Local-Interaction Learning Dynamics

A Markov Random Field View of Deep Learning

Wen Dong · **NeurIPS 2026** · GitHub repository · Paper · Executed notebook · PDF reading copy · Citation

Why can training remain numerically stable while ceasing to learn? Which network interfaces amplify or suppress useful signals? Which fluctuations of the weights actually affect predictions? This tutorial connects these questions through local interactions between parameters and intermediate network states.

The paper's equations are followed by their mathematical implementation, executed examples, and recorded checkpoint diagnostics. The main capabilities are exposing architecture in a sparse precision matrix, distinguishing stability from active response, locating signal attenuation, and reconstructing a split-certified PAC-Bayes table.

One Python tutorial, original experimental implementations. Mathematical kernels and saved-data calculations use NumPy/SciPy. The split-certified models and precision builders use TensorFlow; the criticality vision/GPT checkpoint paths use PyTorch. These are the original experimental paths, not interchangeable backend ports.

Run the default workflow. From the repository root, install `requirements.txt`, then execute this notebook or run `python reproduce_main_artifacts.py`. This checks small identities, recomputes Table 1 from saved posterior evaluations, and redraws diagnostic curves from saved measurements. Training, checkpoint replay, and architecture re-extraction are explicit opt-ins. No checkpoint or dataset download is needed by default.

The architecture previews are retained paper figures; their TensorFlow builders are linked alongside them. Every empirical plot identifies its saved inputs. Code needed to follow an equation is expanded; setup and plotting orchestration are collapsible in the HTML reading copy.

What the computations establish

A network is represented by trainable parameters θ and intermediate states u. Local module relations, loss, and regularization define the lifted energy $E(\theta, u)$ and a checkpoint-local precision. The same representation gives three complementary calculations.

Calculation	Mathematical object	What the output tells us
Activation elimination	Schur complement of a positive quadratic	How local state constraints induce parameter curvature
Criticality and boundary gains	Sampled gradient Grams and loss adjoints	Whether the measured update is stable and active, and where sensitivity changes
Distribution geometry and certificates	Inverse-metric drift defect; separately, exact local Gaussian prior KL	Which stochastic directions affect predictions, and a certified bounded-risk table

Activation elimination and the hard-constraint limit

For a positive lifted quadratic with blocks $P = \begin{pmatrix} A & B \\ B^\top & D \end{pmatrix}$, stationarity in the state displacement gives $\Delta u_* = -D^{-1}B^\top \Delta\theta$. Substitution yields $F_{\kappa,\tau} = A - BD^{-1}B^\top$. This is the **softened** curvature at

finite module precision κ and state ridge τ. Theorem 1 recovers deterministic-network Fisher/Gauss–Newton curvature in its stated hard-constraint limit; finite precision alone is not enough.

For example, $E = \kappa(u-\theta)^2/2 + u^2/2$ gives $F_\kappa = \kappa/(\kappa+1)$, approaching 1 as $\kappa \to \infty$. The small check below verifies both the finite value and the limit.

Criticality and measured proxies

For a quadratic gradient step in a fixed preconditioned coordinate, $\bar{F} = P^{1/2} F_{\mathrm{red}} P^{1/2}$, the upper edge is $\eta_{\max} = 2/\lambda_{\max}(\bar{F})$. An active-space coordinate $\rho_{\mathrm{act}} = |\eta_{\mathrm{eff}}| \lambda_{\min}(\bar{F}|_{\iota_{\mathrm{act}}})$ also asks whether loss-changing directions respond. The experiments use observed-gradient Gram proxies and an optimizer-state scalar projection. These are checkpoint diagnostics, rather than a full spectral stability certificate for Momentum or AdamW.

At adjacent retained boundaries, $A_\ell = \partial L/\partial h_\ell$ gives the measured directional gain $b_\ell = \|A_\ell\|_2 / \max(\|A_{\ell+1}\|_2, \epsilon)$. This localizes attenuation or amplification in the observed loss direction; it is not the separator operator norm itself.

Two distinct distributional calculations

On a fixed comparison space, the drift defect is $D(\beta) = \sum_i (r_i - \beta\lambda_i q_i/2)^2/\lambda_i$. Theorem 4 bounds KL-to-Gibbs under its contraction and diffusion assumptions. The camera-ready Table 1 instead uses an explicit, build-dependent Gaussian prior and posterior with exact $\mathrm{KL}(Q\|P)$ and an independent certificate sample. A frozen checkpoint defect is not substituted for that prior KL.

1. Reproduction and mathematical checks

The default path is CPU-only and reads the included evidence. Numerical calculations are rerun; expensive trained-network evaluations are reused. `verify_core_identities.py` checks Schur elimination, the hard-constraint limit, Gram spectra and eigenvector lifting, centering, defect expansion, KL integration, and Gaussian certificate terms. Deliberately wrong finite-precision and missing-MC calculations are rejected. These finite examples check implementations of the identities; they do not establish the statistical assumptions of a training run.

```
import subprocess
result = subprocess.run([sys.executable, 'verify_core_identities.py'],
                        check=True, text=True, capture_output=True)
print(result.stdout)
import core_math_identities as math_core
show_source(math_core.schur_curvature)
for kappa in (1., 10., 100., 10000.):
    curvature = math_core.schur_curvature(np.array([[kappa]]),
                np.array([[-kappa]]), np.array([[kappa + 1.]]))[0, 0]
    print(f'kappa={kappa:8.0f}: softened curvature={curvature:.8f}; hard limit=1')
```

```
[OK] Schur complement elimination
[OK] Gram-spectrum equivalence
[OK] retained-defect quadratic expansion
[OK] finite-time KL recursion
[OK] finite precision versus hard-constraint limit; negative control
[OK] covariance centering, eigenvector lift, and zero-contraction limit
[OK] same-support Gaussian KL and nonzero Monte Carlo correction
All core identity checks passed.
```

Implementation: schur_curvature

```python
def schur_curvature(parameter_precision, cross_precision, state_precision):
    """Minimize a Gaussian quadratic over its state displacement.

    For blocks (A, B; B.T, D), stationary states solve D du = -B.T dtheta.
    Substitution gives dtheta.T (A - B solve(D,B.T)) dtheta / 2.
    D must be positive definite here; no silent pseudoinverse is substituted.
    At finite module precision this is softened curvature. Recovering the
    deterministic-network curvature additionally requires the hard-constraint
    limit stated in Theorem 1.
    """
    np.linalg.cholesky(state_precision)
    return parameter_precision - cross_precision @ np.linalg.solve(
        state_precision, cross_precision.T)

kappa=       1: softened curvature=0.50000000; hard limit=1
kappa=      10: softened curvature=0.90909091; hard limit=1
kappa=     100: softened curvature=0.99009901; hard limit=1
kappa=   10000: softened curvature=0.99990001; hard limit=1
```

Checkpoints and the training switch

The original checkpoint inventory links model families, diagnostic inputs, and training entry points. The default reports the inventory and skips training. `RUN_STAGE1_LONG_JOBS=1` is reserved for explicitly requested long jobs; select individual jobs rather than launching every experiment.

Training reference examples and the exact certificate producer are distinguished in Section 8. The revised five-model Table 1 is reconstructed from its frozen July 31 records, not from the earlier compact training examples. Optional checkpoint replay requires the original weights, optimizer state when used by the diagnostic, and the matching data/configuration. Missing files stop that replay; they never trigger automatic training.

2. Criticality coordinates from recorded checkpoints

Stage 2A is the checkpoint-to-criticality-coordinate path. It consumes saved checkpoints or bundled checkpoint summaries and writes one row per checkpoint with the quantities used in the main two-sided criticality figure. The row contains an **upper-side stability coordinate** and a **lower-side active-plasticity coordinate**.

The code displayed immediately below is the mathematical kernel. It excludes checkpoint loading, optimizer-state parsing, CSV writing, and plotting. The application cell after it reads checkpoints when available, otherwise uses packaged paper-data summaries.

1. Actual optimizer-state scalar step

At checkpoint t, let θ_t be the parameters, g_t the current training gradient in the coordinate being tested, and $\Delta\theta_t$ the next optimizer update reconstructed from the saved optimizer state. The scalar step used in the upper-band comparison is

$$\eta_{\parallel,t} = -\frac{\Delta\theta_t^\top g_t}{\|g_t\|_2^2}.$$

This scalar is the component of the actual update parallel to the chosen gradient; an orthogonal update component may remain. For SGD with momentum, $\Delta\theta_t$ is reconstructed from the momentum buffer. For AdamW, it is reconstructed from the first- and second-moment buffers. The helper `effective_learning_rate_parallel` implements exactly this projection.

2. Upper-edge curvature from a Gram matrix

The dense reduced curvature proxy is not formed. With m probe gradients stacked as

$$G_t = \begin{bmatrix} g_{t,1}^\top \\ \vdots \\ g_{t,m}^\top \end{bmatrix} \in \mathbb{R}^{m \times p},$$

the observed-gradient outer-product proxy is

$$\hat{F}_t = \frac{1}{m} G_t^\top G_t.$$

The nonzero eigenvalues of $\hat{F}_t$ equal those of the smaller row Gram matrix

$$\widehat{K}_t = \frac{1}{m} G_t G_t^\top \in \mathbb{R}^{m \times m}.$$

The measured scalar edge proxy is therefore

$$\hat{\eta}_{\mathrm{edge},t} = \frac{2}{\lambda_{\max}(\widehat{K}_t)}.$$

The helper `gram_top_eigenvalue` implements this row-Gram computation. Its `center` flag controls whether the rows are centered. In the upper-edge coordinate, the paper-facing vision path uses the uncentered PSD probe in the relevant optimizer coordinate; centered variants are diagnostic and used elsewhere in the supplement.

For SGD-with-momentum vision runs, the coordinate table also reports the scalar heavy-ball adjustment used by the paper figures:

$$\eta_{\text{momentum edge},t} = (1+\mu)\frac{2}{\lambda_{\max}}, \qquad \rho_{\mathrm{upper},t} = \frac{|\eta_{\mathrm{eff,t}}|}{\eta_{\text{momentum edge},t}}.$$

For AdamW/GPT, $\eta_{\mathrm{eff,t}}$ is already reconstructed from the optimizer state, so the coordinate is reported directly in the measured state-effective units.

3. Active subspace and lower-side plasticity

The lower side is not a global smallest eigenvalue and not a lower learning-rate floor. It asks whether the directions currently used by the optimizer still have loss-changing curvature.

Let $U_t \in \mathbb{R}^{p \times r}$ have orthonormal columns spanning a small active subspace, typically built from recent or probe gradient directions. Given a probe-gradient matrix G_t, define the centered version

$$X_t = G_t - \mathbf{1}\bar{g}_t^\top, \qquad \bar{g}_t = \frac{1}{m}\sum_{s=1}^{m} g_{t,s}.$$

The centered active-space proxy used in this implementation is

$$\hat{F}_{\mathrm{act},t} = U_t^\top \left(\frac{1}{m-1} X_t^\top X_t \right) U_t = \frac{1}{m-1} (X_t U_t)^\top (X_t U_t).$$

The helper `active_subspace_lower_eigenvalue` computes

$$\lambda_{\mathrm{act},t} = \lambda_{\min}(\hat{F}_{\mathrm{act},t}).$$

The paper-facing active-plasticity coordinate is

$$\rho_{\mathrm{act},t} = |\eta_{\mathrm{eff},t}| \lambda_{\mathrm{act},t}.$$

The helper `criticality_coordinates` assembles the upper edge, actual effective step, upper coordinate, active lower eigenvalue, and active-plasticity coordinate into the table columns consumed by Stage 4A.

4. Checkpoint replay paths

The application cell uses these math kernels in two data paths.

For vision checkpoints, `recompute_vision_criticality_coordinates_from_checkpoints` restores the paper models, optimizer states, and selected epochs, then builds the upper-edge and active-subspace rows. The vision checkpoint path includes coupled weight decay in the SGD gradient because the saved optimizer update is compared to the same coupled-L2 field.

For GPT checkpoints, `collect_gpt_criticality_coordinates_from_available_artifacts(..., require_raw_checkpoints=True)` restores raw state-complete AdamW checkpoints and reconstructs the actual state-effective update. If raw checkpoint files are absent, the long replay path fails rather than silently replacing the computation by aggregate paper metrics. The quick path uses bundled paper-data summaries.

Implementation: gram_top_eigenvalue

```python
def gram_top_eigenvalue(
    gradient_rows: np.ndarray,
    denom: int | None = None,
    center: bool = False,
) -> float:
    """Top eigenvalue of the empirical Fisher/GGN row Gram matrix.

    Math:
        G = [g_1^T; ...; g_M^T],
        X = G - 1 mean(G)^T if center else G,
        K = X X^T / M,
        lambda_max(F) = lambda_max(K),

    because F = X^T X / M and K have the same nonzero eigenvalues.
    """

    G = np.asarray(gradient_rows, dtype=float)
    if G.ndim != 2 or G.shape[0] == 0:
        raise ValueError("gradient_rows must have shape (M, d) with M > 0")
    M = int(denom or G.shape[0])
    X = G - G.mean(axis=0, keepdims=True) if center and G.shape[0] > 1 else G
    K = (X @ X.T) / max(1, M)
    K = 0.5 * (K + K.T)
    eigs = np.maximum(np.linalg.eigvalsh(K), 0.0)
    return float(eigs.max() if eigs.size else 0.0)
```

Implementation: active_subspace_lower_eigenvalue

```python
def active_subspace_lower_eigenvalue(
    gradient_rows: np.ndarray,
    active_basis: np.ndarray,
    denom: int | None = None,
    center: bool = True,
) -> float:
    """Smallest empirical-Fisher eigenvalue on a chosen active subspace.

    Math:
        Q^T Q = I,
        X = G - 1 mean(G)^T if center else G,
        F = X^T X / M,
        F_Q = Q^T F Q = (X Q)^T (X Q) / M,
        lambda_active = lambda_min(F_Q).
    """

    G = np.asarray(gradient_rows, dtype=float)
    Q = np.asarray(active_basis, dtype=float)
    if G.ndim != 2 or Q.ndim != 2 or G.shape[1] != Q.shape[0]:
        raise ValueError("expected G shape (M,d) and Q shape (d,r)")
    X = G - G.mean(axis=0, keepdims=True) if center and G.shape[0] > 1 else G
    M = int(denom or (X.shape[0] - int(center and X.shape[0] > 1)))
    GQ = X @ Q
    F_Q = (GQ.T @ GQ) / max(1, M)
    F_Q = 0.5 * (F_Q + F_Q.T)
    eigs = np.maximum(np.linalg.eigvalsh(F_Q), 0.0)
    return float(eigs.min() if eigs.size else 0.0)
```

Implementation: effective_learning_rate_parallel

```python
def effective_learning_rate_parallel(update_delta: np.ndarray, gradient: np.ndarray) -> float:
    """Signed scalar eta satisfying update_delta ~= -eta * gradient.

    Math:
        eta_eff_parallel = - <Delta theta, g> / ||g||^2.

    This converts an optimizer-state update, including momentum or AdamW, into
    the scalar learning-rate coordinate comparable to an upper stability edge.
    """

    delta = np.asarray(update_delta, dtype=float).reshape(-1)
    grad = np.asarray(gradient, dtype=float).reshape(-1)
    return float(-np.dot(delta, grad) / max(np.dot(grad, grad), EPS))
```

Stage 2A application: checkpoint-to-coordinate computation

Vision criticality coordinates: first eight records

Architecture	Condition	Epoch	Upper	Active
DenseNet-BC	densenet baseline	0	0.01666	0.00049
DenseNet-BC	densenet baseline	1	0.01380	1.13518
DenseNet-BC	densenet baseline	3	0.01522	0.77639
DenseNet-BC	densenet baseline	8	0.01188	0.95754
DenseNet-BC	densenet baseline	16	0.01659	1.43475
DenseNet-BC	densenet baseline	32	0.01484	0.94055
ResNet-18	resnet baseline	0	0.12227	0.22682
ResNet-18	resnet baseline	1	0.03279	0.61005

GPT criticality coordinates: first eight records

Seed	Step	Tokens	Upper	Active
0	0	0	0.00561	0.00372
0	10	81920	0.01315	0.00880
0	25	204800	0.01045	0.00449
0	50	409600	0.01703	0.00527
0	100	819200	0.02896	0.01050
0	200	1638400	0.02760	0.04081
0	500	4096000	0.06129	0.07069
0	6103	49995776	0.15907	0.18264

3. Backward-gain diagnostics from retained boundaries

Stage 2B supports the separator-transfer mechanism. The criticality theorem says that architecture changes trainability by changing how perturbation and backward-sensitivity signal passes through interfaces. The code here measures a checkpoint-local directional version of that transport.

1. Boundary adjoints and local gains

At checkpoint θ_j, choose architecture-native retained boundaries

$$h_0(\theta_j; x), h_1(\theta_j; x), \dots, h_L(\theta_j; x),$$

such as an input boundary, residual block outputs, DenseNet transition outputs, GPT block outputs, or SSM scan/state boundaries. For a probe minibatch $\mathcal{B}_j$, define the training loss

$$\mathcal{L}_j(\theta_j) = \frac{1}{|\mathcal{B}_j|} \sum_{(x,y)\in\mathcal{B}_j} \ell(y, f_{\theta_j}(x)),$$

and retain the boundary adjoints

$$A_{j,\ell} = \frac{\partial \mathcal{L}_j}{\partial h_\ell}.$$

The local backward gain is

$$b_{j,\ell} = \frac{\|A_{j,\ell}\|_2}{\max\{\|A_{j,\ell+1}\|_2, \varepsilon\}}.$$

The helper `boundary_gradient_l2_norm` computes $\|A_{j,\ell}\|_2$, and `adjacent_backward_gain` computes the adjacent ratio. A gain near one means near-neutral transport in the measured direction; sustained gains above or below one indicate amplification or attenuation.

2. Relation to separator transfers

The theorem-level separator transfer for eliminating internal variables I from a clique C to a separator S can be written schematically as

$$T_{C\to S} = Q_{SI}^{\mathrm{tot}}(Q_{II}^{\mathrm{tot}})^{-1}, \qquad g_{C\to S} = \|T_{C\to S}\|_{\mathrm{op}}.$$

The measured $b_{j,\ell}$ is not a full operator-norm proof of $g_{C\to S}$. It is the checkpoint-local directional VJP proxy in the actual loss direction. It answers the operational question behind the theorem: where does the architecture preserve, magnify, or damp the signal that later appears in the reduced Jacobian and curvature coordinates?

A path gain can also be summarized as

$$B_{j,\mathrm{path}} = \prod_{\ell=0}^{L-1} b_{j,\ell} = \exp\left(\sum_{\ell=0}^{L-1} \log b_{j,\ell}\right),$$

but the paper-facing figure emphasizes layer/block-local gains because they localize which architectural interface is responsible.

3. Implementation map

The displayed source functions are the math-facing part:

- `boundary_gradient_l2_norm`: computes the L2 norm of a retained boundary adjoint.
- `adjacent_backward_gain`: computes the adjacent gain ratio.
- `compact_vision_backward_gain_rows`: shows the checkpoint replay pattern for vision models: register forward hooks at retained modules, retain gradients, backpropagate cross-entropy, read the boundary adjoints, and emit one row per adjacent boundary.

Longer functions such as `compute_vision_checkpoint_backward_gains_from_checkpoints`, `collect_gpt_checkpoint_backward_gains`, and table-normalization helpers are execution plumbing and are not source-dumped in the notebook. The Stage 2B application cells read or regenerate the raw and summary gain tables consumed by Stage 4B.

Implementation: boundary_gradient_l2_norm

```python
def boundary_gradient_l2_norm(boundary_gradient: np.ndarray) -> float:
    """L2 norm of the adjoint at a retained architecture boundary.

    Math:
        A_l = dL / dh_l,
        r_l = ||A_l||_2 = sqrt(sum_j A_{l,j}^2).
    """

    A = np.asarray(boundary_gradient, dtype=float)
    return float(np.sqrt(np.sum(A * A))) if A.size else float("nan")
```

Implementation: adjacent_backward_gain

```python
def adjacent_backward_gain(left_norm: float, right_norm: float, tiny: float = EPS) -> float:
    """Local backward gain between adjacent retained boundaries.

    Math:
        gamma_l = r_l / max(r_{l+1}, tiny).
    """

    return float(float(left_norm) / max(float(right_norm), float(tiny)))
```

Vision backward-gain summary

family	from_boundary	to_boundary	Gain	Checkpoints
densenet	input	conv0	2.91152	7
densenet	conv0	features.0	1.70606	7
densenet	features.0	features.1	1.08907	7
densenet	features.1	features.2	1.03666	7

GPT backward-gain summary

label	block	Gain	Checkpoints
RMSNorm	b0	5.12215	24
RMSNorm	b1	1.58518	24
RMSNorm	b2	1.31490	24
RMSNorm	b3	1.24116	24

SSM backward-gain summary

label	block	Gain	Checkpoints
baseline	ssm0	4.95339	6
baseline	ssm1	1.39681	6
baseline	ssm2	1.35686	6
baseline	ssm3	1.32721	6

4. Resolved KL dynamics and the split-certified PAC-Bayes table

4.1 The fixed-space KL-to-Gibbs calculation

Fix a comparison space E and positive diffusion eigenvalues λ_i on it. Let r_i be the drift-mismatch coordinate at $\beta = 0$, and q_i the bounded-loss score coordinate. Expanding each square gives

$$D_E(\beta) = \sum_{i \in E} \frac{(r_i - \beta\lambda_i q_i/2)^2}{\lambda_i} = d_0 + \beta d_1 + \beta^2 d_2, \qquad d_0 = \sum_i \frac{r_i^2}{\lambda_i}, \quad d_1 = -\sum_i r_i q_i, \quad d_2 = \tfrac{1}{4}\sum_i \lambda_i q_i^2.$$

The inverse factor explains why weakly fluctuating directions can dominate this metric. It does not, by itself, establish that such directions matter for prediction.

Under the theorem's fixed-space diffusion and contraction assumptions, an envelope satisfies $\dot{K} \le -\alpha K + D/(2\varepsilon)$. Multiplication by the integrating factor $e^{\alpha t}$ and integration over an interval gives

$$K(t + \Delta t) \le e^{-\alpha \Delta t} K(t) + \frac{1 - e^{-\alpha \Delta t}}{2\varepsilon\alpha} D.$$

For $\alpha \to 0$, the forcing term is $D\Delta t/(2\varepsilon)$. The code also integrates piecewise-varying coefficients. A numerical checkpoint estimate of D is not automatically an upper bound along the entire path, and an assumed contraction rate is not an empirically established log-Sobolev constant.

4.2 The independent build and certificate samples

Table 1 uses a separate static construction. Conditional on build data, freeze the checkpoint $\hat{\theta}$, orthonormal retained basis $V \in \mathbb{R}^{p\times k}$, and predictor $h_w(x) = f_{\hat{\theta}+Vw}(x)$. Freeze $Q = \mathcal{N}(0, \sigma_Q^2 I_k)$ and $P = \mathcal{N}(0, \sigma_P^2 I_k)$ with $\sigma_P = 2\sigma_Q$ and $\sigma_Q = 5 \times 10^{-4}\|\hat{\theta}\|_2/\sqrt{k}$ (on the selected parameter blocks). Both laws have the same support.

KL derivation. In k dimensions, $\log(q/p) = k\log(\sigma_P/\sigma_Q) + \|w\|^2(\sigma_P^{-2} - \sigma_Q^{-2})/2$. Taking $\mathbb{E}_Q\|w\|^2 = k\sigma_Q^2$ yields

$$\mathrm{KL}(Q\|P) = \frac{k}{2}\left[\frac{\sigma_Q^2}{\sigma_P^2} - 1 + \log\frac{\sigma_P^2}{\sigma_Q^2}\right] = \frac{k}{2}\left[\frac{1}{4} - 1 + \log 4\right].$$

This is an exact Gaussian KL, not a frozen drift-defect proxy.

The loss is $\ell(\theta; x, y) = \min\{-\log p_\theta(y \mid x), 5\}/5 \in [0, 1]$. MNIST uses 50,000 build and 10,000 certificate examples; CIFAR-10 uses 40,000 build and 10,000 certificate examples. The split seed is `20260731`. Test examples are audit-only and were accessed after all certificate rows were finalized.

4.3 Statistical terms and their calculation

For posterior draw $w_s \sim Q$, let $r_s = n^{-1}\sum_{i=1}^n \ell(h_{w_s}; x_i, y_i)$. With $S = 4096$, $\bar{r} = S^{-1}\sum_s r_s$, and unbiased across-draw variance $\hat{v}$, the one-sided empirical-Bernstein correction is

$$\epsilon_{\mathrm{MC}} = \sqrt{\frac{2\hat{v}\log(2/\delta_{\mathrm{MC}})}{S}} + \frac{7\log(2/\delta_{\mathrm{MC}})}{3(S-1)}.$$

The variance here is across **network draws**, each evaluated on the complete certificate sample; it is not a variance over examples substituted into this formula.

For fixed $\lambda > 0$, the bounded-loss exponential inequality and change of measure give $R(Q) \le \widehat{R}(Q) + [\mathrm{KL}(Q\|P) + \log(1/\delta_{\mathrm{PB}})]/\lambda + \lambda/(8n)$ with the stated confidence. Conditional on build data, Q and its KL are fixed, so $\lambda = \sqrt{8n[\mathrm{KL}(Q\|P) + \log(1/\delta_{\mathrm{PB}})]}$ can be chosen before certificate access. Substitution and the MC upper bound yield

$$R(Q) \le \bar{r} + \epsilon_{\mathrm{MC}} + \sqrt{\frac{\mathrm{KL}(Q\|P) + \log(1/\delta_{\mathrm{PB}})}{2n}}.$$

Each row uses $\delta_{\mathrm{PB}} = 0.008$ and $\delta_{\mathrm{MC}} = 0.002$; a union bound over five rows gives simultaneous confidence at least $1 - 5(0.008 + 0.002) = 0.95$. Independence from the certificate sample is conditional on build data; averaging this conditional guarantee gives the joint statement.

The saved evaluations allow every table term to be recalculated without retraining. The bound concerns posterior-averaged normalized clipped NLL, not classification error or the deterministic checkpoint. The accompanying test risk is a separate held-out measurement.

Implementation: defect_coefficients

```
def defect_coefficients(lambdas: np.ndarray, u_coords: np.ndarray, z_coords: np.ndarray) -> DefectCoefficients:
    """Compute d0,d1,d2 from retained eigen-coordinates.

    For retained eigenpair (lambda_i, v_i), define:
        u_i = <u, v_i>,
        z_i = <z, v_i>.
```

```
    The retained inverse-metric defect is:
        D_tau(beta)
          = sum_i (u_i - beta lambda_i z_i / 2)^2 / lambda_i
          = sum_i u_i^2/lambda_i
            - beta sum_i u_i z_i
            + beta^2/4 sum_i lambda_i z_i^2.

    Therefore:
        d0 = sum_i u_i^2/lambda_i,
        d1 = -sum_i u_i z_i,
        d2 = (1/4) sum_i lambda_i z_i^2.
    """

    lam = np.maximum(np.asarray(lambdas, dtype=float), EPS)
    u = np.asarray(u_coords, dtype=float)
    z = np.asarray(z_coords, dtype=float)
    if not (lam.shape == u.shape == z.shape):
        raise ValueError("lambdas, u_coords, z_coords must have the same shape")
    return DefectCoefficients(
        # d0: deterministic drift mismatch penalized by inverse SGD diffusion.
        d0=float(np.sum((u * u) / lam)),
        # d1: cross term between SGD drift mismatch and Gibbs bounded-loss score.
        d1=float(-np.sum(u * z)),
        # d2: Gibbs score contribution measured in the SGD diffusion metric.
        d2=float(0.25 * np.sum(lam * z * z)),
    )
```

Implementation: finite_time_kl_bound

```
def finite_time_kl_bound(
    kl_initial: float,
    defect_value: float,
    alpha: float,
    time: float,
    epsilon: float = 1.0,
) -> float:
    """Frozen-coefficient closed form of the finite-time KL integral.

    Math:
        KL_t <= exp(-alpha t) KL_0
                + (1/(2 epsilon)) D int_0^t exp[-alpha(t-s)] ds.

    This is still an integration, but it is the analytic value of that
    integration after freezing D_s = D and alpha_s = alpha on [0,t].  It is
    appropriate for endpoint/asymptotic local diagnostics.  The checkpoint-path
    theorem-level curve should use finite_time_kl_path_bound below.
    """

    # Initial mismatch decays exponentially under retained contraction.
    decay = float(np.exp(-float(alpha) * float(time)))
    # Defect source enters through the exponentially weighted time integral.
    path = float(defect_value) * exponential_integral_multiplier(alpha, time) / (2.0 * float(epsilon))
    return float(decay * float(kl_initial) + path)
```

Implementation: gaussian_kl

```
def gaussian_kl(k, sigma_q, sigma_p):
    """KL[N(0,sigma_q^2 I_k) || N(0,sigma_p^2 I_k)].

    Taking the Q expectation of log(q/p) gives
    k/2 * [sigma_q^2/sigma_p^2 - 1 + log(sigma_p^2/sigma_q^2)].
    Both distributions live on the same frozen retained affine subspace.
    """
```

```
    ratio = (sigma_q / sigma_p) ** 2
    return 0.5 * k * (ratio - 1 - np.log(ratio))
```

Implementation: certificate_from_draws

```
def certificate_from_draws(risks, k, sigma_q, sigma_p, n,
                           delta_pb=0.008, delta_mc=0.002):
    """Separate posterior Monte Carlo uncertainty from the PAC-Bayes term.

    risks[s] is the empirical loss, averaged over all n certificate examples,
    for one independent posterior network. Each entry is in [0,1].
    The empirical-Bernstein upper correction uses the variance ACROSS draws.
    A fixed-prior/fixed-Q Hoeffding PAC-Bayes calculation supplies the PB term;
    the build-only Q and bound parameter are frozen before certificate access.
    Five rows allocate 5*(delta_pb + delta_mc) = 0.05 failure probability.
    """
    risks = np.asarray(risks, dtype=np.float64)
    assert risks.ndim == 1 and len(risks) > 1 and n > 0
    assert np.isfinite(risks).all() and np.all((0 <= risks) & (risks <= 1))
    S = len(risks)
    mean, variance = risks.mean(), risks.var(ddof=1)
    log_mc = np.log(2 / delta_mc)
    mc = np.sqrt(2 * variance * log_mc / S) + 7 * log_mc / (3 * (S - 1))
    kl = gaussian_kl(k, sigma_q, sigma_p)
    pb = np.sqrt((kl + np.log(1 / delta_pb)) / (2 * n))
    return dict(certificate_mean=mean, mc_correction=mc, kl=kl,
                pb_complexity=pb, population_bound=mean + mc + pb)

# Recompute from the 20,480 saved certificate-network evaluations.
evidence = PROJECT_ROOT / 'paper_data/pac_bayes'
verified = subprocess.run([sys.executable, str(evidence / 'verify_compact_evidence.py')],
                          check=True, text=True, capture_output=True)
certificate_table = reconstruct_table(evidence)
certificate_table.to_csv(OUTPUT_ROOT / 'table1_recomputed.csv', index=False)
display(certificate_table[['dataset', 'model', 'k', 'kl', 'population_bound', 'test_risk']]
        .rename(columns={'kl':'Exact KL', 'population_bound':'Bound (lower)', 'test_risk':'Test risk (lower)'})
        .style.format({'Exact KL':'{:.5f}', 'Bound (lower)':'{:.5f}', 'Test risk (lower)':'{:.5f}'}))
print('All five rows match the frozen table; compact evidence hashes and arithmetic pass.')
```

dataset	model	k	Exact KL	Bound (lower)	Test risk (lower)
MNIST	MLP	1	0.31815	0.04258	0.02198
MNIST	LeNet	1	0.31815	0.02983	0.00898
CIFAR-10	ResNet-32	16	5.09035	0.07657	0.05840
CIFAR-10	WideResNet-16-4	11	3.49962	0.06349	0.04448
CIFAR-10	DenseNet-small	8	2.54518	0.07669	0.06100

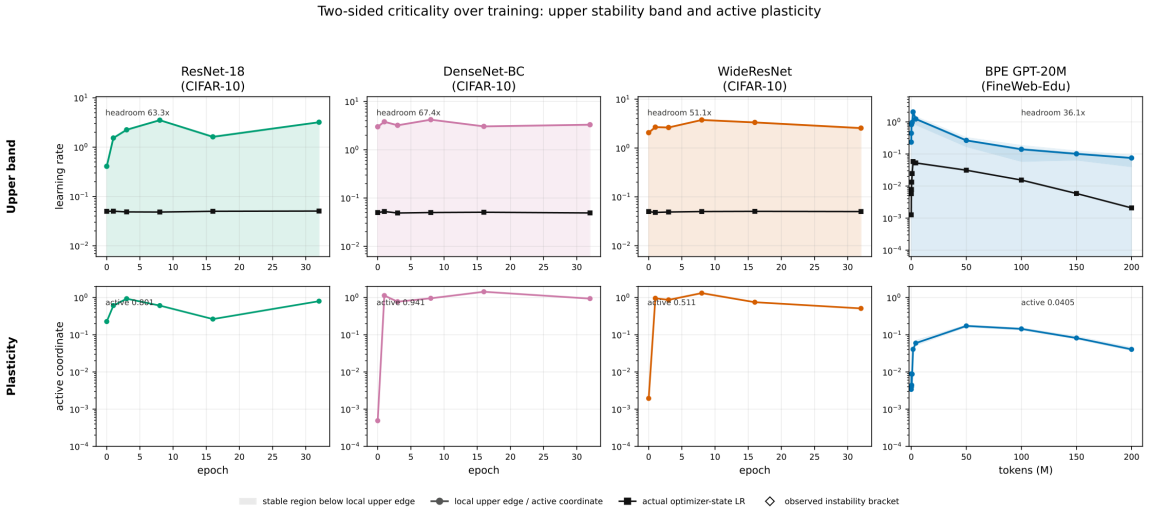
```
All five rows match the frozen table; compact evidence hashes and arithmetic pass.
```

Interpretation. The five bounds are nonvacuous for a loss bounded by one. They concern the specified local randomized predictors and the independent certificate protocol. The Gaussian KL grows with retained dimension k; this table does not claim an unrestricted full-weight posterior certificate. The original TensorFlow producer and all frozen settings are included under `paper_data/pac_bayes/experiment/`; the compact reader above is the same statistical calculation, with no optimizer or training dependency.

5. Stability and active response over training

The following curves are redrawn from the recorded checkpoint-coordinate CSVs. The upper row compares the optimizer-state scalar step with the sampled-curvature edge. The lower row

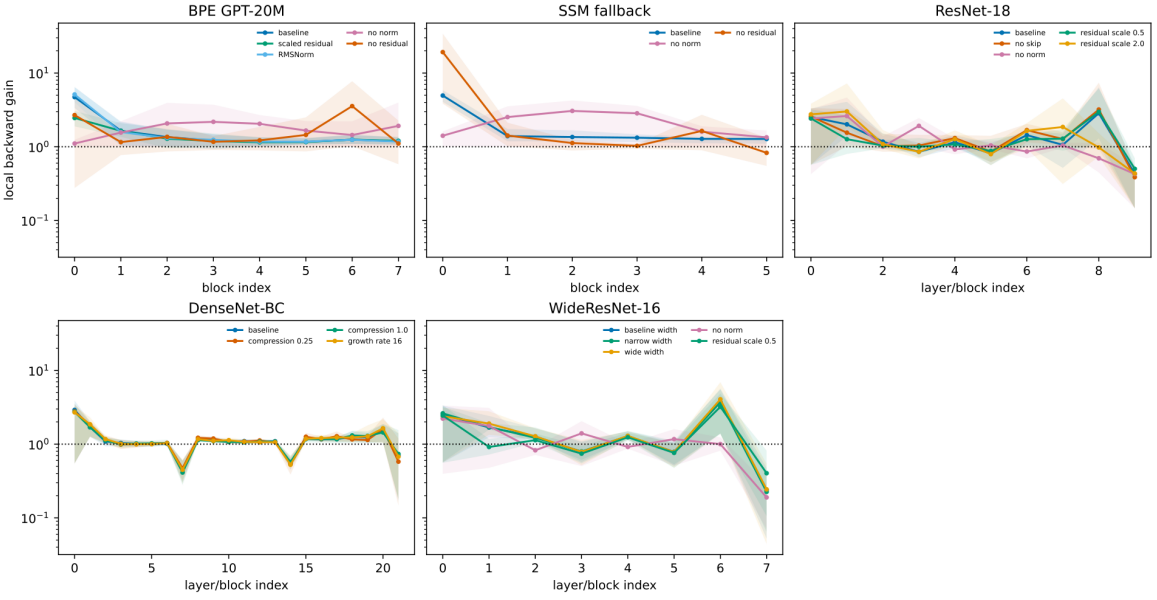
examines active-space response. These are complementary: remaining below an upper edge does not ensure substantial useful response. The two sides describe local quadratic diagnostics, not a global convergence guarantee.



Download figure PDF

Localizing the affected interface

Backward-gain curves are regenerated from the saved boundary-gradient summaries. Their role is localization: a global scalar can indicate weak response, whereas the layerwise curve shows the interfaces where the observed loss sensitivity changes. The dashed neutral-gain reference is one; gains are directional and scale-dependent, so they should be compared within the stated boundary convention.



Download figure PDF

6. Architecture-visible precision

This section presents the lifted precision figures used to introduce the paper's local-interaction object. These figures are the code supplement's bridge from the computation graph to the Schur-complement curvature identity.

1. Local factors and lifted precision support

The lifted state is $z = (\theta, u)$, where θ are parameters and u are intermediate states already present in the forward computation. Each local factor f touches a small scope S_f of these variables. At a checkpoint, a residual-form factor has first variation

$$r_f(z_0 + \Delta z_{S_f}) = r_f(z_0) + J_f(z_0)\Delta z_{S_f} + o(\|\Delta z_{S_f}\|).$$

With local weight $W_f \succeq 0$, this factor contributes the embedded information block

$$\mathrm{Embed}_{S_f}\left(J_f(z_0)^\top W_f J_f(z_0)\right)$$

to the lifted precision matrix. Thus the support pattern of the lifted precision is determined by which variable groups co-occur in local computational factors.

2. Activation elimination

After assembling the lifted precision,

$$P_{\theta,u} = \Lambda_\theta \oplus \tau I_u + \sum_f \mathrm{Embed}_{S_f}\left(J_f^\top W_f J_f\right),$$

partition it into parameter and state blocks. Eliminating intermediate states gives

$$F_{\kappa,\tau} = P_{\theta\theta} - P_{\theta u} P_{uu}^{\dagger} P_{u\theta}.$$

At finite precision this is the softened curvature; Theorem 1 supplies the additional hard-constraint limit. The figures shown in this section visualize the sparse object before elimination. They make the architecture-visible locality clear: MLP blocks produce short chain-like supports; residual CNN blocks add skip and normalization interfaces; transformer blocks add residual-stream, attention, and feed-forward couplings.

3. Implementation map

The displayed source function `generate_block_precision_figures` is the compact entry point. It calls the renderer, writes the MLP, residual-CNN, and transformer block figures, and copies them under stable paper-facing names. The exact support builders are mentioned with file/line locations rather than source-dumped because the transformer support builder is long:

- `exact_mlp_block_elementwise_support`
- `exact_resnet_block_elementwise_support`
- `exact_transformer_block_elementwise_support`

The whole-network panels are generated or copied in the following cell. They use the same lifted-support principle, but at a larger architecture scale.

The default previews retain the paper PDFs. Their TensorFlow builders are included; `RUN_STAGE5_RENDERERS=1` explicitly re-extracts and redraws the architecture supports. This rendering switch does not train the models.

Lifted precision: implementation entry point

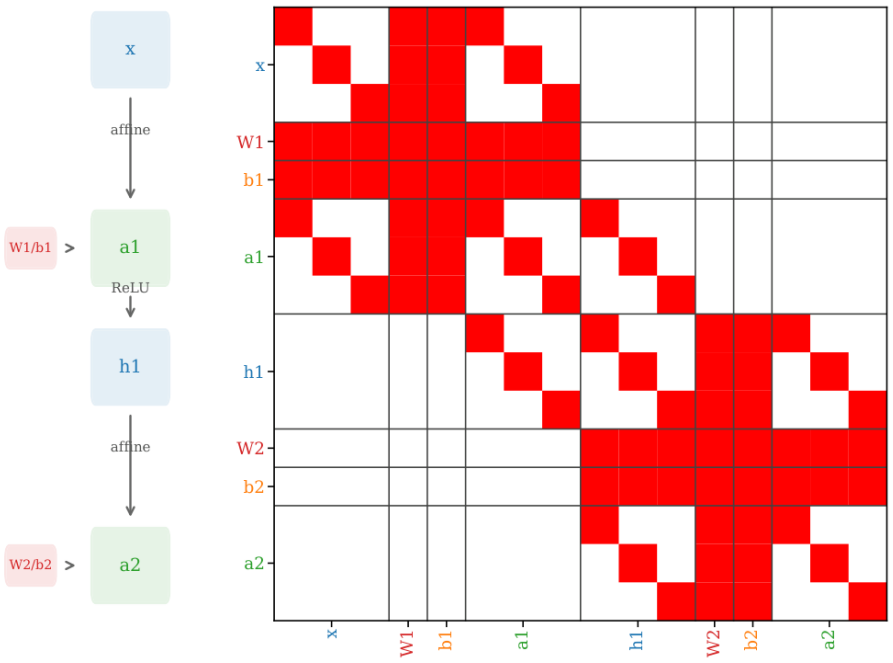
The short entry point assembles the block figures. The detailed elementwise support builders are linked below it.

Implementation: generate_block_precision_figures

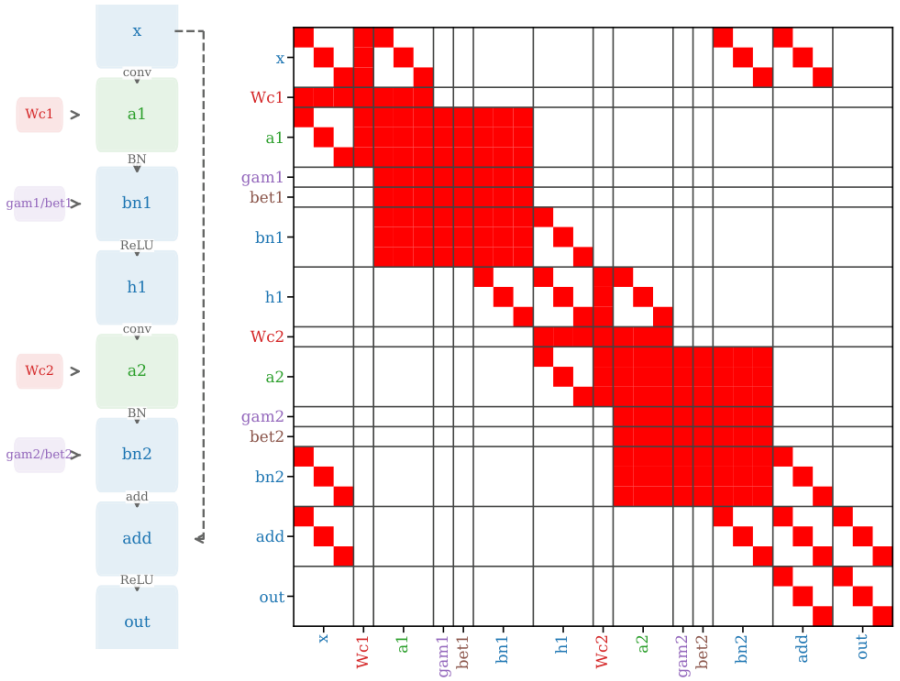
```
def generate_block_precision_figures(out_dir: Path = OUT_DIR) -> list[Path]:
    """Generate paper-facing MLP/ResNet/Transformer block precision figures."""

    out_dir = Path(out_dir)
    out_dir.mkdir(parents=True, exist_ok=True)
    renderer = _import_renderer()
    old_argv = sys.argv[:]
    try:
        sys.argv = [
            "local_precision_pattern_renderer.py",
            "--out_dir",
            str(out_dir),
            "--batch_size_display",
            "3",
            "--max_points",
            "1000000",
        ]
        renderer.main()
    finally:
        sys.argv = old_argv

    written: list[Path] = []
    for paper_name, compact_name in PAPER_BLOCK_FILES.items():
        paper_path = out_dir / paper_name
        compact_path = out_dir / compact_name
        if not paper_path.exists():
            raise FileNotFoundError(f"renderer did not write {paper_path}")
        shutil.copy2(paper_path, compact_path)
        written.extend([paper_path, compact_path])

    overview = out_dir / "lifted_precision_overview.pdf"
    compact_triptych = out_dir / "compact_lifted_precision_triptych.pdf"
    if overview.exists():
        shutil.copy2(overview, compact_triptych)
        written.append(compact_triptych)
    return written
```

Exact support builders available for inspection

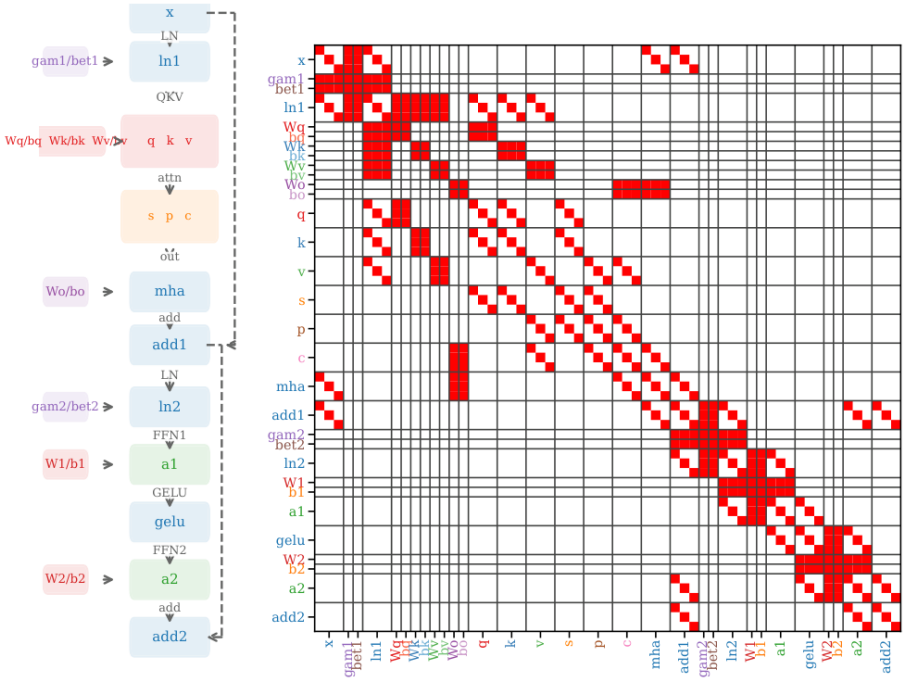
- `local_precision_pattern_renderer.exact_mlp_block_elementwise_support`: `code/local_precision_pattern_renderer.py:2207`
- `local_precision_pattern_renderer.exact_resnet_block_elementwise_support`: `code/local_precision_pattern_renderer.py:2253`
- `local_precision_pattern_renderer.exact_transformer_block_elementwise_support`: `code/local_precision_pattern_renderer.py:2333`



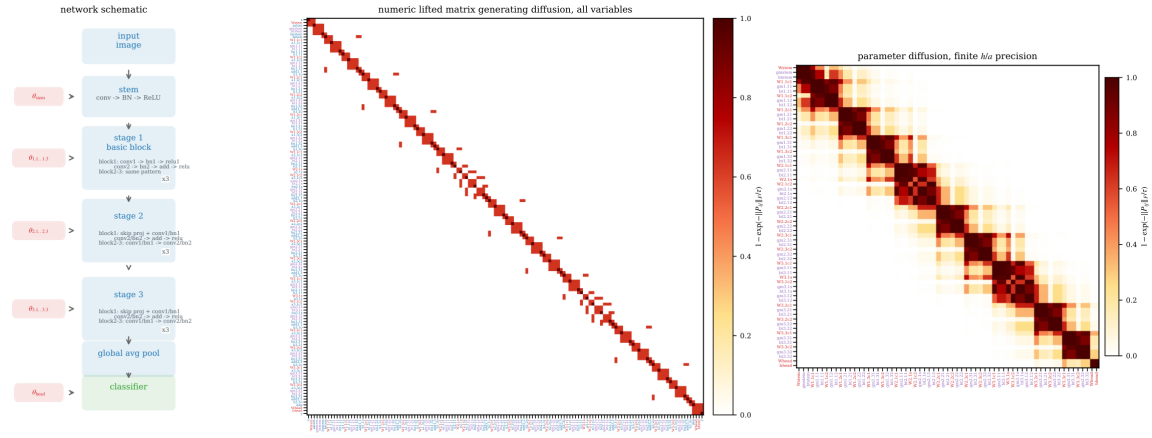
Download figure PDF



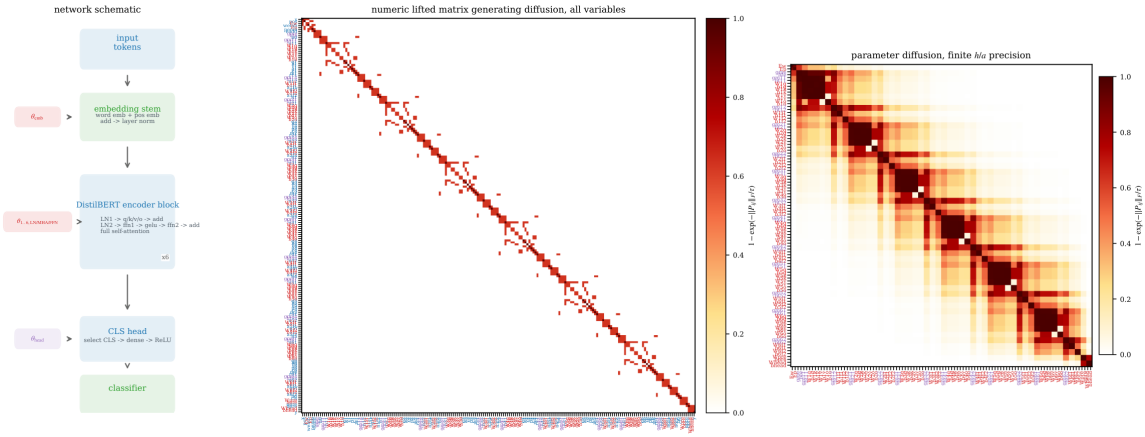
Download figure PDF



Download figure PDF



Download figure PDF



Download figure PDF

7. Stochastic geometry and predictive relevance

This section explains the probability geometry behind the resolved KL/PAC-Bayes construction. A Gibbs comparison over the full parameter space uses an inverse covariance metric. Directions with very small minibatch-gradient covariance eigenvalues can therefore dominate the full-parameter Gibbs defect, even when finite movement along those directions changes held-out predictions very little. The theorem studies a declared comparison space. The split-certified table separately freezes its local prior and posterior on a common retained affine space; the geometry plots alone do not establish that certificate.

1. Direction definitions

Let $f_\theta(X)$ be the vector of logits on a measured batch X, and let $J_f(\theta; X)$ be its parameter Jacobian.

A **symmetry direction** is a parameter displacement v for which $J_f(\theta; X)v \approx 0$ on the measured batch. It changes coordinates while leaving the represented function nearly unchanged on the measured data.

A **structural gauge** is a known coordinate redundancy, such as ReLU/BatchNorm scaling, that produces or approximates such a symmetry.

A **loss-space gauge** is an output redundancy. For softmax cross-entropy, adding the same scalar to every logit leaves probabilities unchanged.

A **loss tangent** is a direction v whose output tangent $q_v = J_f(\theta; X)v$ has small bounded-loss curvature, for example

$$\|H_\ell^{1/2} q_v\| \ll \|q_v\|,$$

where H_ℓ is the positive-semidefinite local output curvature on a smooth loss region. Such a direction can move logits while barely changing clipped loss or classification error.

An **augmentation tangent** compares $J_f(\theta; X)v$ with a data-transformation response $f_\theta(TX) - f_\theta(X)$ for a transformation such as crop, color, MixUp, or CutMix. This probes whether a parameter direction resembles a data-space transformation in output space.

The weak-direction panels focus on low-covariance directions that are not already identified by explicit symmetry or gauge probes. The XAI plots then inspect whether those remaining

directions are better explained as output/loss tangents or as simple input-transform tangents.

2. Direction-level normalization

For each condition and direction i, the direction table provides a minibatch-gradient covariance eigenvalue λ_i and an inverse-metric Gibbs-defect contribution a_i, such as a directionwise $d_{0,i}$. Normalize these into

$$s_i = \frac{a_i}{\sum_j a_j}, \qquad m_i = \frac{\lambda_i}{\sum_j \lambda_j}.$$

When a GGN/train-Fisher curvature κ_i is available, the ordinary-geometry mass is

$$f_i = \frac{\max(\kappa_i, 0)}{\sum_j \max(\kappa_j, 0)}.$$

Here λ_i is the minibatch-gradient covariance eigenvalue in direction v_i. If $r_i = \langle b - b_{\mathrm{rev}}, v_i \rangle$ is the drift/reference mismatch coordinate, then

$$d_{0,i} = \frac{r_i^2}{\lambda_i}$$

is the directionwise inverse-metric contribution to the Gibbs defect at $\beta = 0$. When available, $\kappa_i = v_i^\top \hat{F}_{\mathrm{train}} v_i$ is the train Fisher/GGN Rayleigh value used as ordinary curvature mass.

For a low-covariance tail B, the aggregate comparison is

$$S_B = \sum_{i \in B} s_i, \qquad M_B = \sum_{i \in B} m_i, \qquad F_B = \sum_{i \in B} f_i.$$

Large S_B with small M_B and F_B means that the ambient full-parameter Gibbs defect is dominated by directions that carry little ordinary covariance or train-Fisher mass.

3. Prediction and augmentation diagnostics

For selected direction groups, the code perturbs the checkpoint at a fixed radius and recomputes held-out clipped-loss and error changes. This asks whether a high inverse-metric defect group produces visible finite predictive movement.

For augmentation probes, the code compares a direction-induced output tangent $h_i = J_f(\theta; X) v_i$ with transformation tangents $t_a = f_\theta(T_a X) - f_\theta(X)$ using

$$c_i(a) = \frac{|\langle h_i, t_a \rangle|}{\|h_i\| \, \|t_a\|}.$$

The spectrum overlays and retained train/validation alignment plot use these direction-level quantities to explain how augmentation changes the local probability geometry.

4. Reading the measured panels

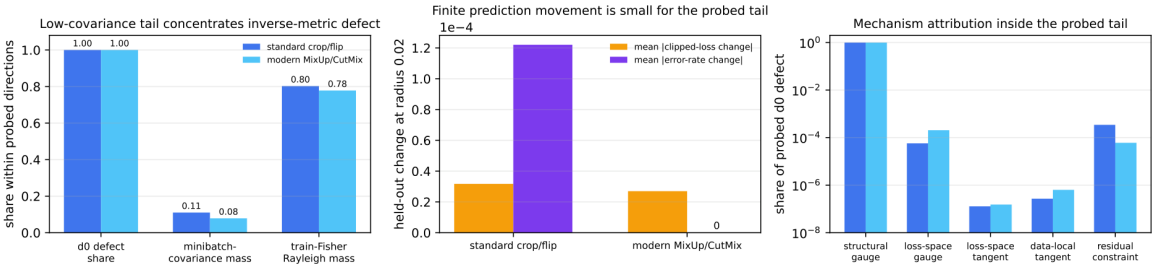
The first panel compares inverse-metric defect, covariance mass, curvature, and finite prediction changes for the same low-covariance directions. In these ResNet records the tail carries about 0.11/0.08 of covariance mass but 0.80/0.78 of probed curvature mass. Thus small diffusion does **not** imply small curvature. The finite-radius measurements provide the separate evidence for small predictive movement; a zero error-rate change is shown as zero.

The next panels compare retained-direction train/validation alignment and clean train/validation/test risks for crop/flip and MixUp/CutMix. These are checkpoint diagnostics, not new training runs or additional population certificates.

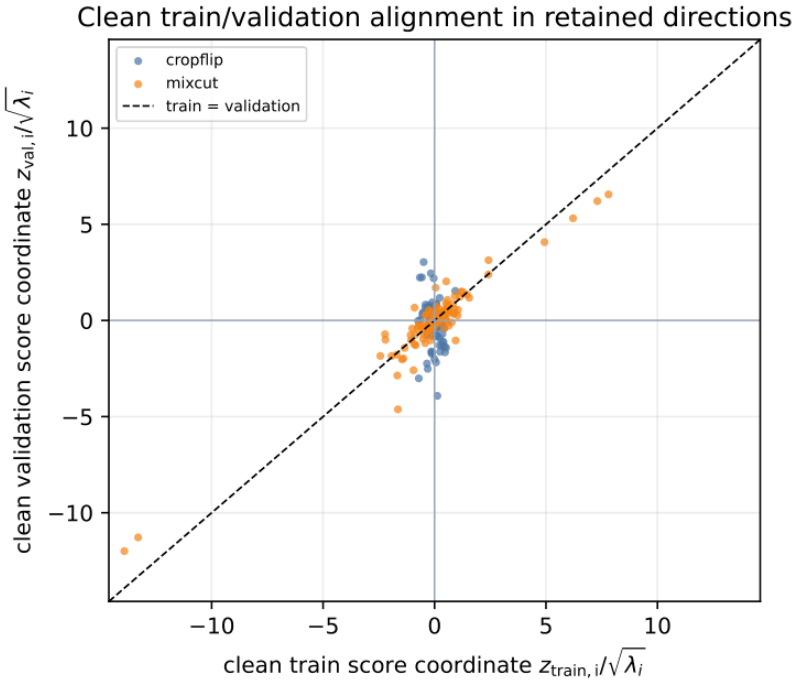
All three panels use the archived ResNet tables listed in `paper_data/measured_directions/SOURCES.json`. Constructed examples are kept separately under `paper_data/illustrations/` and are excluded from this empirical path.

Implementation: direction_defect_contributions

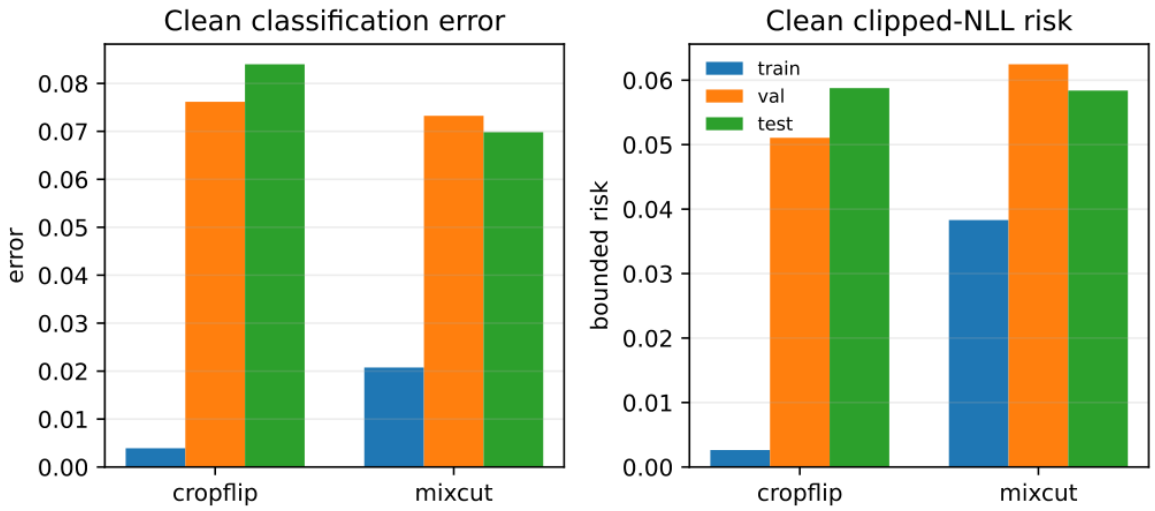
```python
def direction_defect_contributions(
    lambdas: np.ndarray,
    u_coords: np.ndarray,
    z_coords: np.ndarray,
) -> dict[str, np.ndarray]:
    """Directionwise d0,d1,d2 terms used before prefix selection.

    For each covariance eigen-direction v_i:
        lambda_i = retained SGD diffusion eigenvalue,
        u_i = <u, v_i>,
        z_i = <z, v_i>.

    The one-direction contribution is
        D_i(beta)
          = (u_i - beta lambda_i z_i / 2)^2 / lambda_i
          = d0_i + beta d1_i + beta^2 d2_i,

    where
        d0_i = u_i^2 / lambda_i,
        d1_i = -u_i z_i,
        d2_i = lambda_i z_i^2 / 4.
    """

    lam = np.maximum(np.asarray(lambdas, dtype=float), EPS)
    u = np.asarray(u_coords, dtype=float)
    z = np.asarray(z_coords, dtype=float)
    if not (lam.shape == u.shape == z.shape):
        raise ValueError("lambdas, u_coords, z_coords must have the same shape")
    return {
        "d0_contrib": (u * u) / lam,
        "d1_contrib": -(u * z),
        "d2_contrib": 0.25 * lam * z * z,
    }
```



Download figure PDF



Download figure PDF



Download figure PDF

8. Code reading and reproduction scope

Reader task	Entry point	Inputs	Framework
Follow the equations	`code/core_math_identities.py`	Small arrays	NumPy
Reconstruct Table 1	`code/pac_bayes_certificate.py`	Frozen per-network risks	NumPy/Pandas
Verify the evidence	`paper_data/pac_bayes/verify_compact_evidence.py`	Hashed records	Python standard library
Redraw criticality curves	`code/criticality_coordinates.py`	Recorded CSVs	Matplotlib; PyTorch for checkpoint replay
Read architecture supports	`code/local_precision_pattern_renderer.py`	Module scopes	TensorFlow
Inspect direction geometry	`code/resolved_direction_figures.py`	Measured direction tables	NumPy/Matplotlib
Inspect the original certificate run	`paper_data/pac_bayes/experiment/run_safe_table1.py`	Original data, checkpoints and basis	TensorFlow

Read, recompute, or replay

Open `README.executed.html` to read the tutorial, or `README.executed.ipynb` to inspect its executed cells. The clean source is `README.ipynb`. Run `python verify_core_identities.py` for the algebraic checks and `python reproduce_main_artifacts.py` for the notebook's default calculations. The latter writes `README.executed.ipynb` and results under `output/`. It reconstructs statistics from saved measurements; it does not retrain models.

Optional checkpoint replay needs the original weights, optimizer state where used, and matching data. `RECOMPUTE_CHECKPOINT_DIAGNOSTICS=1` enables that path; `RUN_STAGE5_RENDERERS=1` enables the TensorFlow precision builders. Missing checkpoints stop execution rather than silently triggering training.

For original training, distinguish `code/checkpoint_training_reference.py` (whose default examples include synthetic execution checks) from the exact revised-table producer `paper_data/pac_bayes/experiment/run_safe_table1.py`. Its five checkpoint manifests record TensorFlow 2.20.0; submitted settings, source dependencies, frozen protocol and training logs accompany it. It retains its original directory assumptions: its sibling imports must be restored from `reference_sources/` before launching it. `RUN_STAGE1_LONG_JOBS=1` is not a one-command reproduction of all published training.

The evidence verifier checks saved hashes, arithmetic and recorded access order; it does not independently re-evaluate the omitted networks. The complete checkpoint/basis archive is optional, with omitted filenames and hashes recorded in `paper_data/pac_bayes/OMITTED_HEAVY_ARTIFACTS.csv`. The default tutorial needs neither that archive nor a cloud account.