)
# ============================================================
# Purpose:
#   Define the reference expression state using control cells.
#   This is the sole baseline against which all perturbation
#   effects will be measured.
#
```

```
# Scope:
#   - One aggregation operation: control mean
#   - No perturbation information used
#   - No centering, scaling, or normalization
#
# This operation is irreversible and must be correct.
# =====================================================

# ------------------------
# Control expression matrix
# ------------------------
X_control = adata_control.X

assert X_control is not None, "Control expression matrix is None"
assert X_control.shape[0] == n_control, "Control cell count mismatch"
assert X_control.shape[1] == adata.n_vars, "Gene dimension mismatch"

# ------------------------
# Compute control mean (gene-wise)
# ------------------------
# IMPORTANT:
# - We operate on the AnnData slice, not raw arrays, to remain
#   compatible with sparse-backed matrices.
# - The result is explicitly densified and flattened.
control_mean = np.asarray(X_control.mean(axis=0)).ravel()

# ------------------------
# Postconditions
# ------------------------
assert control_mean.ndim == 1, "Control mean is not 1D"
assert control_mean.shape[0] == adata.n_vars, "Control mean length
mismatch"
assert np.all(np.isfinite(control_mean)), "Non-finite values in
control mean"

# ------------------------
# Reference fingerprint
# ------------------------
REFERENCE_FINGERPRINT = {
    "reference_type": "control_mean_expression",
    "n_genes": control_mean.shape[0],
    "mean_min": float(control_mean.min()),
    "mean_max": float(control_mean.max()),
    "mean_l2_norm": float(np.linalg.norm(control_mean)),
}

for k, v in REFERENCE_FINGERPRINT.items():
    print(f"{k:>22s}: {v}")

print("\nControl reference state constructed.")
```

```
        reference_type: control_mean_expression
               n_genes: 21592
              mean_min: 0.0
              mean_max: 11.328900940274325
          mean_l2_norm: 234.93018084696672

Control reference state constructed.

# ============================================================
# Cell 5 — Perturbation-Level Mean Effects (Delta Matrix)
# ============================================================
# Purpose:
#   Construct a perturbation-by-gene matrix of mean effects
#   relative to the fixed control reference state.
#
# Definition:
#   For each perturbation g:
#       Δ_g = mean(X | perturbation = g) − control_mean
#
# Scope:
#   - One aggregation per perturbation
#   - No dimensionality reduction
#   - No use of external labels
#
# This defines the core geometric object for all downstream work.
# ============================================================

# ------------------------
# Pre-conditions
# ------------------------
assert "control_mean" in globals(), "control_mean not defined"
assert adata_perturb is not None, "Perturbation AnnData not available"
assert len(PERTURBATION_GENES) > 0, "No perturbations indexed"

n_genes = adata.n_vars
n_perts = len(PERTURBATION_GENES)

# ------------------------
# Allocate delta matrix
# ------------------------
delta_matrix = np.zeros((n_perts, n_genes), dtype=np.float64)

# ------------------------
# Compute perturbation means
# ------------------------
for gene, row_idx in PERTURBATION_INDEX.items():
    mask = adata_perturb.obs["gene"] == gene
    n_cells_g = int(mask.sum())

    assert n_cells_g > 0, f"No cells found for perturbation '{gene}'"
    X_g = adata_perturb[mask].X
    mean_g = np.asarray(X_g.mean(axis=0)).ravel()

    # Sanity
    assert mean_g.shape[0] == n_genes, "Gene dimension mismatch"
    assert np.all(np.isfinite(mean_g)), f"Non-finite values for
perturbation '{gene}'"

    # Contrastive delta
    delta_matrix[row_idx, :] = mean_g - control_mean

# -------------------------
# Post-conditions
# -------------------------
assert delta_matrix.shape == (n_perts, n_genes), "Delta matrix shape
incorrect"
assert np.all(np.isfinite(delta_matrix)), "Non-finite values in delta
matrix"

# -------------------------
# Delta fingerprint
# -------------------------
DELTA_FINGERPRINT = {
    "n_perturbations": n_perts,
    "n_genes": n_genes,
    "delta_min": float(delta_matrix.min()),
    "delta_max": float(delta_matrix.max()),
    "delta_l2_norm_mean": float(np.linalg.norm(delta_matrix,
axis=1).mean()),
}

for k, v in DELTA_FINGERPRINT.items():
    print(f"{k:>26s}: {v}")

print("\nPerturbation delta matrix constructed.")
```

```
           n_perturbations: 122
                   n_genes: 21592
                 delta_min: -2.413160548324867
                 delta_max: 2.4744439633529143
        delta_l2_norm_mean: 7.122000826190643

Perturbation delta matrix constructed.
```

```
# ==============================================================
# Cell 6 — Low-Dimensional Structure Diagnostic (PCA)
# ==============================================================
# Purpose:
#   Assess whether perturbation effect vectors concentrate along
```

```python
#   a small number of dominant directions in gene-expression space.
#
# Scope:
#   - PCA used strictly as a geometric diagnostic
#   - No biological interpretation
#   - No external labels
#
# PCA is NOT a model here. It is a variance accounting tool.
# ============================================================

from sklearn.decomposition import PCA

# ------------------------
# Preconditions
# ------------------------
assert "delta_matrix" in globals(), "delta_matrix not defined"
assert delta_matrix.ndim == 2, "delta_matrix must be 2D"
assert np.all(np.isfinite(delta_matrix)), "Non-finite values in 
delta_matrix"

n_perts, n_genes = delta_matrix.shape
assert n_perts > 1, "Need at least two perturbations for PCA"

# ------------------------
# PCA configuration
# ------------------------
# NOTE:
# - Centering is intrinsic to PCA and does not use control labels.
# - No scaling is applied; gene variance structure is preserved.
N_PCS = min(50, n_perts)

pca = PCA(
    n_components=N_PCS,
    svd_solver="full",
    random_state=RANDOM_STATE,
)

# ------------------------
# Fit PCA
# ------------------------
pca.fit(delta_matrix)

explained_var = pca.explained_variance_ratio_
cum_explained_var = np.cumsum(explained_var)

# ------------------------
# Sanity checks
# ------------------------
assert explained_var.ndim == 1
assert cum_explained_var[-1] <= 1.0 + 1e-6
assert np.all(np.diff(cum_explained_var) >= 0), "Cumulative variance
not monotone"

# ------------------------
# Diagnostic report
# ------------------------
PCA_FINGERPRINT = {
    "n_perturbations": n_perts,
    "n_genes": n_genes,
    "n_pcs_fitted": N_PCS,
    "var_pc1": float(cum_explained_var[0]),
    "var_pc2": float(cum_explained_var[1]) if N_PCS >= 2 else None,
    "var_pc5": float(cum_explained_var[4]) if N_PCS >= 5 else None,
    "var_pc10": float(cum_explained_var[9]) if N_PCS >= 10 else None,
    "var_pc20": float(cum_explained_var[19]) if N_PCS >= 20 else None,
    "var_pc50": float(cum_explained_var[49]) if N_PCS >= 50 else None,
}

for k, v in PCA_FINGERPRINT.items():
    print(f"{k:>22s}: {v}")

print("\nLow-dimensional structure diagnostic complete.")
```

```
       n_perturbations: 122
               n_genes: 21592
          n_pcs_fitted: 50
               var_pc1: 0.36763845402452505
               var_pc2: 0.4203934428059476
               var_pc5: 0.5166831540505471
              var_pc10: 0.6004512392854648
              var_pc20: 0.6963495155295178
              var_pc50: 0.8360428911277359

Low-dimensional structure diagnostic complete.
```

```
# ============================================================
# Cell 7 — Leading Perturbation Direction (PC1)
# ============================================================
# Purpose:
#   Extract the leading principal direction (PC1) from the
#   perturbation delta matrix and project perturbations onto it.
#
# Scope:
#   - Pure linear algebra
#   - No biological interpretation
#   - No external labels
#
# PC1 is treated strictly as a geometric object.
# ============================================================
```

```python
# -------------------------
# Preconditions
# -------------------------
assert "pca" in globals(), "PCA object not found"
assert hasattr(pca, "components_"), "PCA not fitted"
assert pca.components_.shape[1] == n_genes, "PC dimension mismatch"

# -------------------------
# Extract leading direction
# -------------------------
v1 = pca.components_[0].astype(np.float64, copy=True)

# -------------------------
# Normalize (explicit, defensive)
# -------------------------
v1_norm = np.linalg.norm(v1)
assert v1_norm > 0.0, "Zero-norm leading direction"

v1 /= v1_norm

# -------------------------
# Direction sanity checks
# -------------------------
assert v1.shape == (n_genes,), "Leading direction shape incorrect"
assert np.isclose(np.linalg.norm(v1), 1.0, atol=1e-8), "Leading
direction not unit-norm"
assert np.all(np.isfinite(v1)), "Non-finite values in leading
direction"

# -------------------------
# Project perturbations onto PC1
# -------------------------
pc1_scores = delta_matrix @ v1

# -------------------------
# Post-conditions
# -------------------------
assert pc1_scores.ndim == 1, "PC1 scores not 1D"
assert pc1_scores.shape[0] == n_perts, "PC1 score length mismatch"
assert np.all(np.isfinite(pc1_scores)), "Non-finite PC1 scores"

# -------------------------
# PC1 fingerprint
# -------------------------
PC1_FINGERPRINT = {
    "pc1_l2_norm": float(np.linalg.norm(v1)),
    "pc1_score_min": float(pc1_scores.min()),
    "pc1_score_max": float(pc1_scores.max()),
    "pc1_score_mean": float(pc1_scores.mean()),
    "pc1_score_std": float(pc1_scores.std(ddof=1)),
}

for k, v in PC1_FINGERPRINT.items():
    print(f"{k:>22s}: {v}")

print("\nLeading perturbation direction extracted and validated.")
```

```
           pc1_l2_norm: 1.0
         pc1_score_min: -10.57258044385466
         pc1_score_max: 24.115190994354215
        pc1_score_mean: 0.6232159042381975
         pc1_score_std: 4.5684611139926306

Leading perturbation direction extracted and validated.
```

```
# ============================================================
# Cell 8B — Random-Axis Null Test (Falsification)
# ============================================================
# Purpose:
#   Test whether the observed alignment between the leading
#   perturbation direction (PC1) and an external scalar signal
#   could plausibly arise from arbitrary directions in gene space.
#
# Interpretation:
#   - If random axes perform similarly to PC1, the alignment is
trivial.
#   - If PC1 is exceptional, it should lie far in the null tail.
#
# Scope:
#   - No modification of v1 or pc1_scores
#   - External labels used strictly for evaluation
#   - Sign-invariant test statistic
# ============================================================

from scipy.stats import spearmanr

# ------------------------
# Load external labels (evaluation-only)
# ------------------------
PROGRAM_PATH = DATA_DIR / "program_proportion.csv"
program_props = pd.read_csv(PROGRAM_PATH, index_col=0)

# ------------------------
# Align perturbation ordering
# ------------------------
# Ensure strict alignment with PERTURBATION_GENES ordering
program_props = program_props.loc[list(PERTURBATION_GENES)]

assert program_props.shape[0] == n_perts, "Program proportion
alignment failed"
```

```python
assert "adipo" in program_props.columns, "Required column 'adipo' not
found"

adipo_prop = program_props["adipo"].values
assert np.all(np.isfinite(adipo_prop)), "Non-finite values in adipo
proportions"

# ------------------------
# Observed statistic (PC1)
# ------------------------
rho_pc1, _ = spearmanr(pc1_scores, adipo_prop)
rho_pc1_abs = abs(rho_pc1)

# ------------------------
# Random-axis null distribution
# ------------------------
N_RANDOM = 1000
rng = np.random.default_rng(RANDOM_STATE)

null_rhos = np.empty(N_RANDOM, dtype=np.float64)

for i in range(N_RANDOM):
    v_rand = rng.normal(size=n_genes)
    v_rand /= np.linalg.norm(v_rand)

    scores_rand = delta_matrix @ v_rand
    rho_rand, _ = spearmanr(scores_rand, adipo_prop)

    null_rhos[i] = abs(rho_rand)

# ------------------------
# Empirical significance
# ------------------------
p_empirical = float(np.mean(null_rhos >= rho_pc1_abs))

# ------------------------
# Null fingerprint
# ------------------------
NULL_FINGERPRINT = {
    "n_random_axes": N_RANDOM,
    "null_mean_abs_rho": float(null_rhos.mean()),
    "null_p95_abs_rho": float(np.percentile(null_rhos, 95)),
    "observed_abs_rho_pc1": float(rho_pc1_abs),
    "empirical_p_value": p_empirical,
}

for k, v in NULL_FINGERPRINT.items():
    print(f"{k:>28s}: {v}")

print("\nRandom-axis null test complete.")
```

```
               n_random_axes: 1000
           null_mean_abs_rho: 0.38763371285210396
            null_p95_abs_rho: 0.752655718962565
        observed_abs_rho_pc1: 0.9657210929213287
           empirical_p_value: 0.0

Random-axis null test complete.
```

```
# ============================================================
# Cell 9 — Orthogonalized Random-Axis Null Test
# ============================================================
# Purpose:
#   Test whether directions constrained to lie orthogonal to PC1
#   can achieve comparable alignment with the external signal.
#
# Interpretation:
#   - If PC1-orthogonal axes perform similarly, signal is diffuse.
#   - If they do not, PC1 captures a genuinely privileged direction.
#
# Scope:
#   - PC1 is fixed and not refit
#   - External labels used strictly for evaluation
#   - Sign-invariant statistic
# ============================================================

# ------------------------
# Preconditions
# ------------------------
assert "v1" in globals(), "Leading direction v1 not found"
assert np.isclose(np.linalg.norm(v1), 1.0), "v1 is not unit-norm"

# ------------------------
# Orthogonalized random-axis null
# ------------------------
N_RANDOM_ORTHO = 1000
rng = np.random.default_rng(RANDOM_STATE)

null_rhos_ortho = np.empty(N_RANDOM_ORTHO, dtype=np.float64)

for i in range(N_RANDOM_ORTHO):
    # Sample random direction
    v_rand = rng.normal(size=n_genes)

    # Explicit orthogonalization against PC1
    v_rand -= np.dot(v_rand, v1) * v1

    v_norm = np.linalg.norm(v_rand)
    assert v_norm > 0.0, "Degenerate orthogonal random vector"

    v_rand /= v_norm
    # Project perturbations
    scores_rand = delta_matrix @ v_rand

    rho_rand, _ = spearmanr(scores_rand, adipo_prop)
    null_rhos_ortho[i] = abs(rho_rand)

# ------------------------
# Empirical comparison
# ------------------------
p_empirical_ortho = float(np.mean(null_rhos_ortho >= rho_pc1_abs))

# ------------------------
# Orthogonal null fingerprint
# ------------------------
ORTHO_NULL_FINGERPRINT = {
    "n_random_axes_orthogonal": N_RANDOM_ORTHO,
    "orth_null_mean_abs_rho": float(null_rhos_ortho.mean()),
    "orth_null_p95_abs_rho": float(np.percentile(null_rhos_ortho,
95)),
    "observed_abs_rho_pc1": float(rho_pc1_abs),
    "empirical_p_value_orthogonal": p_empirical_ortho,
}

for k, v in ORTHO_NULL_FINGERPRINT.items():
    print(f"{k:>34s}: {v}")

print("\nPC1-orthogonal random-axis null test complete.")
```

```
          n_random_axes_orthogonal: 1000
            orth_null_mean_abs_rho: 0.06339672956731678
             orth_null_p95_abs_rho: 0.15206768873907353
              observed_abs_rho_pc1: 0.9657210929213287
      empirical_p_value_orthogonal: 0.0

PC1-orthogonal random-axis null test complete.
```

```
# ==============================================================
# Cell 10 — External Alignment Report (Correlation Table)
# ==============================================================
# Purpose:
#   Quantify alignment between the fixed PC1 coordinate and
#   independently estimated cell-state program proportions.
#
# Scope:
#   - Evaluation only (no construction, no tuning)
#   - Rank-based correlation (Spearman)
#   - No causal or biological interpretation
#
# This cell reports numbers. Nothing more.
# ==============================================================

from scipy.stats import spearmanr

# ------------------------
# Preconditions
# ------------------------
assert "pc1_scores" in globals(), "pc1_scores not found"
assert "program_props" in globals(), "program proportions not loaded"
assert program_props.shape[0] == n_perts, "Program proportion shape
mismatch"

# ------------------------
# Correlation computation
# ------------------------
alignment_results = {}

for col in program_props.columns:
    values = program_props[col].values

    assert np.all(np.isfinite(values)), f"Non-finite values in column
'{col}'"

    rho, pval = spearmanr(pc1_scores, values)
    alignment_results[col] = {
        "spearman_rho": float(rho),
        "abs_rho": float(abs(rho)),
        "p_value": float(pval),
    }

# ------------------------
# Reporting (sorted by |rho|)
# ------------------------
alignment_df = (
    pd.DataFrame(alignment_results)
      .T
      .sort_values("abs_rho", ascending=False)
)

print("PC1 alignment with external programs (Spearman):")
display(alignment_df)

print("\nExternal alignment report complete.")
```

```
PC1 alignment with external programs (Spearman):
```

{"columns":[{"name":"index","rawType":"object","type":"string"},
{"name":"spearman_rho","rawType":"float64","type":"float"},
{"name":"abs_rho","rawType":"float64","type":"float"},
{"name":"p_value","rawType":"float64","type":"float"}],"ref":"c6263077
-c0ee-4100-a30b-f158bd2fdfc9","rows":

```
[["adipo","0.9657210929213287","0.9657210929213287","3.883899654166383
e-72"],
["lipo","0.8877885007802403","0.8877885007802403","2.9622764991267545e
-42"],["pre_adipo","-
0.8547952719738552","0.8547952719738552","5.557561553812468e-36"],
["lipo_adipo","0.5285918690375089","0.5285918690375089","3.89846977287
88264e-10"],["other","-
0.38724739401382247","0.38724739401382247","1.0508223366333835e-
05"]],"shape":{"columns":3,"rows":5}}
```

```
External alignment report complete.
```

```
# ============================================================
# Cell 11A — Residual Decomposition (PC1 vs Orthogonal)
# ============================================================
# Purpose:
#   Decompose each perturbation effect vector into:
#     (1) Component aligned with PC1
#     (2) Orthogonal residual component
#
# This allows explicit identification of:
#   - Perturbations well-explained by PC1
#   - Perturbations where PC1 fails
#
# Scope:
#   - Pure linear algebra
#   - No model fitting
#   - No biological interpretation
# ============================================================

# ------------------------
# Preconditions
# ------------------------
assert "delta_matrix" in globals(), "delta_matrix not found"
assert "v1" in globals(), "Leading direction v1 not found"
assert np.isclose(np.linalg.norm(v1), 1.0), "v1 must be unit-norm"

# ------------------------
# Parallel component (along PC1)
# ------------------------
# Δ_parallel = (Δ · v1) v1
parallel_component = np.outer(pc1_scores, v1)

# ------------------------
# Orthogonal residual
# ------------------------
residual_matrix = delta_matrix - parallel_component

# ------------------------
```

```python
# Norm diagnostics
# ------------------------
delta_norms = np.linalg.norm(delta_matrix, axis=1)
parallel_norms = np.linalg.norm(parallel_component, axis=1)
residual_norms = np.linalg.norm(residual_matrix, axis=1)

# ------------------------
# Sanity checks
# ------------------------
reconstruction_error = np.linalg.norm(
    delta_matrix - (parallel_component + residual_matrix)
)

assert np.isclose(reconstruction_error, 0.0, atol=1e-8),
"Decomposition not exact"
assert np.all(residual_norms >= 0.0)
assert np.all(parallel_norms >= 0.0)

# ------------------------
# Explained fraction per perturbation
# ------------------------
explained_fraction = np.divide(
    parallel_norms,
    delta_norms,
    out=np.zeros_like(parallel_norms),
    where=delta_norms > 0,
)

# ------------------------
# Residual fingerprint
# ------------------------
RESIDUAL_FINGERPRINT = {
    "mean_fraction_explained_by_pc1":
float(explained_fraction.mean()),
    "median_fraction_explained_by_pc1":
float(np.median(explained_fraction)),
    "min_fraction_explained_by_pc1": float(explained_fraction.min()),
    "max_fraction_explained_by_pc1": float(explained_fraction.max()),
}

for k, v in RESIDUAL_FINGERPRINT.items():
    print(f"{k:>36s}: {v}")

print("\nResidual decomposition complete.")
```

```
      mean_fraction_explained_by_pc1: 0.3961941117146196
    median_fraction_explained_by_pc1: 0.40647545669646434
       min_fraction_explained_by_pc1: 0.013439956341176O3
       max_fraction_explained_by_pc1: 0.9240362045188794
```

```
Residual decomposition complete.

# ==============================================================
# Cell 12 — Necessary vs Sufficient Constraint Test
# ==============================================================
# Purpose:
#   Test whether positive movement along PC1 is:
#     - necessary for high adipogenesis
#     - sufficient for high adipogenesis
#
# This is framed as a constraint test, not a prediction task.
#
# Scope:
#   - Thresholds defined by quantiles (non-parametric, non-tuned)
#   - No model fitting
#   - No causal interpretation
# ==============================================================

# ------------------------
# Preconditions
# ------------------------
assert "pc1_scores" in globals(), "pc1_scores not found"
assert "program_props" in globals(), "program proportions not found"
assert "adipo" in program_props.columns, "'adipo' column missing"

# ------------------------
# Define thresholds (explicit, fixed)
# ------------------------
# PC1-positive: top 50% of PC1 scores
pc1_threshold = np.quantile(pc1_scores, 0.50)

# High adipogenesis: top 25% of adipo proportions
adipo_values = program_props["adipo"].values
adipo_threshold = np.quantile(adipo_values, 0.75)

# ------------------------
# Binary indicators
# ------------------------
pc1_positive = pc1_scores >= pc1_threshold
adipo_high = adipo_values >= adipo_threshold

# ------------------------
# Confusion structure
# ------------------------
true_positive = np.sum(pc1_positive & adipo_high)
false_positive = np.sum(pc1_positive & ~adipo_high)
false_negative = np.sum(~pc1_positive & adipo_high)
true_negative = np.sum(~pc1_positive & ~adipo_high)
```

```
# ------------------------
# Rates (constraint diagnostics)
# ------------------------
false_positive_rate = false_positive / max(1, np.sum(pc1_positive))
false_negative_rate = false_negative / max(1, np.sum(adipo_high))

# ------------------------
# Report
# ------------------------
NECESSITY_FINGERPRINT = {
    "pc1_threshold_median": float(pc1_threshold),
    "adipo_threshold_p75": float(adipo_threshold),
    "true_positive": int(true_positive),
    "false_positive": int(false_positive),
    "false_negative": int(false_negative),
    "true_negative": int(true_negative),
    "false_positive_rate": float(false_positive_rate),
    "false_negative_rate": float(false_negative_rate),
}

for k, v in NECESSITY_FINGERPRINT.items():
    print(f"{k:>28s}: {v}")

print("\nNecessary vs sufficient constraint test complete.")
```

```
        pc1_threshold_median: 0.13761680390429137
         adipo_threshold_p75: 0.30651814714345277
               true_positive: 31
              false_positive: 30
              false_negative: 0
               true_negative: 61
         false_positive_rate: 0.4918032786885246
         false_negative_rate: 0.0

Necessary vs sufficient constraint test complete.
```

```
# ============================================================
# Cell 13 — Failure-Case Enumeration (PC1 Violations)
# ============================================================
# Purpose:
#   Explicitly identify perturbations where PC1 fails to explain
#   behavior, using two independent criteria:
#
#   (A) Large PC1-orthogonal residuals
#   (B) PC1-positive but low adipogenesis (false positives)
#
# Scope:
#   - No model fitting
#   - No interpretation
#   - Explicit, reproducible rankings
# ============================================================

# ------------------------
# Preconditions
# ------------------------
assert "residual_norms" in globals(), "residual_norms not found"
assert "explained_fraction" in globals(), "explained_fraction not
found"
assert "program_props" in globals(), "program_props not found"

# ------------------------
# Build perturbation-level table
# ------------------------
failure_df = pd.DataFrame({
    "perturbation": PERTURBATION_GENES,
    "pc1_score": pc1_scores,
    "delta_norm": delta_norms,
    "pc1_parallel_norm": parallel_norms,
    "pc1_residual_norm": residual_norms,
    "fraction_explained_by_pc1": explained_fraction,
    "adipo": program_props["adipo"].values,
})

# ------------------------
# Failure mode A: largest residuals
# ------------------------
failure_df["residual_rank"] = (
    failure_df["pc1_residual_norm"]
    .rank(ascending=False, method="min")
)

# ------------------------
# Failure mode B: false positives
# ------------------------
failure_df["pc1_positive"] = pc1_scores >= pc1_threshold
failure_df["adipo_high"] = failure_df["adipo"] >= adipo_threshold

failure_df["false_positive"] = (
    failure_df["pc1_positive"] & ~failure_df["adipo_high"]
)

# ------------------------
# Reporting
# ------------------------
print("Top perturbations by PC1-orthogonal residual norm:")
display(
    failure_df
    .sort_values("pc1_residual_norm", ascending=False)
    .head(10)
)
```

```
print("\nPC1-positive but low-adipogenesis perturbations (false
positives):")
display(
    failure_df
    .loc[failure_df["false_positive"]]
    .sort_values("pc1_score", ascending=False)
)

print("\nFailure-case enumeration complete.")
```

```
Top perturbations by PC1-orthogonal residual norm:
```

{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"perturbation","rawType":"object","type":"string"},
{"name":"pc1_score","rawType":"float64","type":"float"},
{"name":"delta_norm","rawType":"float64","type":"float"},
{"name":"pc1_parallel_norm","rawType":"float64","type":"float"},
{"name":"pc1_residual_norm","rawType":"float64","type":"float"},
{"name":"fraction_explained_by_pc1","rawType":"float64","type":"float"
},{"name":"adipo","rawType":"float64","type":"float"},
{"name":"residual_rank","rawType":"float64","type":"float"},
{"name":"pc1_positive","rawType":"bool","type":"boolean"},
{"name":"adipo_high","rawType":"bool","type":"boolean"},
{"name":"false_positive","rawType":"bool","type":"boolean"}],"ref":"cb
987d9b-1410-4beb-9e85-fb100858a283","rows":[["88","SRPK1","-
5.997044053494493","17.492289256379774","5.997044053494493","16.432152
812376888","0.3428392913927636","0.1951219512195122","1.0","False","Fa
lse","False"],
["98","TMEM107","10.890743426486434","15.392489297391645","10.89074342
6486434","10.877519679998692","0.7075362026291576","0.4009433962264151
","2.0","True","True","False"],
["80","RNASEH2C","24.115190994354215","26.09766898355497","24.11519099
4354215","9.977268648333794","0.9240362045188794","0.5941644562334217"
,"3.0","True","True","False"],
["23","EP400","13.410412667644328","16.369463534922776","13.4104126676
4433","9.387234337367472","0.8192334855100427","0.4319444444444444","4
.0","True","True","False"],["90","SUPT5H","-
0.31537297611995907","9.244722528244527","0.31537297611995907","9.2393
4167082077","0.03411383902074182","0.2489539748953975","5.0","False","
False","False"],
["71","PLIN1","1.5447864544415866","8.919735600190823","1.544786454441
5868","8.784948377052952","0.17318747143228536","0.2924107142857143","
6.0","True","False","True"],["25","EWSR1","-
5.458601366506409","10.342153269679276","5.458601366506409","8.7842931
06170361","0.5278012444961269","0.1786600496277915","7.0","False","Fal
se","False"],["10","CEBPA","-
10.57258044385466","13.524040122085685","10.57258044385466","8.4332795
5080379","0.7817619844671202","0.086864406779661","8.0","False","False
","False"],

["26","FAM136A","17.327423694402366","19.11074918985827","17.327423694
40237","8.061086943603321","0.9066846894520346","0.4965986394557823","
9.0","True","True","False"],["49","MEF2A","-
2.604504533742002","8.27004240539823","2.604504533742002","7.849213815
459628","0.31493242792103754","0.2121212121212121","10.0","False","Fal
se","False"]],"shape":{"columns":11,"rows":10}}

PC1-positive but low-adipogenesis perturbations (false positives):

{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"perturbation","rawType":"object","type":"string"},
{"name":"pc1_score","rawType":"float64","type":"float"},
{"name":"delta_norm","rawType":"float64","type":"float"},
{"name":"pc1_parallel_norm","rawType":"float64","type":"float"},
{"name":"pc1_residual_norm","rawType":"float64","type":"float"},
{"name":"fraction_explained_by_pc1","rawType":"float64","type":"float"
},{"name":"adipo","rawType":"float64","type":"float"},
{"name":"residual_rank","rawType":"float64","type":"float"},
{"name":"pc1_positive","rawType":"bool","type":"boolean"},
{"name":"adipo_high","rawType":"bool","type":"boolean"},
{"name":"false_positive","rawType":"bool","type":"boolean"}],"ref":"68
0e06e5-3dd5-4d76-916f-0ce65d847414","rows":
[["74","PPARD","4.226545250867674","7.658879824250283","4.226545250867
674","6.387077219251068","0.5518490102802213","0.2948717948717949","34
.0","True","False","True"],
["77","RBAK","3.5456600138367467","6.041067598616929","3.5456600138367
467","4.891093210861812","0.58692606165316","0.3054619015509103","103.
0","True","False","True"],
["38","HMBOX1","3.5167376067404037","6.440463285192998","3.51673760674
0404","5.395565209804879","0.5460379868674339","0.30441400304414","80.
0","True","False","True"],
["60","NPM1","2.796092148629351","7.1072614209696185","2.7960921486293
51","6.5341436778185695","0.3934134377524965","0.2968036529680365","28
.0","True","False","True"],
["108","ZHX3","2.479717383960802","5.2316010990886195","2.479717383960
802","4.606587864750638","0.47398823744279467","0.2917251051893408","1
09.0","True","False","True"],
["41","IRF4","2.3609880151476212","4.622221400178483","2.3609880151476
212","3.9737471314361765","0.5107907671961481","0.2986463620981388","1
18.0","True","False","True"],
["109","ZNF138","2.3196258780302093","7.1088366769222295","2.319625878
0302093","6.7197391828200645","0.32630175420410573","0.297650130548302
8","23.0","True","False","True"],
["100","TRRAP","1.9446440101382974","6.131416069106359","1.94464401013
82976","5.814862224191464","0.31716066700097967","0.2890716803760282",
"57.0","True","False","True"],
["121","ZNF480","1.7509503426815418","6.151594125712761","1.7509503426
81542","5.897141950552584","0.28463359365059965","0.3054054054054054",
"53.0","True","False","True"],

```
["32","GRB14","1.6127582198245831","7.540458651126101","1.612758219824
5833","7.36597091996233","0.21388065294724906","0.2614770459081836","1
4.0","True","False","True"],
["116","ZNF331","1.5469423188291875","4.513816448671485","1.5469423188
291875","4.240460870531956","0.34271272135677705","0.284313725490196",
"113.0","True","False","True"],
["71","PLIN1","1.5447864544415866","8.919735600190823","1.544786454441
5868","8.784948377052952","0.17318747143228536","0.2924107142857143","
6.0","True","False","True"],
["79","REXO4","1.452962252604016","5.903686047099606","1.4529622526040
16","5.72209835945053","0.24611102978923394","0.2887323943661972","66.
0","True","False","True"],
["30","FOXP1","1.4067763908376507","4.30629909064897","1.4067763908376
507","4.070035877520731","0.32667874692968574","0.2955326460481099","1
16.0","True","False","True"],
["96","TFAP2A","1.3190211628770583","4.381034915791265","1.31902116287
70585","4.177756587603523","0.3010752455139537","0.2637759710930442","
114.0","True","False","True"],
["0","AATF","1.2599448589640136","5.734985684162657","1.25994485896401
36","5.59487263035726","0.21969450812116076","0.2680608365019011","74.
0","True","False","True"],
["120","ZNF44","1.2003228970307789","5.18764657297059","1.200322897030
7789","5.046870506454187","0.23138100873811857","0.2829888712241653","
93.0","True","False","True"],
["42","JARID2","1.1180565810954715","6.630103919014511","1.11805658109
54715","6.53515320848721","0.16863334191323773","0.253012048192771","2
7.0","True","False","True"],
["43","KLF15","1.0593467336304943","6.0005422227351115","1.05934673363
04943","5.9062925312562395","0.1765418347723304","0.271523178807947","
50.0","True","False","True"],
["8","BTG2","0.9139844779438023","5.856970209848422","0.91398447794380
24","5.7852167127195555","0.15605073018929635","0.2757201646090535","5
9.0","True","False","True"],
["5","BDP1","0.878115422651578","6.4640573705242055","0.87811542265157
8","6.404135460304516","0.13584585846278882","0.2629427792915531","33.
0","True","False","True"],
["117","ZNF334","0.8198327139459889","5.361348701207203","0.8198327139
459889","5.298295406739805","0.15291538745864244","0.2863128491620111"
,"85.0","True","False","True"],
["47","KLF7","0.7847111449311853","5.082024381466912","0.7847111449311
853","5.02107560517116","0.15440916572397095","0.263494967978042","95.
0","True","False","True"],
["13","CIC","0.7221483905065753","6.028832852512755","0.72214839050657
53","5.985426239260312","0.11978245344877846","0.2642045454545454","47
.0","True","False","True"],
["55","NFIA","0.7178394902860197","6.896208622431515","0.7178394902860
197","6.858746228742151","0.10409190463743812","0.2621082621082621","2
1.0","True","False","True"],
["112","ZNF215","0.5011777191822525","5.776697235917788","0.5011777191
822525","5.754915451095306","0.08675852285733071","0.259765625","63.0"
,"True","False","True"],
["16","DDX5","0.46173761872534624","5.780225476603983","0.461737618725
3463","5.761753633385896","0.07988228497214747","0.2752","62.0","True"
,"False","True"],
["92","TCERG1","0.4222376690488525","5.670412760366987","0.42223766904
88525","5.654670301951207","0.07446330397675062","0.2728971962616822",
"68.0","True","False","True"],
["35","HDAC2","0.200432263424174","6.231908411579686","0.2004322634241
74","6.228684400264337","0.0321622607693889","0.2821576763485477","39.
0","True","False","True"],
["40","HMGN3","0.15982157659859675","7.075706840765032","0.15982157659
859675","7.073901636303873","0.022587365502174578","0.2641083521444695
","16.0","True","False","True"]],"shape":{"columns":11,"rows":30}}
```

```
Failure-case enumeration complete.
```

```python
# ============================================================
# Cell 14 — Holdout Validation (Axis Generalization)
# ============================================================
# Purpose:
#   Test whether the leading perturbation direction generalizes
#   across perturbation subsets.
#
# Procedure:
#   - Split perturbations into train / test sets
#   - Learn PC1 on training perturbations only
#   - Project held-out perturbations onto the learned direction
#   - Evaluate alignment with external program proportions
#
# Scope:
#   - No reuse of full-data PC1
#   - No tuning
#   - Evaluation only on held-out perturbations
# ============================================================

from sklearn.model_selection import ShuffleSplit

# ------------------------
# Preconditions
# ------------------------
assert "delta_matrix" in globals(), "delta_matrix not found"
assert "program_props" in globals(), "program_props not found"
assert "adipo" in program_props.columns, "'adipo' column missing"

# ------------------------
# Holdout configuration
# ------------------------
N_SPLITS = 50
```

```
TEST_FRACTION = 0.20

splitter = ShuffleSplit(
    n_splits=N_SPLITS,
    test_size=TEST_FRACTION,
    random_state=RANDOM_STATE,
)

# ------------------------
# Holdout evaluation
# ------------------------
holdout_rhos = []

for train_idx, test_idx in splitter.split(delta_matrix):
    X_train = delta_matrix[train_idx]
    X_test = delta_matrix[test_idx]

    # Fit PCA on training perturbations only
    pca_train = PCA(n_components=1, svd_solver="full",
random_state=RANDOM_STATE)
    pca_train.fit(X_train)

    # Extract and normalize learned direction
    v_train = pca_train.components_[0].astype(np.float64, copy=True)
    v_train /= np.linalg.norm(v_train)

    # Project held-out perturbations
    scores_test = X_test @ v_train

    # External labels (held-out only)
    adipo_test = program_props["adipo"].values[test_idx]

    rho, _ = spearmanr(scores_test, adipo_test)
    holdout_rhos.append(rho)

holdout_rhos = np.array(holdout_rhos)

# ------------------------
# Holdout fingerprint
# ------------------------
HOLDOUT_FINGERPRINT = {
    "n_splits": N_SPLITS,
    "test_fraction": TEST_FRACTION,
    "mean_spearman_rho": float(np.mean(holdout_rhos)),
    "std_spearman_rho": float(np.std(holdout_rhos, ddof=1)),
    "min_spearman_rho": float(np.min(holdout_rhos)),
    "max_spearman_rho": float(np.max(holdout_rhos)),
}

for k, v in HOLDOUT_FINGERPRINT.items():
    print(f"{k:>28s}: {v}")

print("\nHoldout validation complete.")
```

```
                    n_splits: 50
               test_fraction: 0.2
           mean_spearman_rho: 0.9552566280215649
            std_spearman_rho: 0.022125137382360764
            min_spearman_rho: 0.8867089988991884
            max_spearman_rho: 0.9884615384615385

Holdout validation complete.
```

```
# ============================================================
# Cell 15 — Secondary Structure (Residual PCA)
# ============================================================
# Purpose:
#   Characterize structure remaining after removing the PC1
#   component from perturbation effects.
#
# Questions answered:
#   - Is residual variation still low-dimensional?
#   - How much variance remains in PC2 / PC3 of residual space?
#
# Scope:
#   - PCA applied ONLY to residual_matrix
#   - No reuse of original PCA
#   - No biological interpretation
# ============================================================

# ------------------------
# Preconditions
# ------------------------
assert "residual_matrix" in globals(), "residual_matrix not found"
assert residual_matrix.shape == delta_matrix.shape, "Residual shape
mismatch"
assert np.all(np.isfinite(residual_matrix)), "Non-finite values in
residual_matrix"

# ------------------------
# Residual PCA configuration
# ------------------------
N_RESIDUAL_PCS = min(10, n_perts - 1)

resid_pca = PCA(
    n_components=N_RESIDUAL_PCS,
    svd_solver="full",
    random_state=RANDOM_STATE,
)
```

```
# ------------------------
# Fit residual PCA
# ------------------------
resid_pca.fit(residual_matrix)

resid_explained = resid_pca.explained_variance_ratio_
resid_cum_explained = np.cumsum(resid_explained)

# ------------------------
# Sanity checks
# ------------------------
assert resid_explained.ndim == 1
assert resid_cum_explained[-1] <= 1.0 + 1e-6
assert np.all(np.diff(resid_cum_explained) >= 0), "Residual cumulative
variance not monotone"

# ------------------------
# Residual structure fingerprint
# ------------------------
RESIDUAL_PCA_FINGERPRINT = {
    "residual_pc1_var": float(resid_cum_explained[0]),
    "residual_pc2_var": float(resid_cum_explained[1]) if
N_RESIDUAL_PCS >= 2 else None,
    "residual_pc3_var": float(resid_cum_explained[2]) if
N_RESIDUAL_PCS >= 3 else None,
    "residual_pc5_var": float(resid_cum_explained[4]) if
N_RESIDUAL_PCS >= 5 else None,
    "residual_pc10_var": float(resid_cum_explained[9]) if
N_RESIDUAL_PCS >= 10 else None,
}

for k, v in RESIDUAL_PCA_FINGERPRINT.items():
    print(f"{k:>28s}: {v}")

print("\nResidual secondary-structure analysis complete.")
```

```
            residual_pc1_var: 0.0834253586688975
            residual_pc2_var: 0.14604429111779837
            residual_pc3_var: 0.1980013930008015
            residual_pc5_var: 0.2668266609161573
           residual_pc10_var: 0.3889877336923761

Residual secondary-structure analysis complete.
```

```
# ============================================================
# Cell 16 — Artifact Freezing & Reproducibility Contract
# ============================================================
# Purpose:
#   Freeze all critical derived artifacts and record immutable
#   fingerprints so results can be:
```

```python
#     - audited
#     - reproduced
#     - compared across versions (v3 vs v4)
#
# Scope:
#   - No computation
#   - No analysis
#   - No interpretation
# ============================================================

import hashlib
import json

# ------------------------
# Preconditions
# ------------------------
REQUIRED_OBJECTS = [
    "delta_matrix",
    "control_mean",
    "v1",
    "pc1_scores",
    "explained_fraction",
    "residual_norms",
    "failure_df",
    "holdout_rhos",
]

for obj in REQUIRED_OBJECTS:
    assert obj in globals(), f"Missing required artifact: {obj}"

# ------------------------
# Output directory
# ------------------------
ARTIFACT_DIR = PROJECT_ROOT / "artifacts_v4"
ARTIFACT_DIR.mkdir(exist_ok=True)

# ------------------------
# Helper: array fingerprint
# ------------------------
def array_fingerprint(arr: np.ndarray) -> str:
    arr = np.ascontiguousarray(arr)
    h = hashlib.sha256(arr.view(np.uint8)).hexdigest()
    return h

# ------------------------
# Save core numeric artifacts
# ------------------------
np.save(ARTIFACT_DIR / "delta_matrix.npy", delta_matrix)
np.save(ARTIFACT_DIR / "control_mean.npy", control_mean)
np.save(ARTIFACT_DIR / "pc1_direction.npy", v1)
np.save(ARTIFACT_DIR / "pc1_scores.npy", pc1_scores)
np.save(ARTIFACT_DIR / "explained_fraction.npy", explained_fraction)
np.save(ARTIFACT_DIR / "residual_norms.npy", residual_norms)
np.save(ARTIFACT_DIR / "holdout_rhos.npy", holdout_rhos)

failure_df.to_csv(ARTIFACT_DIR / "failure_cases.csv", index=False)

# ------------------------
# Fingerprint registry
# ------------------------
FINGERPRINT_REGISTRY = {
    "delta_matrix_sha256": array_fingerprint(delta_matrix),
    "control_mean_sha256": array_fingerprint(control_mean),
    "pc1_direction_sha256": array_fingerprint(v1),
    "pc1_scores_sha256": array_fingerprint(pc1_scores),
    "explained_fraction_sha256":
array_fingerprint(explained_fraction),
    "residual_norms_sha256": array_fingerprint(residual_norms),
    "holdout_rhos_sha256": array_fingerprint(holdout_rhos),
    "n_perturbations": int(n_perts),
    "n_genes": int(n_genes),
    "random_state": int(RANDOM_STATE),
}

with open(ARTIFACT_DIR / "fingerprints.json", "w") as f:
    json.dump(FINGERPRINT_REGISTRY, f, indent=2, sort_keys=True)

# ------------------------
# Report
# ------------------------
print("Artifacts frozen to:", ARTIFACT_DIR.resolve())
print("\nFingerprint registry:")
for k, v in FINGERPRINT_REGISTRY.items():
    print(f"{k:>32s}: {v}")

print("\nReproducibility contract complete.")
```

```
Artifacts frozen to: C:\Users\Bryan\Documents\CrunchDAO Obesity\
artifacts_v4

Fingerprint registry:
             delta_matrix_sha256:
4cefbaa9e8c2673c4c2d4aa2336816c29602ee091a8c43d43f6919c4b9fa14e7
             control_mean_sha256:
cc7f883b6aab56779e7732a70c2a506f0e54c0d50fb908f585fe3f77c464acb1
            pc1_direction_sha256:
889bf46fdc866ec1967b69d01f104dcdd44d8b035e04a302b885005e066616ac
               pc1_scores_sha256:
64d17545a30f016fd4bf6d6519e82e2443ab77b5e55255fab7879e1dc4695c00
       explained_fraction_sha256:
```

```
a0a791ee11a4c980bd63a7a19a8f6788c85d5c6ff6248433c7d7740aae9a1a5d
           residual_norms_sha256:
f20e3ee775b712cda839ec43ee81a1add2b480d4f9313ddf2259b1a4351c9c52
             holdout_rhos_sha256:
8e4ce7be02cd05617f1cafc1ef8aab8a77b4432176a14dbea20123727b05eee5
                 n_perturbations: 122
                         n_genes: 21592
                    random_state: 42

Reproducibility contract complete.

# ============================================================
# Cell 17 — Environment & Data Provenance Lock
# ============================================================
# Purpose:
#   Record the exact execution environment and input data
#   fingerprints required to reproduce this run.
#
# Scope:
#   - No computation
#   - No analysis
#   - No modification of artifacts
# ============================================================

import platform
import subprocess
from datetime import datetime

# ------------------------
# Timestamp
# ------------------------
RUN_METADATA = {
    "run_timestamp_utc":
datetime.utcnow().isoformat(timespec="seconds") + "Z",
}

# ------------------------
# System information
# ------------------------
SYSTEM_METADATA = {
    "python_version": platform.python_version(),
    "python_implementation": platform.python_implementation(),
    "os": platform.system(),
    "os_release": platform.release(),
    "machine": platform.machine(),
    "processor": platform.processor(),
}

# ------------------------
# Package versions (explicit)
# ------------------------
PACKAGE_METADATA = {
    "numpy": np.__version__,
    "pandas": pd.__version__,
    "scipy": sp.__version__,
    "scanpy": sc.__version__,
    "anndata": ad.__version__,
    "sklearn": pkg_version("scikit-learn"),
}

# ------------------------
# Input data fingerprints
# ------------------------
def file_sha256(path: Path) -> str:
    h = hashlib.sha256()
    with open(path, "rb") as f:
        for chunk in iter(lambda: f.read(8192), b""):
            h.update(chunk)
    return h.hexdigest()

DATA_FINGERPRINTS = {
    "obesity_challenge_1.h5ad_sha256": file_sha256(DATA_DIR /
"obesity_challenge_1.h5ad"),
    "program_proportion.csv_sha256": file_sha256(DATA_DIR /
"program_proportion.csv"),
}

# ------------------------
# Consolidated provenance record
# ------------------------
PROVENANCE_RECORD = {
    **RUN_METADATA,
    **SYSTEM_METADATA,
    **PACKAGE_METADATA,
    **DATA_FINGERPRINTS,
}

# ------------------------
# Persist provenance
# ------------------------
with open(ARTIFACT_DIR / "provenance.json", "w") as f:
    json.dump(PROVENANCE_RECORD, f, indent=2, sort_keys=True)

# ------------------------
# Report
# ------------------------
print("Provenance record written to:")
print((ARTIFACT_DIR / "provenance.json").resolve())

print("\nProvenance summary:")
```

```
for k, v in PROVENANCE_RECORD.items():
    print(f"{k:>36s}: {v}")

print("\nEnvironment and data provenance locked.")
```

```
C:\Users\Bryan\AppData\Local\Temp\ipykernel_21120\2878744532.py:44:
FutureWarning: `__version__` is deprecated, use
`importlib.metadata.version('scanpy')` instead
  "scanpy": sc.__version__,
C:\Users\Bryan\AppData\Local\Temp\ipykernel_21120\2878744532.py:45:
FutureWarning: `__version__` is deprecated, use
`importlib.metadata.version('anndata')` instead.
  "anndata": ad.__version__,

Provenance record written to:
C:\Users\Bryan\Documents\CrunchDAO Obesity\artifacts_v4\
provenance.json

Provenance summary:
                   run_timestamp_utc: 2025-12-17T15:18:44Z
                      python_version: 3.11.9
               python_implementation: CPython
                                  os: Windows
                          os_release: 10
                             machine: AMD64
                           processor: AMD64 Family 23 Model 113
Stepping 0, AuthenticAMD
                               numpy: 2.2.0
                              pandas: 2.3.2
                               scipy: 1.16.1
                              scanpy: 1.11.5
                             anndata: 0.12.6
                             sklearn: 1.7.1
     obesity_challenge_1.h5ad_sha256:
2f7dfa0d2c9d4dd38f9d25b6707a8b681fc2bb99ad73f9ba55d6824bb4570aac
       program_proportion.csv_sha256:
5ef3bc9249c4d5381ee8103bb70a5c0f6a2ffd1b52fc0fcde2f37f08fb92bcf3

Environment and data provenance locked.
```

```
# ==============================================================
# Cell 18 — Provenance Patch (Warning-Free)
# ==============================================================
# Purpose:
#   Regenerate provenance metadata using non-deprecated APIs.
#   This cell exists solely to remove FutureWarnings and does
#   NOT alter any analytical artifacts.
#
# Scope:
#   - No computation
```

```python
#   - No analysis
#   - No artifact modification
# ============================================================

from importlib.metadata import version as pkg_version
from datetime import datetime
import platform
import hashlib
import json

# ------------------------
# Timestamp (patch time)
# ------------------------
RUN_METADATA = {
    "run_timestamp_utc":
datetime.utcnow().isoformat(timespec="seconds") + "Z",
    "provenance_patch": True,
}

# ------------------------
# System information
# ------------------------
SYSTEM_METADATA = {
    "python_version": platform.python_version(),
    "python_implementation": platform.python_implementation(),
    "os": platform.system(),
    "os_release": platform.release(),
    "machine": platform.machine(),
    "processor": platform.processor(),
}

# ------------------------
# Package versions (non-deprecated)
# ------------------------
PACKAGE_METADATA = {
    "numpy": pkg_version("numpy"),
    "pandas": pkg_version("pandas"),
    "scipy": pkg_version("scipy"),
    "scanpy": pkg_version("scanpy"),
    "anndata": pkg_version("anndata"),
    "sklearn": pkg_version("scikit-learn"),
}

# ------------------------
# Input data fingerprints (unchanged)
# ------------------------
def file_sha256(path: Path) -> str:
    h = hashlib.sha256()
    with open(path, "rb") as f:
        for chunk in iter(lambda: f.read(8192), b""):
            h.update(chunk)
    return h.hexdigest()

DATA_FINGERPRINTS = {
    "obesity_challenge_1.h5ad_sha256": file_sha256(DATA_DIR /
"obesity_challenge_1.h5ad"),
    "program_proportion.csv_sha256": file_sha256(DATA_DIR /
"program_proportion.csv"),
}

# ------------------------
# Consolidated provenance record
# ------------------------
PROVENANCE_RECORD = {
    **RUN_METADATA,
    **SYSTEM_METADATA,
    **PACKAGE_METADATA,
    **DATA_FINGERPRINTS,
}

# ------------------------
# Persist patched provenance
# ------------------------
with open(ARTIFACT_DIR / "provenance.json", "w") as f:
    json.dump(PROVENANCE_RECORD, f, indent=2, sort_keys=True)

# ------------------------
# Report
# ------------------------
print("Provenance record updated (warning-free):")
print((ARTIFACT_DIR / "provenance.json").resolve())

print("\nProvenance summary:")
for k, v in PROVENANCE_RECORD.items():
    print(f"{k:>36s}: {v}")

print("\nProvenance patch complete. No deprecated APIs used.")
```

```
Provenance record updated (warning-free):
C:\Users\Bryan\Documents\CrunchDAO Obesity\artifacts_v4\
provenance.json

Provenance summary:
                   run_timestamp_utc: 2025-12-17T15:20:36Z
                    provenance_patch: True
                      python_version: 3.11.9
               python_implementation: CPython
                                  os: Windows
                          os_release: 10
                             machine: AMD64
```

```
                           processor: AMD64 Family 23 Model 113
Stepping 0, AuthenticAMD
                               numpy: 2.2.0
                              pandas: 2.3.2
                               scipy: 1.16.1
                              scanpy: 1.11.5
                             anndata: 0.12.6
                             sklearn: 1.7.1
     obesity_challenge_1.h5ad_sha256:
2f7dfa0d2c9d4dd38f9d25b6707a8b681fc2bb99ad73f9ba55d6824bb4570aac
       program_proportion.csv_sha256:
5ef3bc9249c4d5381ee8103bb70a5c0f6a2ffd1b52fc0fcde2f37f08fb92bcf3

Provenance patch complete. No deprecated APIs used.

# ==============================================================
# Cell 19 — Self-Audit & Integrity Verification
# ==============================================================
# Purpose:
#   Verify that all persisted artifacts match their recorded
#   cryptographic fingerprints exactly.
#
# This is a forensic self-audit. Any mismatch aborts execution.
#
# Scope:
#   - No analysis
#   - No recomputation
#   - No modification of artifacts
# ==============================================================

import json
import hashlib

# ------------------------
# Preconditions
# ------------------------
assert ARTIFACT_DIR.exists(), "Artifact directory missing"
assert (ARTIFACT_DIR / "fingerprints.json").exists(), "Fingerprint
registry missing"

# ------------------------
# Load fingerprint registry
# ------------------------
with open(ARTIFACT_DIR / "fingerprints.json", "r") as f:
    fingerprint_registry = json.load(f)

# ------------------------
# Helper: recompute array hash
# ------------------------
def verify_array(path: Path, expected_hash: str):
    arr = np.load(path)
    arr = np.ascontiguousarray(arr)
    h = hashlib.sha256(arr.view(np.uint8)).hexdigest()
    assert h == expected_hash, f"Hash mismatch for {path.name}"
    return h

# ------------------------
# Verify numeric artifacts
# ------------------------
verified_hashes = {}

verified_hashes["delta_matrix"] = verify_array(
    ARTIFACT_DIR / "delta_matrix.npy",
    fingerprint_registry["delta_matrix_sha256"],
)

verified_hashes["control_mean"] = verify_array(
    ARTIFACT_DIR / "control_mean.npy",
    fingerprint_registry["control_mean_sha256"],
)

verified_hashes["pc1_direction"] = verify_array(
    ARTIFACT_DIR / "pc1_direction.npy",
    fingerprint_registry["pc1_direction_sha256"],
)

verified_hashes["pc1_scores"] = verify_array(
    ARTIFACT_DIR / "pc1_scores.npy",
    fingerprint_registry["pc1_scores_sha256"],
)

verified_hashes["explained_fraction"] = verify_array(
    ARTIFACT_DIR / "explained_fraction.npy",
    fingerprint_registry["explained_fraction_sha256"],
)

verified_hashes["residual_norms"] = verify_array(
    ARTIFACT_DIR / "residual_norms.npy",
    fingerprint_registry["residual_norms_sha256"],
)

verified_hashes["holdout_rhos"] = verify_array(
    ARTIFACT_DIR / "holdout_rhos.npy",
    fingerprint_registry["holdout_rhos_sha256"],
)

# ------------------------
# Verify tabular artifact
# ------------------------
assert (ARTIFACT_DIR / "failure_cases.csv").exists(), "Failure cases
```

```
CSV missing"

# ------------------------
# Report
# ------------------------
print("Artifact integrity verification passed.\n")

for k, v in verified_hashes.items():
    print(f"{k:>22s}: {v}")

print("\nAll persisted artifacts match recorded fingerprints
exactly.")
print("Self-audit complete.")
```

```
Artifact integrity verification passed.

          delta_matrix:
4cefbaa9e8c2673c4c2d4aa2336816c29602ee091a8c43d43f6919c4b9fa14e7
          control_mean:
cc7f883b6aab56779e7732a70c2a506f0e54c0d50fb908f585fe3f77c464acb1
         pc1_direction:
889bf46fdc866ec1967b69d01f104dcdd44d8b035e04a302b885005e066616ac
            pc1_scores:
64d17545a30f016fd4bf6d6519e82e2443ab77b5e55255fab7879e1dc4695c00
    explained_fraction:
a0a791ee11a4c980bd63a7a19a8f6788c85d5c6ff6248433c7d7740aae9a1a5d
        residual_norms:
f20e3ee775b712cda839ec43ee81a1add2b480d4f9313ddf2259b1a4351c9c52
          holdout_rhos:
8e4ce7be02cd05617f1cafc1ef8aab8a77b4432176a14dbea20123727b05eee5

All persisted artifacts match recorded fingerprints exactly.
Self-audit complete.
```

```
# ============================================================
# Cell 20 — Claims & Non-Claims Ledger (Scope Lock)
# ============================================================
# Purpose:
#   Record the exact scientific claims supported by this
#   notebook, alongside explicit non-claims.
#
# This cell prevents post-hoc overstatement and reviewer drift.
#
# Scope:
#   - No computation
#   - No analysis
#   - No modification of artifacts
# ============================================================

import json
```

```
# ------------------------
# Claims supported by this analysis
# ------------------------
CLAIMS = [
    "Perturbation-induced gene expression effects concentrate along a
dominant geometric direction (PC1).",
    "This direction is statistically exceptional relative to random
and PC1-orthogonal directions.",
    "Projection onto this direction aligns strongly with independently
estimated adipogenesis programs.",
    "Positive movement along this direction is necessary but not
sufficient for high adipogenesis.",
    "The direction generalizes across perturbation subsets under
holdout validation.",
    "Substantial residual structure exists, indicating additional
mechanisms beyond PC1.",
]

# ------------------------
# Explicit non-claims
# ------------------------
NON_CLAIMS = [
    "No causal relationship between genes and adipogenesis is
inferred.",
    "No mechanistic regulatory network is identified.",
    "No predictive model for individual cells is proposed.",
    "No claim is made that PC1 fully explains perturbation effects.",
    "No biological interpretation of residual components is
asserted.",
]

# ------------------------
# Scope metadata
# ------------------------
CLAIM_SCOPE = {
    "analysis_version": "v4",
    "intended_use": "Descriptive, geometric analysis of perturbation
effects",
    "unit_of_analysis": "Perturbation-level mean expression effects",
    "labels_usage": "Evaluation only; never used for representation
construction",
}

# ------------------------
# Persist ledger
# ------------------------
CLAIM_LEDGER = {
    "claims": CLAIMS,
    "non_claims": NON_CLAIMS,
    "scope": CLAIM_SCOPE,
}

with open(ARTIFACT_DIR / "claims_ledger.json", "w") as f:
    json.dump(CLAIM_LEDGER, f, indent=2, sort_keys=True)

# ------------------------
# Report
# ------------------------
print("Claims & Non-Claims ledger written to:")
print((ARTIFACT_DIR / "claims_ledger.json").resolve())

print("\nClaims:")
for c in CLAIMS:
    print(f"  - {c}")

print("\nExplicit non-claims:")
for nc in NON_CLAIMS:
    print(f"  - {nc}")

print("\nClaim scope locked.")
```

```
Claims & Non-Claims ledger written to:
C:\Users\Bryan\Documents\CrunchDAO Obesity\artifacts_v4\
claims_ledger.json

Claims:
  - Perturbation-induced gene expression effects concentrate along a
dominant geometric direction (PC1).
  - This direction is statistically exceptional relative to random and
PC1-orthogonal directions.
  - Projection onto this direction aligns strongly with independently
estimated adipogenesis programs.
  - Positive movement along this direction is necessary but not
sufficient for high adipogenesis.
  - The direction generalizes across perturbation subsets under
holdout validation.
  - Substantial residual structure exists, indicating additional
mechanisms beyond PC1.

Explicit non-claims:
  - No causal relationship between genes and adipogenesis is inferred.
  - No mechanistic regulatory network is identified.
  - No predictive model for individual cells is proposed.
  - No claim is made that PC1 fully explains perturbation effects.
  - No biological interpretation of residual components is asserted.

Claim scope locked.
```

```
# ============================================================
# Cell 21b — README Freeze (Syntax-Safe Patch)
# ============================================================
# Purpose:
#   Regenerate README.txt using syntax-safe string construction.
#   This cell exists solely to correct a formatting error.
#
# Scope:
#   - No computation
#   - No analysis
#   - No artifact modification (README only)
# ============================================================

# ------------------------
# Preconditions
# ------------------------
assert ARTIFACT_DIR.exists(), "Artifact directory missing"

# ------------------------
# README content (literal string, no f-string, no chaining)
# ------------------------
README_TEXT = (
    "CrunchDAO Obesity — Geometric Perturbation Analysis (v4)\n"
    "==========================================================\n\n"
    "Overview\n"
    "--------\n"
    "This directory contains frozen artifacts produced by a fully\n"
    "deterministic, perturbation-level geometric analysis of gene\n"
    "expression responses in the CrunchDAO Obesity Challenge dataset.\
n\n"
    "No model fitting, prediction, or causal inference is performed.\
n"
    "All results are descriptive and evaluated at the perturbation 
level.\n\n"
    "What Was Done\n"
    "-------------\n"
    "1. Constructed perturbation-level mean expression deltas\n"
    "   relative to a fixed control reference.\n"
    "2. Identified a dominant geometric direction (PC1) in gene 
space.\n"
    "3. Falsified this direction against random and orthogonal nulls.\
n"
    "4. Evaluated alignment with external adipogenesis programs.\n"
    "5. Demonstrated necessity (not sufficiency) via constraint 
testing.\n"
    "6. Quantified residual structure and explicit failure cases.\n"
    "7. Verified generalization via holdout validation.\n"
    "8. Froze artifacts, provenance, and claims.\n\n"
    "What Was NOT Done\n"
    "-----------------\n"
    "- No causal claims\n"
    "- No regulatory network inference\n"
    "- No single-cell prediction\n"
    "- No biological interpretation beyond geometry\n\n"
    "Reproducibility\n"
    "---------------\n"
    "All numerical artifacts are fingerprinted with SHA-256 hashes.\n"
    "Exact package versions, system information, and input data
hashes\n"
    "are recorded in provenance.json.\n\n"
    "Integrity\n"
    "---------\n"
    "Artifacts were reloaded from disk and verified against their\n"
    "recorded fingerprints. See execution_manifest.json for details.\
n"
)

# ------------------------
# Write README
# ------------------------
readme_path = ARTIFACT_DIR / "README.txt"
with open(readme_path, "w", encoding="utf-8") as f:
    f.write(README_TEXT)

# ------------------------
# Report
# ------------------------
print("README regenerated successfully:")
print(readme_path.resolve())
print("\nREADME patch complete. No syntax warnings possible.")
```

```
README regenerated successfully:
C:\Users\Bryan\Documents\CrunchDAO Obesity\artifacts_v4\README.txt

README patch complete. No syntax warnings possible.
```