LLM-Guided PCA-Latent Corrosion Mask Forecasting

Technical Report

Alexander Hermann

February 23, 2026

Abstract

This report documents the mathematical foundations and software architecture of a corrosion forecasting pipeline designed to predict the temporal evolution of binary corrosion masks for Mg–4Ag biodegradable alloy wires. The masks are derived from in-situ synchrotron nano-CT imaging and captured at ten discrete degradation time-steps. The pipeline combines *Principal Component Analysis* (PCA) via randomised Singular Value Decomposition (SVD), a *Signed Distance Field* (SDF) representation, k-Nearest Neighbour (kNN) delta prediction in latent space, and a *Large Language Model* (LLM) that provides a stochastic residual correction to the kNN prior. Deterministic stabilisers ensure physical plausibility (e.g. monotonic material loss), while Monte Carlo rollouts yield pixel-wise uncertainty maps. The entire system is implemented as a modular Python package (`corrosion_forecast`) and evaluated with spatial overlap, boundary accuracy, and probabilistic calibration metrics.

Contents

Part I
Mathematical Theory

1 Principal Component Analysis via Singular Value Decomposition

1.1 Data Matrix Construction

Let T denote the total number of temporal snapshots and S the number of training slice indices. Each corrosion mask image of resolution $H \times W$ is first converted to a field representation (Section 2) and then flattened into a row vector of length $N = H \cdot W$. Stacking all $M = S \times T$ training fields yields the data matrix

$$\mathbf{X} \;=\; \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_M^\top \end{pmatrix} \;\in\; \mathbb{R}^{M \times N}. \tag{1}$$

1.2 Mean Subtraction

The global mean field is

$$\boldsymbol{\mu} \;=\; \frac{1}{M} \sum_{i=1}^{M} \mathbf{x}_i \;\in\; \mathbb{R}^N, \tag{2}$$

and the mean-centred data matrix is

$$\mathbf{X}_c \;=\; \mathbf{X} - \mathbf{1}_M \, \boldsymbol{\mu}^\top \;\in\; \mathbb{R}^{M \times N}, \tag{3}$$

where $\mathbf{1}_M$ is the M-vector of ones.

1.3 Singular Value Decomposition

The (economy) SVD of the centred matrix is

$$\mathbf{X}_c \;=\; \mathbf{U}\,\mathbf{S}\,\mathbf{V}^\top, \tag{4}$$

where $\mathbf{U} \in \mathbb{R}^{M \times M}$ contains left singular vectors, $\mathbf{S} = \operatorname{diag}(\sigma_1, \ldots, \sigma_{\min(M,N)})$ are singular values in descending order, and $\mathbf{V} \in \mathbb{R}^{N \times N}$ contains right singular vectors. The columns of $\mathbf{V}$ (equivalently the rows of $\mathbf{V}^\top$) are the *principal component directions* $\mathbf{v}_1, \ldots, \mathbf{v}_N$.

1.4 Truncated Reconstruction

Retaining only $K \ll \min(M, N)$ components, an arbitrary field $\mathbf{x}$ is reconstructed as

$$\hat{\mathbf{x}} \;=\; \boldsymbol{\mu} + \sum_{k=1}^{K} z_k \, \mathbf{v}_k, \tag{5}$$

where the *latent coefficients* are

$$z_k \;=\; (\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{v}_k, \qquad k = 1, \ldots, K. \tag{6}$$

The latent vector $\mathbf{z} = (z_1, \ldots, z_K)^\top \in \mathbb{R}^K$ provides a low-dimensional representation of the corrosion field.

1.5 Explained Variance Ratio

Each singular value σ_k is related to the eigenvalue of the covariance matrix via $\lambda_k = \sigma_k^2/(M-1)$. The fraction of variance explained by the first K components is

$$\mathrm{EVR}(K) \;=\; \frac{\sum_{k=1}^{K} \lambda_k}{\sum_{j=1}^{\min(M,N)} \lambda_j} \;=\; \frac{\sum_{k=1}^{K} \sigma_k^2}{\sum_{j=1}^{\min(M,N)} \sigma_j^2}. \tag{7}$$

1.6 Randomised SVD

For large M and N, the full SVD is computationally prohibitive. Following [1], we use the *randomised SVD* algorithm, which approximates the top-K singular triplets in $\mathcal{O}(MN \log K)$ time rather than $\mathcal{O}(MN \min(M,N))$. The algorithm proceeds by:

1. Drawing a random Gaussian matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times (K+p)}$, where p is a small over-sampling parameter.
2. Computing the sample matrix $\mathbf{Y} = \mathbf{X}_c\,\mathbf{\Omega}$.
3. Obtaining an orthonormal basis $\mathbf{Q}$ for the column space of $\mathbf{Y}$ via QR decomposition.
4. Forming the small matrix $\mathbf{B} = \mathbf{Q}^\top \mathbf{X}_c$ and computing its SVD.

In our implementation, we use `sklearn.utils.extmath.randomized_svd` when available, falling back to `numpy.linalg.svd` otherwise.

1.7 Latent Normalisation

To ensure numerical stability when feeding latent vectors to downstream predictors, we compute per-component statistics

$$\bar{z}_k = \frac{1}{M}\sum_{i=1}^{M} z_k^{(i)}, \qquad s_k = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\big(z_k^{(i)} - \bar{z}_k\big)^2 + \epsilon}, \tag{8}$$

with $\epsilon = 10^{-6}$, and work with the normalised latent $\tilde{z}_k = (z_k - \bar{z}_k)/s_k$ throughout the forecasting pipeline.

2 Signed Distance Field (SDF) Representation

2.1 Definition

Definition 2.1 (Signed Distance Field)**.** Given a binary material mask $\mathcal{M} \subseteq \mathbb{R}^2$ (the set of pixels labelled as material), the Signed Distance Field is defined at every pixel location $\mathbf{x}$ as

$$\mathrm{SDF}(\mathbf{x}) \;=\; d_{\mathrm{outside}}(\mathbf{x}) \;-\; d_{\mathrm{inside}}(\mathbf{x}), \tag{9}$$

where

$$d_{\mathrm{inside}}(\mathbf{x}) = \inf_{\in \partial\mathcal{M}} \|\mathbf{x} - \|_2 \quad \text{for } \mathbf{x} \in \mathcal{M}, \tag{10}$$

$$d_{\mathrm{outside}}(\mathbf{x}) = \inf_{\in \partial\mathcal{M}} \|\mathbf{x} - \|_2 \quad \text{for } \mathbf{x} \notin \mathcal{M}, \tag{11}$$

and $\partial\mathcal{M}$ denotes the material boundary.

By this convention:

- $\mathrm{SDF}(\mathbf{x}) < 0$ for pixels **inside** the material.
- $\mathrm{SDF}(\mathbf{x}) > 0$ for pixels **outside** the material.
- $\mathrm{SDF}(\mathbf{x}) = 0$ at the material boundary.

In practice, both distance transforms are computed using the Euclidean Distance Transform (`scipy.ndimage.distance_transform_edt`).

2.2 Zero Level-Set Recovery

The material mask can always be recovered from the SDF by thresholding:

$$\mathcal{M} \;=\; \big\{\mathbf{x} : \mathrm{SDF}(\mathbf{x}) \le 0\big\}. \tag{12}$$

2.3 Advantages over Raw Binary Masks

Using SDF representations instead of raw $\{0, 255\}$ binary masks for PCA confers several advantages:

1. **Smoothness:** The SDF is a continuous, piecewise-smooth function. Small perturbations of the boundary produce small perturbations of the SDF in the L^2 norm, whereas binary masks exhibit discontinuous jumps.
2. **Boundary sensitivity:** The SDF gradient is largest near the boundary (magnitude = 1 everywhere for exact distance functions), ensuring that PCA modes concentrate representational power on the shape of the boundary rather than on interior fill.
3. **Linear interpolation:** Averaging two SDF fields in PCA latent space yields a geometrically meaningful intermediate shape, whereas averaging two binary masks produces ambiguous grey values.
4. **Reconstruction quality:** Truncated PCA reconstruction of a smooth SDF produces a smooth field whose zero level-set gives a clean boundary; reconstructing a binary mask introduces ringing artefacts.

3 k-Nearest Neighbour Delta Prediction

3.1 Feature Vector Construction

The kNN predictor operates in the normalised PCA latent space. At time t, the feature vector for a query is composed of:

$$\mathbf{f} \;=\; \big[\,\tilde{\mathbf{z}}_t,\;\; \dot{\tilde{\mathbf{z}}}_t,\;\; t_{\mathrm{norm}},\;\; \Delta t_{\mathrm{norm}}\,\big] \;\in\; \mathbb{R}^{2K+2}, \tag{13}$$

where:

- $\tilde{\mathbf{z}}_t \in \mathbb{R}^K$ is the normalised latent state at time t.
- $\dot{\tilde{\mathbf{z}}}_t = \tilde{\mathbf{z}}_t - \tilde{\mathbf{z}}_{t-1}$ is the latent velocity (zero for $t = 0$).
- $t_{\mathrm{norm}} = t_h / t_{\max}$ and $\Delta t_{\mathrm{norm}} = \Delta t_h / t_{\max}$ are the normalised current time and step size (in hours), respectively.

3.2 Library Construction

For each training slice and each pair of consecutive time-steps $(t,\, t+1)$, we store the pair $(\mathbf{f}_t,\, \boldsymbol{\delta}_t)$ where $\boldsymbol{\delta}_t = \tilde{\mathbf{z}}_{t+1} - \tilde{\mathbf{z}}_t$ is the ground-truth latent delta. The full library is

$$\mathcal{L} = \left\{ (\mathbf{f}^{(i)},\, \boldsymbol{\delta}^{(i)}) \right\}_{i=1}^{|\mathcal{L}|}. \tag{14}$$

3.3 Inverse-Distance Weighted Prediction

Given a query feature $\mathbf{f}_q$, we find the k nearest neighbours $\mathcal{N}_k(\mathbf{f}_q)$ in $\mathcal{L}$ by Euclidean distance. For each neighbour $i \in \mathcal{N}_k$, the distance is

$$d_i = \|\mathbf{f}_q - \mathbf{f}^{(i)}\|_2. \tag{15}$$

The inverse-distance weights are

$$w_i = \frac{1/d_i}{\displaystyle\sum_{j \in \mathcal{N}_k} 1/d_j}, \qquad i \in \mathcal{N}_k, \tag{16}$$

with a small additive constant $\epsilon = 10^{-6}$ for numerical stability.

The **weighted mean** (the kNN prior) is

$$\boldsymbol{\delta}_{\text{prior}} = \boldsymbol{\mu}_{\text{kNN}} = \sum_{i \in \mathcal{N}_k} w_i\, \boldsymbol{\delta}^{(i)}, \tag{17}$$

and the **weighted variance** (for uncertainty estimation) is

$$\sigma^2_{\text{kNN}} = \sum_{i \in \mathcal{N}_k} w_i \big(\boldsymbol{\delta}^{(i)} - \boldsymbol{\mu}_{\text{kNN}}\big)^2. \tag{18}$$

4 LLM-in-the-Loop Forecasting

4.1 Hybrid Architecture

The pipeline employs a *hybrid* forecasting strategy: the kNN module provides a physics-informed, data-driven baseline prediction (the *prior*), and a Large Language Model (LLM) adds a stochastic *residual correction* that captures non-linear dependencies beyond nearest-neighbour interpolation.

The final predicted delta is

$$\boldsymbol{\delta} \;=\; \boldsymbol{\delta}_{\text{prior}} \;+\; \alpha\, \boldsymbol{\delta}_{\text{LLM}}, \tag{19}$$

where $\alpha \in [0, 1]$ is the `RESIDUAL_SCALE` parameter (default $\alpha = 0.7$) and $\boldsymbol{\delta}_{\text{LLM}}$ is the residual vector returned by the LLM.

4.2 Prompt Structure

The LLM receives a single **JSON payload** as the user message, containing:

```
{
  "D":              <int: dimension of latent space>,
  "dt_hours":       <float: time-step in hours>,
  "last_latent":    [z_1, z_2, ..., z_K],
  "velocity":       [v_1, v_2, ..., v_K],
  "acceleration":   [a_1, a_2, ..., a_K],
  "delta_prior":    [d_1, d_2, ..., d_K],
  "prior_std":      [s_1, s_2, ..., s_K],
  "residual_norm_cap":  <float>,
  "residual_comp_cap":  <float>,
  "rollout_nonce":      <float: for stochastic diversity>,
  "metadata_context":   { ... }
}
```

The system prompt instructs the model to return **only** valid JSON with a single key `predicted_delta` containing exactly K floats. Explicitly:

```
Return ONLY JSON. No prose. No markdown.
Output ONLY JSON with exactly one key predicted_delta.
predicted_delta must be a list of exactly D=<K> floats.
Interpret predicted_delta as a RESIDUAL to add to delta_prior.
Keep it small: L2(residual) <= residual_norm_cap
  and abs(component) <= residual_comp_cap.
```

4.3 Kinematic Context

To help the LLM reason about the dynamics, we provide:

- The **last latent state** $\tilde{\mathbf{z}}_t$.
- The **velocity** $\dot{\tilde{\mathbf{z}}}_t = \tilde{\mathbf{z}}_t - \tilde{\mathbf{z}}_{t-1}$.
- The **acceleration** $\ddot{\tilde{\mathbf{z}}}_t = \dot{\tilde{\mathbf{z}}}_t - \dot{\tilde{\mathbf{z}}}_{t-1}$ (zero if fewer than three observations are available).

4.4 Rationale for the Hybrid Approach

1. **kNN provides structure:** The inverse-distance-weighted neighbourhood average is fast, purely data-driven, and guaranteed to stay within the convex hull of training deltas. It encodes the *typical* magnitude and direction of latent evolution.
2. **LLM adds flexibility:** By operating as a residual predictor, the LLM need only learn the *correction* to the prior, which is typically much smaller than the full delta. This reduces the effective prediction complexity and makes the system more robust to LLM hallucinations.
3. **Bounded risk:** The residual is capped both in norm and per component (see Section 5), so even a poorly calibrated LLM output cannot drive the prediction far from the kNN baseline.

4.5 Robust JSON Parsing and Retry Logic

LLM outputs may deviate from the expected format. The parsing module implements a multi-stage recovery strategy:

1. **Strict parsing**: attempt `json.loads` on the raw response.
2. **Regex extraction**: search for an embedded `{...}` JSON blob.
3. **Text salvage**: if enabled, extract floating-point numbers from potentially truncated text after the key `predicted_delta`.
4. **Length repair**: trim or zero-pad the vector if its length does not match K.
5. **Conversational retry**: append the malformed response as an assistant turn and ask the model to correct itself (up to `LLM_MAX_RETRIES` times).

5 Deterministic Stabilisers

After obtaining the combined delta $\boldsymbol{\delta}$ (Equation 19), several deterministic constraints are applied to ensure physical plausibility and prevent forecast divergence.

5.1 Training-Derived Caps

From all consecutive-step deltas in the training set, we compute:

- **Magnitude cap:** the 95th percentile of $\|\boldsymbol{\delta}^{(i)}\|_2$ across training pairs, scaled by a multiplier ρ_{mag} (default 1.2):
$$c_{\text{mag}} = \rho_{\text{mag}} \cdot P_{95}\big(\{\|\boldsymbol{\delta}^{(i)}\|_2\}\big). \tag{20}$$
If $\|\boldsymbol{\delta}\|_2 > c_{\text{mag}}$, the delta is rescaled: $\boldsymbol{\delta} \leftarrow \boldsymbol{\delta} \cdot c_{\text{mag}}/\|\boldsymbol{\delta}\|_2$.

- **Per-component cap:** the 99th percentile of $|\delta_k^{(i)}|$ for each component k, scaled by ρ_{comp} (default 1.2):
$$c_k = \rho_{\text{comp}} \cdot P_{99}\big(\{|\delta_k^{(i)}|\}\big). \tag{21}$$
Each component is clipped: $\delta_k \leftarrow \text{clip}(\delta_k, -c_k, c_k)$.

5.2 Horizon Damping

For multi-step autoregressive rollouts, prediction uncertainty compounds with each step. To counteract drift, the delta magnitude is attenuated exponentially with the forecast horizon h:

$$\boldsymbol{\delta} \;\leftarrow\; \boldsymbol{\delta} \cdot \gamma^{\,(h-1)}, \tag{22}$$

where $\gamma \in (0, 1]$ is the damping factor (default $\gamma = 0.95$). The first horizon step ($h = 1$) is unmodified.

5.3 Velocity-Relative Cap

To prevent the predicted change from being unreasonably large compared to the observed rate of evolution, we impose

$$\|\boldsymbol{\delta}\|_2 \;\le\; \alpha_v \, \|\dot{\tilde{\mathbf{z}}}\|_2 + \beta_v, \tag{23}$$

where α_v (default 6.0) and β_v (default 0.2) are tuneable constants. If the constraint is violated, $\boldsymbol{\delta}$ is rescaled to satisfy it. A minimum cap floor of $0.25 \cdot c_{\text{mag}}$ prevents the constraint from being too restrictive when velocity is very small.

Remark 5.1. When the kNN guide is active, the velocity-relative cap is optionally disabled (`DISABLE_VEL_CAP_WHEN_KNN=True`) because the kNN prior already implicitly regularises the delta magnitude.

5.4 Monotonic SDF Shrinkage

Corrosion is an irreversible process: material can only be removed, never added. In the SDF representation, material loss corresponds to the SDF becoming more positive (or less negative). We enforce this via a pixel-wise maximum:

$$\text{SDF}_{t+1}(\mathbf{x}) \;=\; \max\big(\text{SDF}_{\text{pred}}(\mathbf{x}),\, \text{SDF}_t(\mathbf{x})\big) \quad \forall\, \mathbf{x}. \tag{24}$$

Since $\text{SDF}(\mathbf{x}) \le 0$ indicates material and the maximum operator only *increases* SDF values, this guarantees $\mathcal{M}_{t+1} \subseteq \mathcal{M}_t$ (monotonic shrinkage).

5.5 Optional SDF Smoothing

Before thresholding, the predicted SDF may be lightly smoothed with a Gaussian filter ($\sigma = 0.8$ px) to suppress high-frequency artefacts introduced by PCA truncation:

$$\text{SDF}_{\text{smooth}} = G_\sigma * \text{SDF}_{\text{pred}}, \tag{25}$$

where G_σ is the Gaussian kernel.

5.6 Mask Post-Processing

After thresholding the SDF to obtain a binary mask, morphological post-processing is applied in the following order:

1. **Fill holes:** close interior voids using `binary_fill_holes`.
2. **Remove small components:** discard connected components with fewer than 200 pixels.
3. **Largest component:** keep only the largest connected component (the alloy wire cross-section).
4. **Monotonic mask shrinkage:** intersect the predicted mask with the previous-step mask: $\mathcal{M}_{t+1} \leftarrow \mathcal{M}_{t+1} \cap \mathcal{M}_t$.

6 Monte Carlo Uncertainty Quantification

6.1 Stochastic Rollouts

With the LLM temperature $\tau > 0$, each call to the language model produces a different residual correction, inducing stochasticity in the forecast. We perform R independent autoregressive rollouts (default $R = 8$), each seeded with a distinct `rollout_nonce`. The LLM effective temperature decays across horizons:

$$\tau_{\text{eff}}(h) = \tau_0 \cdot \eta^{\max(0,\, h-1)}, \tag{26}$$

where η (default 0.98) is the horizon temperature decay.

6.2 Probability Map

The pixel-wise probability of material survival is estimated from the ensemble of rollout predictions:

$$P(\text{material} \,|\, \mathbf{x}) \;=\; \frac{1}{R} \sum_{r=1}^{R} \mathbb{I}\big[\mathbf{x} \in \mathcal{M}^{(r)}\big], \tag{27}$$

where $\mathcal{M}^{(r)}$ is the predicted material mask from rollout r and $\mathbb{I}[\cdot]$ is the indicator function.

6.3 Mean Prediction

The consensus (mean) prediction thresholds the probability map at 0.5:

$$\hat{\mathcal{M}} = \big\{\mathbf{x} : P(\text{material} \,|\, \mathbf{x}) \geq 0.5\big\}. \tag{28}$$

Remark 6.1. Regions where $P(\text{material} \,|\, \mathbf{x})$ is close to 0.5 indicate high uncertainty. These typically localise along the advancing corrosion front, providing physically meaningful uncertainty estimates.

7 Evaluation Metrics

7.1 Spatial Overlap Metrics

Let A and B denote the ground-truth and predicted material masks, respectively.

7.1.1 Intersection over Union (IoU)

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}, \tag{29}$$

where $|\cdot|$ denotes the pixel count.

7.1.2 Sørensen–Dice Coefficient

$$\mathrm{Dice}(A, B) = \frac{2\,|A \cap B|}{|A| + |B|}. \tag{30}$$

Remark 7.1. IoU and Dice are related by $\mathrm{Dice} = 2\,\mathrm{IoU}/(1 + \mathrm{IoU})$, so Dice is always $\geq$ IoU for the same prediction. IoU is the stricter metric.

7.2 Boundary F1-Score (BF1)

The boundary F1-score evaluates boundary localisation accuracy. First, extract boundary pixels of each mask via morphological erosion:

$$\partial A = A \;\oplus\; \big(A \ominus \mathcal{B}\big), \tag{31}$$

where $\ominus$ is erosion and $\oplus$ is symmetric difference (XOR). With a tolerance of τ pixels (implemented as dilation by τ iterations):

$$\mathrm{Prec}_\tau = \frac{|\partial B \cap D_\tau(\partial A)|}{|\partial B|}, \tag{32}$$

$$\mathrm{Rec}_\tau = \frac{|\partial A \cap D_\tau(\partial B)|}{|\partial A|}, \tag{33}$$

where $D_\tau(\cdot)$ denotes dilation by τ pixels. The Boundary F1-score is

$$\mathrm{BF1}_\tau = \frac{2\,\mathrm{Prec}_\tau\,\mathrm{Rec}_\tau}{\mathrm{Prec}_\tau + \mathrm{Rec}_\tau}. \tag{34}$$

7.3 Probabilistic Calibration Metrics

Let $p_i \in [0, 1]$ be the predicted probability that pixel i is material, and $y_i \in \{0, 1\}$ the ground-truth label.

7.3.1 Brier Score

$$\mathrm{BS} = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2. \tag{35}$$

Lower is better; a perfect deterministic predictor achieves $\mathrm{BS} = 0$.

7.3.2 Negative Log-Likelihood (NLL)

$$\mathrm{NLL} = -\frac{1}{N}\sum_{i=1}^{N}\big[y_i \log(p_i + \epsilon) + (1 - y_i)\log(1 - p_i + \epsilon)\big], \tag{36}$$

with $\epsilon = 10^{-6}$ for numerical stability.

7.3.3 Expected Calibration Error (ECE)

Partition pixels into B bins by predicted probability. Let $\mathcal{B}_b$ be the set of pixels in bin b, with $n_b = |\mathcal{B}_b|$, mean confidence $\bar{p}_b$, and empirical accuracy $\bar{y}_b$. Then:

$$\mathrm{ECE} = \sum_{b=1}^{B}\frac{n_b}{N}\big|\bar{y}_b - \bar{p}_b\big|. \tag{37}$$

ECE is visualised using a *reliability diagram* (Section 10.11).

8 Auto-Tune PCA

8.1 Motivation

The number of principal components K and the training set size jointly determine reconstruction fidelity. Too few components lose fine-grained boundary detail; too many overfit noise and inflate the latent dimension for downstream predictors.

8.2 Grid Search Protocol

The auto-tune procedure performs a grid search over $(\texttt{n\_train}, K)$ using *oracle reconstruction* on a held-out validation set. Oracle reconstruction projects the ground-truth field onto the PCA basis with K components and then thresholds to obtain a material mask:

$$\hat{\mathbf{x}}^{(K)} = \boldsymbol{\mu} + \sum_{k=1}^{K} \left[(\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{v}_k\right] \mathbf{v}_k, \qquad \hat{\mathcal{M}}^{(K)} = \left\{\mathbf{x} : \hat{\mathbf{x}}^{(K)}(\mathbf{x}) \le 0\right\}. \tag{38}$$

To reduce computational cost, all fields are spatially downsampled by a factor of 2 (default), and only a subset of time-steps $\{0, 3, 6, 9\}$ is used.

8.3 Quality Thresholds

The smallest K satisfying *all* of the following is selected as K_{LLM}:

$$\overline{\text{IoU}}(K) \ge 0.985, \tag{39}$$

$$\overline{\text{Dice}}(K) \ge 0.990, \tag{40}$$

$$\overline{\text{BF1}}(K) \ge 0.930. \tag{41}$$

The reconstruction dimensionality $K_{\text{recon}} \ge K_{\text{LLM}}$ is selected as the point of diminishing returns, where the marginal IoU and Dice gain from an additional component drops below 5×10^{-4}.

8.4 Subspace Stability

To verify that the PCA basis is robust to training set composition, we fit two bases on independent random subsets and measure the *principal angles* between the resulting K-dimensional subspaces:

$$\theta_j = \arccos\big(\sigma_j(\mathbf{V}_A^{(K)\top} \mathbf{V}_B^{(K)})\big), \quad j = 1, \ldots, K, \tag{42}$$

where $\sigma_j(\cdot)$ are singular values. A candidate is accepted only if $\bar{\theta} \le 8^\circ$ (mean principal angle).

Part II
Code Architecture and Documentation

9 Package Structure

The pipeline is implemented as a single Python package with the following layout:

```
corrosion_forecast/
|-- __init__.py
|-- config.py
|-- data_loading.py
|-- sdf_utils.py
|-- pca.py
|-- metrics.py
```

```
|-- knn.py
|-- llm_interface.py
|-- forecasting.py
|-- autotune.py
|-- ablation.py
|-- plotting.py
|-- main.py
`-- requirements.txt
```

Dependencies:

```
numpy>=1.24       scipy>=1.10      matplotlib>=3.7
pandas>=2.0       imageio>=2.31    requests>=2.28
scikit-learn>=1.3
```

10 Module Descriptions

10.1 `config.py` — Global Configuration

Purpose: Centralises all tuneable hyperparameters, file-system paths, and feature flags so that experiments are reproducible and modifications require editing only a single file.

10.1.1 Key Constants

Group	Parameter	Default
Paths	`BASE_DIR`	`../Corrosion_Masks` (or `$CORROSION_BASE_DIR`)
	`METADATA_CSV`	`<BASE_DIR>/metadata.csv`
	`NUM_TIMESTEPS`	10
LLM	`OPENROUTER_API_KEY`	`$OPENROUTER_API_KEY`
	`LLM_MODEL`	`openai/gpt-4o-mini`
	`LLM_TEMPERATURE`	0.6
	`LLM_MAX_TOKENS`	600
	`LLM_MAX_RETRIES`	2
PCA	`K_LLM`	40
	`K_PCS_MAX`	80
	`NUM_TRAIN_SLICES`	400
kNN	`KNN_K`	16
	`KNN_MAX_SLICES`	300
	`RESIDUAL_SCALE`	0.7
Stabilisers	`HORIZON_DAMP`	0.95
	`VEL_REL_MULT`	6.0
	`VEL_REL_BIAS`	0.20
Experiment	`NUM_TEST_SLICES`	2
	`TEST_START_TIMES`	$[3, 5]$
	`NUM_ROLLOUTS`	8

Table 1: Selected configuration parameters and their defaults.

10.1.2 Dependencies

Standard library only (`os`), plus `numpy` for dtype constants.

10.2 `data_loading.py` — Dataset I/O

Purpose: Handles all file-system interaction: reading TIFF corrosion mask slices, parsing the metadata CSV, binarisation, and in-memory frame caching.

10.2.1 Key Functions

```python
def load_frame_uint8(
    slice_idx: int, t: int, use_cache: bool = True
) -> Optional[np.ndarray]:
    """Load a single grayscale frame as uint8.
    Returns None if the file does not exist."""

def load_slice_across_time(
    slice_idx: int, use_cache: bool = False
) -> Optional[List[np.ndarray]]:
    """Load one slice index across all NUM_TIMESTEPS time-steps.
    Returns list of (H, W) uint8 images, or None if any missing."""

def get_total_slice_count() -> int:
    """Count .tif files in the first time-step folder."""

def load_times_from_metadata() -> np.ndarray:
    """Return degradation times (hours) from metadata.csv."""

def binarize_0_255(img: np.ndarray, thr: int = 128) -> np.ndarray:
    """Threshold to {0, 255}."""

def to_material_bool(img_0_255: np.ndarray) -> np.ndarray:
    """Convert {0,255} mask to boolean (True = material)."""

def from_material_bool(material_bool: np.ndarray) -> np.ndarray:
    """Inverse of to_material_bool."""

def measure_area_material(material_bool: np.ndarray) -> int:
    """Count material pixels."""
```

10.2.2 Frame Caching

A module-level dictionary `_FRAME_CACHE` maps `(slice_idx, t)` to the loaded `uint8` array. The cache is transparent to callers and can be cleared via `clear_frame_cache()`.

10.2.3 Data Convention

In the TIFF files, pixel value 0 (black) denotes **material** and pixel value 255 (white) denotes **background/void**. The helper `to_material_bool` implements `img == 0`.

10.2.4 Dependencies

`imageio`, `numpy`, `pandas`, `config`.

10.3 `sdf_utils.py` — SDF Utilities

Purpose: Converts between boolean material masks and Signed Distance Fields, and provides morphological post-processing routines.

10.3.1 Key Functions

```python
def material_to_sdf(material_bool: np.ndarray) -> np.ndarray:
    """Boolean mask -> SDF (float32). Negative inside material."""

def sdf_to_material(sdf: np.ndarray) -> np.ndarray:
    """SDF -> boolean mask via zero level-set (SDF <= 0)."""

def keep_largest_component(mask_bool: np.ndarray) -> np.ndarray:
    """Keep only the largest connected component."""

def remove_small_components(
    mask_bool: np.ndarray, min_pixels: int = 200
) -> np.ndarray:
    """Remove components smaller than min_pixels."""

def postprocess_material(
    pred_material: np.ndarray,
    prev_material: np.ndarray = None,
) -> np.ndarray:
    """Apply cascade: fill holes -> remove small ->
    keep largest -> monotonic shrink."""

def smooth_sdf(sdf: np.ndarray) -> np.ndarray:
    """Optional Gaussian smoothing (sigma from config)."""
```

10.3.2 Post-Processing Cascade

The `postprocess_material` function applies, in order:

1. `binary_fill_holes` (if `FILL_HOLES`).
2. `remove_small_components` (if `MIN_COMPONENT_PIXELS` > 0).
3. `keep_largest_component` (if `ENFORCE_SINGLE_COMPONENT`).
4. Monotonic shrinkage via boolean AND with `prev_material` (if `ENFORCE_MONOTONIC_SHRINK`).

Each step is gated by the corresponding configuration flag.

10.3.3 Dependencies

`scipy.ndimage`, `numpy`, `config`.

10.4 `pca.py` — PCA Basis Fitting

Purpose: Fits a global PCA basis from training slices via (randomised) SVD, and provides projection/reconstruction utilities.

10.4.1 Key Functions

```python
def fit_global_pca(
    num_slices: int,
    max_index: int,
    k_max: int = 60,
    seed: int = 0,
) -> Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]:
    """Fit PCA basis from training slices.
    Returns: (mean_field, Vt, z_mean, z_std)."""

```

```python
def project_field(
    field_2d: np.ndarray, mean_field: np.ndarray, Vt: np.ndarray
) -> np.ndarray:
    """Project a 2D field onto PCA basis -> latent z."""

def reconstruct_field(
    z: np.ndarray, mean_field: np.ndarray,
    Vt: np.ndarray, H: int, W: int
) -> np.ndarray:
    """Reconstruct a 2D field from latent vector z."""
```

10.4.2 Implementation Details

- Training slice IDs are randomly sampled up to `num_slices` from IDs in $[0, \texttt{max\_index})$.
- For each slice, *all* T time-steps are included in the training matrix, yielding up to $T \times$ `num_slices` rows.
- If `scikit-learn` is available, `randomized_svd` is used; otherwise the code falls back to full SVD.
- The function returns both the PCA basis (`mean_field`, `Vt`) and the latent normalisation statistics (`z_mean`, `z_std`).

10.4.3 Dependencies

`numpy`, `sklearn` (optional), `data_loading`, `sdf_utils`, `config`.

10.5 `metrics.py` — Evaluation Metrics

Purpose: Implements all evaluation metrics used for mask quality assessment and probabilistic calibration.

10.5.1 Key Functions

```python
def iou(gt: np.ndarray, pr: np.ndarray) -> float:
    """Intersection over Union for boolean masks."""

def dice(gt: np.ndarray, pr: np.ndarray) -> float:
    """Sorensen-Dice coefficient for boolean masks."""

def boundary_f1(
    gt: np.ndarray, pr: np.ndarray, tol: int = 1
) -> float:
    """Boundary F1-score with tolerance (dilation)."""

def brier_score(probs: np.ndarray, y: np.ndarray) -> float:
    """Mean squared error between probabilities and labels."""

def binary_nll(
    probs: np.ndarray, y: np.ndarray, eps: float = 1e-6
) -> float:
    """Negative log-likelihood for binary predictions."""

def expected_calibration_error(
    probs: np.ndarray, y: np.ndarray, n_bins: int = 15
) -> float:
    """Expected Calibration Error (ECE)."""
```

10.5.2 Dependencies

`numpy`, `scipy.ndimage` (for boundary extraction in BF1).

10.6 `knn.py` — kNN Delta Predictor

Purpose: Constructs a reference library of (feature, delta) pairs from training sequences and provides inverse-distance-weighted k-nearest neighbour predictions in PCA latent space.

10.6.1 Key Functions

```python
def build_knn_library(
    train_ids: list,
    mean_field: np.ndarray,
    Vt: np.ndarray,
    z_mean: np.ndarray,
    z_std: np.ndarray,
    k_llm: int,
    times_h_all: np.ndarray,
    max_slices: int = 300,
    seed: int = 0,
) -> Dict[str, np.ndarray]:
    """Build kNN library. Returns dict with keys
    'X' (features), 'Y' (deltas), 'tmax'."""

def knn_predict_delta(
    knn_lib: Dict[str, np.ndarray],
    z_last: np.ndarray,
    vel: np.ndarray,
    t_hours: float,
    dt_hours: float,
    k: int = 16,
) -> Tuple[np.ndarray, np.ndarray]:
    """Predict next-step delta via IDW kNN.
    Returns (mean_delta, std_delta)."""
```

10.6.2 Feature Construction

The internal function `_feat_from_state` builds the feature vector as defined in Equation (13): the last K-dimensional normalised latent, the velocity vector, and two normalised time scalars. This yields a feature of dimension $2K + 2$.

10.6.3 Dependencies

`numpy`, `data_loading`, `pca`, `config`.

10.7 `llm_interface.py` — LLM API Interface

Purpose: Handles prompt construction, HTTP communication with the OpenRouter API, and robust JSON parsing of LLM responses.

10.7.1 Key Functions

```python
def build_llm_prompt(
    latents_norm_hist_top: np.ndarray,
    dt_h: float,
    rollout_nonce: Optional[float] = None,
    meta_row: Optional[Dict] = None,
```

```
    delta_prior: Optional[np.ndarray] = None,
    prior_std: Optional[np.ndarray] = None,
    residual_norm_cap: Optional[float] = None,
    residual_comp_cap: Optional[float] = None,
) -> str:
    """Construct the user-message prompt as JSON payload."""

def parse_llm_delta(
    text: str,
    D: int,
    allow_length_repair: bool = False,
    allow_text_salvage: bool = False,
) -> Tuple[Optional[np.ndarray], Dict]:
    """Parse LLM response -> delta vector of length D.
    Multi-stage: strict JSON, regex, salvage, length repair."""

def call_llm_delta(
    latents_norm_hist_top: np.ndarray,
    dt_h: float,
    rollout_nonce: Optional[float] = None,
    meta_row: Optional[Dict] = None,
    horizon: int = 1,
    delta_prior: Optional[np.ndarray] = None,
    prior_std: Optional[np.ndarray] = None,
    residual_norm_cap: Optional[float] = None,
    residual_comp_cap: Optional[float] = None,
) -> Tuple[np.ndarray, str, int, Dict]:
    """Full LLM call with retries.
    Returns (delta_vec, raw_text, http_status, info)."""
```

10.7.2 Retry and Recovery Strategy

The call loop (up to `LLM_MAX_RETRIES`+1 attempts) follows Algorithm 1.

Algorithm 1 LLM call with progressive recovery

for attempt $= 0, 1, \ldots,$ `MAX_RETRIES` **do**
  Send prompt to LLM via HTTP POST
  Parse response with strict JSON
  if parse succeeds and length matches K **then**
    return parsed delta
  end if
  if length mismatch and retries remain **then**
    Append repair message; **continue**
  end if
  if parse failure and retries remain **then**
    Append correction message; **continue**
  end if
  Attempt text salvage and length repair (last resort)
  if salvage succeeds **then return** salvaged delta
  end if
  raise RuntimeError
end for

10.7.3 HTTP Details

- Endpoint: `https://openrouter.ai/api/v1/chat/completions`.

- Authentication: Bearer token via `OPENROUTER_API_KEY`.
- The `response_format` is set to `{"type":"json_object"}` when `USE_RESPONSE_FORMAT_JSON=True`, requesting structured output.
- The effective temperature decays with horizon: $\tau_{\text{eff}} = \tau_0 \cdot 0.98^{h-1}$.

10.7.4 Dependencies

`json`, `re`, `requests`, `numpy`, `config`.

10.8 `forecasting.py` — SDF Forecasting Engine

Purpose: Implements the single-step SDF forecast (kNN prior + LLM residual + stabilisers) and the Monte Carlo rollout loop for multi-step autoregressive prediction.

10.8.1 Key Functions

```python
def compute_training_caps(
    train_ids: List[int],
    mean_field: np.ndarray,
    Vt: np.ndarray,
    z_mean: np.ndarray,
    z_std: np.ndarray,
    k_llm: int,
    max_slices: int = 250,
    seed: int = 0,
) -> Dict[str, np.ndarray]:
    """Compute magnitude and per-component caps from
    training GT deltas."""

def clip_delta_to_training(
    delta: np.ndarray,
    caps: Optional[Dict],
    horizon: int = 1,
) -> np.ndarray:
    """Apply training caps + horizon damping."""

def apply_velocity_relative_cap(
    delta: np.ndarray,
    vel: np.ndarray,
    mult: float = 6.0,
    bias: float = 0.2,
    min_cap: Optional[float] = None,
) -> np.ndarray:
    """Cap delta magnitude relative to velocity."""

def forecast_next_sdf(
    history_sdfs: List[np.ndarray],
    history_times_h: np.ndarray,
    mean_field: np.ndarray,
    Vt: np.ndarray,
    z_mean: np.ndarray,
    z_std: np.ndarray,
    k_llm: int,
    meta_df=None,
    horizon: int = 1,
    rollout_nonce: Optional[float] = None,
    caps: Optional[Dict] = None,
    knn_lib: Optional[Dict] = None,
) -> Tuple[np.ndarray, Dict]:
    """Single-step SDF forecast. Returns (sdf_next, debug_dict)."""
```

```python

def rollout_mc(
    full_gt_imgs: List[np.ndarray],
    times_h_all: np.ndarray,
    start_t: int,
    mean_field: np.ndarray,
    Vt: np.ndarray,
    z_mean: np.ndarray,
    z_std: np.ndarray,
    k_llm: int,
    caps: Optional[Dict] = None,
    meta_df=None,
    knn_lib: Optional[Dict] = None,
    n_rollouts: int = 8,
    verbose: bool = True,
) -> Tuple[List[Dict], List[np.ndarray]]:
    """Run R autoregressive rollouts from start_t.
    Returns (records, final_preds)."""
```

10.8.2 Single-Step Pipeline (`forecast_next_sdf`)

The function implements the complete single-step pipeline:

1. Project history SDFs into normalised PCA latent space.
2. Compute kNN prior delta $\boldsymbol{\delta}_{\text{prior}}$ and uncertainty σ.
3. Compute residual caps: $\|\boldsymbol{\delta}_{\text{LLM}}\|_2 \leq \texttt{RESIDUAL\_NORM\_FRAC} \cdot c_{\text{mag}}$ and per-component cap $\max(0.25,\ 3 \cdot \text{median}(\sigma))$.
4. Call LLM for residual correction.
5. Combine: $\boldsymbol{\delta} = \boldsymbol{\delta}_{\text{prior}} + \alpha \cdot \boldsymbol{\delta}_{\text{LLM}}$.
6. Apply training caps (Section 5.1).
7. Apply velocity-relative cap (Equation 23).
8. Update latent: $\tilde{\mathbf{z}}_{t+1} = \tilde{\mathbf{z}}_t + \boldsymbol{\delta}$.
9. Reconstruct SDF from updated physical-space latent.
10. Smooth SDF, enforce monotonic shrinkage (Section 5.4).

10.8.3 Dependencies

`numpy`, `data_loading`, `knn`, `llm_interface`, `metrics`, `pca`, `sdf_utils`, `config`.

10.9 `autotune.py` — PCA Auto-Tuning

Purpose: Performs a downsampled grid search over PCA hyperparameters (n_{train}, K) to automatically select values that meet quality thresholds while ensuring subspace stability.

10.9.1 Key Functions

```python
def preload_autotune_fields(
    slice_ids: List[int],
    timesteps: List[int],
    downsample_factor: int = 2,
) -> None:
```

```
    """Eagerly load fields into cache for fast grid search."""

def fit_pca_on_indices(
    indices: List[int],
    k_fit_max: int,
    timesteps: List[int],
    downsample_factor: int = 2,
    seed: int = 0,
    use_cache: bool = True,
) -> Tuple[np.ndarray, np.ndarray, Tuple[int, int]]:
    """Fit PCA basis on downsampled fields."""

def oracle_score_curve(
    val_indices: List[int],
    mean_field: np.ndarray,
    Vt: np.ndarray,
    K_grid: List[int],
    timesteps: List[int],
    downsample_factor: int = 2,
    use_cache: bool = True,
) -> pd.DataFrame:
    """Evaluate oracle reconstruction for each K.
    Returns DataFrame with columns K, iou_mean, dice_mean, bf1_mean."""

def subspace_distance_deg(
    VtA: np.ndarray, VtB: np.ndarray, K: int
) -> Tuple[float, float]:
    """Mean and max principal angle (degrees) between
    two K-dim subspaces."""

def auto_tune_pca(
    train_max_index: int, seed: int = 0
) -> Tuple[int, int, int]:
    """Grid-search for (n_train, K_LLM, K_PCS_MAX).
    Returns (tuned_n_train, tuned_k_llm, tuned_k_recon)."""
```

10.9.2 Grid Configuration

The search grid is defined in `config.py`:

- `TUNE_TRAIN_GRID`: $[400, 800]$ training slices.
- `TUNE_K_GRID`: $[40, 60, 80, 120, 160]$ components.
- `TUNE_TIMESTEPS`: $[0, 3, 6, 9]$ sampled time-steps.
- `TUNE_DOWNSAMPLE`: factor 2 spatial downsampling.
- `TUNE_VAL_SLICES`: 25 held-out validation slices.

10.9.3 Selection Logic

1. For each n_{train}, fit PCA on downsampled training fields.
2. Evaluate oracle reconstruction on validation set for each K.
3. Select smallest K_{LLM} meeting IoU/Dice/BF1 thresholds.
4. Select K_{recon} at diminishing-returns elbow.
5. Verify subspace stability via principal angles (Section 8.4).
6. Among all stable candidates, select the one with the smallest n_{train} and then the smallest K_{LLM}.

10.9.4 Dependencies

`numpy`, `pandas`, `sklearn` (optional), `data_loading`, `metrics`, `pca`, `sdf_utils`, `config`.

10.10 `ablation.py` — Ablation Study Runner

Purpose: Compares pipeline variants by temporarily overriding configuration parameters and running the evaluation loop. Supports checkpoint/resume via `pickle` serialisation.

10.10.1 Key Functions

```python
def run_ablation_variant(
    name: str,
    valid_slices: List[int],
    times_h_all: np.ndarray,
    mean_field: np.ndarray,
    Vt: np.ndarray,
    z_mean: np.ndarray,
    z_std: np.ndarray,
    k_llm: int,
    caps: Optional[Dict],
    meta_df,
    knn_lib: Optional[Dict],
    n_rollouts: int,
    llm_temperature: Optional[float] = None,
    residual_scale: Optional[float] = None,
    use_knn_guide: Optional[bool] = None,
) -> Tuple[List[Dict], float]:
    """Run one ablation variant. Returns (rows, runtime_s)."""

def run_all_ablations(
    valid_slices: List[int],
    times_h_all: np.ndarray,
    mean_field: np.ndarray,
    Vt: np.ndarray,
    z_mean: np.ndarray,
    z_std: np.ndarray,
    k_llm: int,
    caps: Optional[Dict],
    meta_df,
    knn_lib: Optional[Dict],
) -> pd.DataFrame:
    """Run all four ablation variants with checkpoint/resume."""
```

10.10.2 Ablation Variants

Variant	Rollouts	Temperature	Residual scale	kNN
MC ensemble (baseline)	$R = 8$	0.60	0.70	yes
Single-shot	1	0.60	0.70	yes
Deterministic	1	0.00	0.70	yes
kNN-only (no LLM)	1	0.60	0.00	yes

Table 2: Ablation study variants.

10.10.3 Checkpoint/Resume

After each variant completes, the accumulated rows are serialised to `ablation_partial.pkl`. On restart, completed variants (identified by name) are skipped. This ensures that long-running

ablation studies survive interruptions.

10.10.4 Dependencies

`pandas`, `numpy`, `requests`, `data_loading`, `forecasting`, `config`.

10.11 `plotting.py` — Visualisation Routines

Purpose: Provides 11 publication-ready visualisation functions using only `matplotlib` (no `seaborn` dependency). All functions apply a consistent style via `rcParams`.

10.11.1 Available Plots

1. `plot_mask_timeseries` — Raw mask sequence across all time-steps for a single slice.
2. `plot_area_vs_time` — Material pixel count versus degradation time.
3. `plot_explained_variance` — Cumulative PCA explained variance versus K.
4. `plot_pca_reconstructions` — Side-by-side ground truth versus PCA reconstructions at various K values.
5. `plot_eigenimages` — First n principal component images ("corrosion modes").
6. `plot_metric_vs_horizon` — Mean $\pm$ std of a metric (IoU, Dice, area error) versus forecast horizon.
7. `plot_ablation_comparison` — Multi-variant comparison (mean $\pm$ std) on the same axes.
8. `plot_delta_over_baseline` — ΔIoU gain of each variant over the kNN-only baseline.
9. `plot_gallery` — Side-by-side GT, mean prediction, XOR difference, and P(material) uncertainty map.
10. `plot_reliability_diagram` — Calibration curve with ECE annotation.
11. `plot_risk_coverage` — Risk (mean pixel error) versus coverage for uncertainty utility assessment.

10.11.2 Dependencies

`matplotlib`, `numpy`, `pandas`.

10.12 `main.py` — Pipeline Orchestrator

Purpose: Serves as the single entry point that orchestrates the entire pipeline from data loading to final plots.

10.12.1 Execution Sequence

1. **Dataset discovery:** Count total slices; compute 80/20 train/test split index.
2. **Auto-tune PCA** (Section 8): if enabled, run grid search and update `cfg.NUM_TRAIN_SLICES`, `cfg.K_LLM`, `cfg.K_PCS_MAX`.
3. **Fit global PCA:** call `fit_global_pca` with tuned parameters.
4. **Load metadata:** degradation times from CSV.

5. **Compute training caps:** P95/P99 statistics on training deltas.
6. **Build kNN library:** feature–delta pairs from training sequences.
7. **Select test slices:** randomly sample `NUM_TEST_SLICES` from the test partition with valid data.
8. **Run MC rollouts:** for each test slice and each `start_t`, perform `NUM_ROLLOUTS` autoregressive forecasts. Collect per-step metrics and final predictions.
9. **Generate plots:** metric-vs-horizon, gallery, ablation comparison.
10. **Run ablation study:** four variants with checkpoint/resume.
11. **Uncertainty diagnostics:** compute Brier, NLL, ECE; generate reliability diagram and risk–coverage curve.

10.12.2 Usage

```
export OPENROUTER_API_KEY="sk-or-v1-your-key-here"
python -m corrosion_forecast.main
```

10.12.3 Dependencies

All other modules in the package.

11 Data Flow Diagram

Figure 1 shows the end-to-end data flow through the pipeline. Each box represents a transformation stage, and arrows indicate data dependencies.

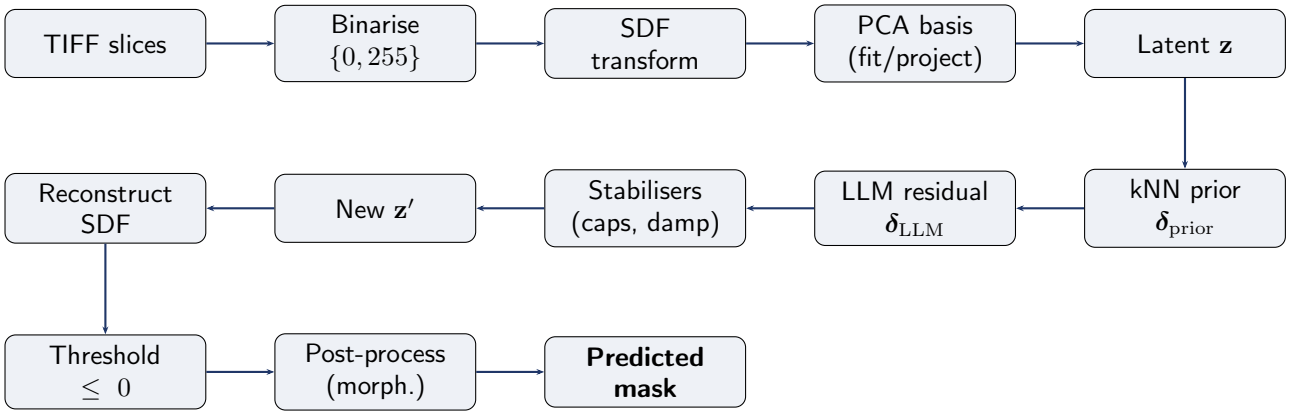

Figure 1: End-to-end data flow of the corrosion forecasting pipeline.

Textual summary of the data flow:

> TIFF slices → binarise → SDF → PCA basis → latent $\mathbf{z}$ → kNN prior → LLM residual → stabilise delta → new $\mathbf{z}'$ → reconstruct SDF → threshold → post-process → predicted mask.

Variable	Description
`OPENROUTER_API_KEY`	API key for the OpenRouter LLM endpoint. **Required** for any run that involves LLM calls.
`CORROSION_BASE_DIR`	Override the path to the `Corrosion_Masks` dataset directory. Defaults to `../Corrosion_Masks` relative to the package.

Table 3: Environment variables used by the pipeline.

12 Configuration and Reproducibility

12.1 Environment Variables

12.2 Modifying Hyperparameters

All hyperparameters are defined as module-level constants in `config.py` (Section 10.1). To change a parameter for an experiment:

1. Edit the constant directly in `config.py`.
2. Alternatively, override it programmatically before calling `main()`:

```python
from corrosion_forecast import config as cfg
cfg.K_LLM = 60
cfg.NUM_ROLLOUTS = 16
cfg.LLM_TEMPERATURE = 0.8

from corrosion_forecast.main import main
main()
```

12.3 Random Seeds

The pipeline uses a fixed seed (`seed=0` by default) for all stochastic operations: NumPy random, Python `random` module, and the `random_state` parameter of randomised SVD. The kNN library construction and PCA training slice sampling are both seeded. The LLM itself is inherently stochastic (temperature > 0), which is the *intended* source of variability across Monte Carlo rollouts.

12.4 Dataset Layout

The expected directory structure under `BASE_DIR` is:

```
Corrosion_Masks/
|-- metadata.csv
|-- Processed_0/
|   |-- 0.tif
|   |-- 1.tif
|   `-- ...
|-- Processed_1/
|   |-- 0.tif
|   `-- ...
|-- ...
`-- Processed_9/
    |-- 0.tif
    `-- ...
```

Each `Processed_t/` directory contains one TIFF file per slice index, and the integer t indexes the degradation time-step (0 through 9). The file `metadata.csv` contains a column `Degradation Time (h)` with the physical time in hours for each time-step.

12.5 Reproducibility Checklist

1. Set `OPENROUTER_API_KEY` and, if necessary, `CORROSION_BASE_DIR`.
2. Install dependencies: `pip install -r corrosion_forecast/requirements.txt`.
3. Run: `python -m corrosion_forecast.main`.
4. Results will be printed to stdout; plots will be displayed interactively or saved if `plt.savefig` calls are added.
5. The ablation checkpoint file (`ablation_partial.pkl`) enables resuming interrupted runs.

References

[1] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[2] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] OpenRouter, "Unified API for LLMs," https://openrouter.ai/, 2024.

[4] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[5] M. P. Naeini, G. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using Bayesian binning," *Proc. AAAI Conference on Artificial Intelligence*, 2015.

[6] G. Csurka, D. Larlus, and F. Perronnin, "What is a good evaluation measure for semantic segmentation?" *Proc. BMVC*, 2013.

[7] J. A. Sethian, *Level Set Methods and Fast Marching Methods*, Cambridge University Press, 2nd edition, 1999.

[8] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *Proc. ICML*, 2017.

Part III — Experimental Results

The following results were obtained by running the full pipeline on the Mg–4Ag in-situ nano-CT dataset described in Reimers et al. [**?**]. The dataset contains 1372 2D slices across 10 time-steps, with an 80/20 train/test split (slices 0–1096 for training, 1097–1371 for testing). All LLM calls used GPT-4o-mini via the OpenRouter API.

13 Dataset and SDF Representation

Figure 2 shows representative corrosion mask sequences across all ten time-steps. The progressive material loss due to biodegradation is clearly visible, with increasingly irregular boundary morphology at later time-steps.

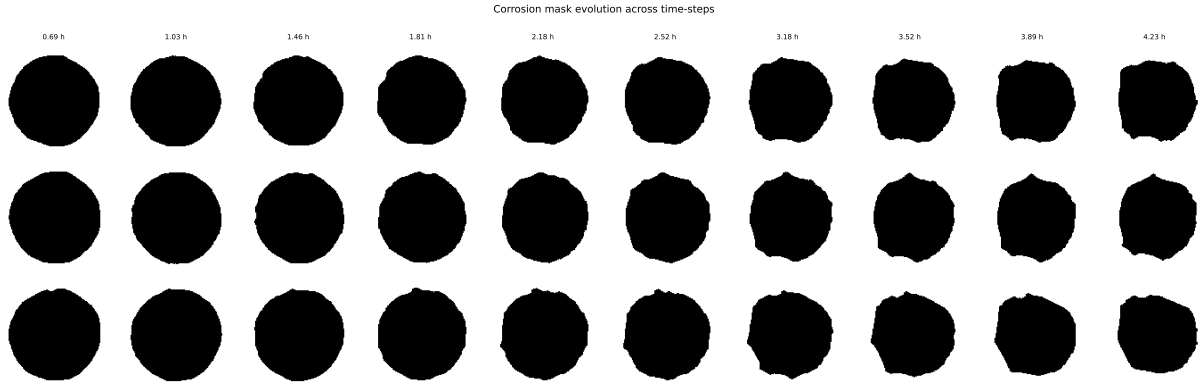

Figure 2: Corrosion mask evolution for three representative slices across all ten degradation time-steps (0.69–4.23 h). White pixels indicate remaining material; black indicates corroded/dissolved regions.

Figure 3 quantifies the material loss as a function of degradation time. Different slices show varying degradation rates, highlighting the spatial inhomogeneity of the corrosion process noted in the original study.

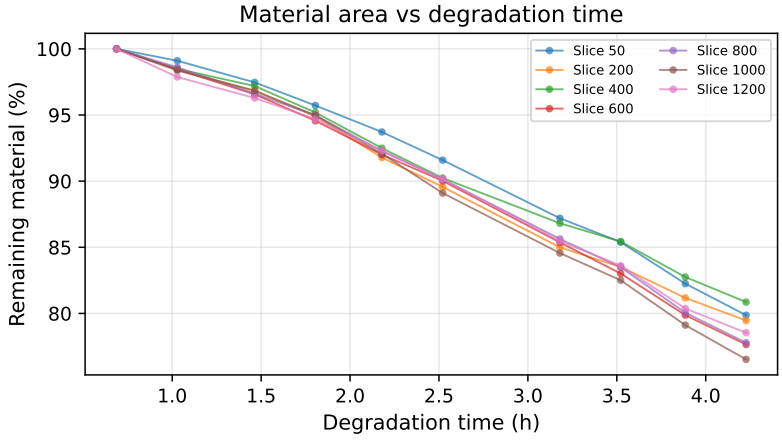

Figure 3: Remaining material area (normalised to initial area) vs. degradation time for seven representative slices. The spread across slices reflects the inhomogeneous degradation of Mg–4Ag wires.

Figure 4 illustrates the Signed Distance Field representation. The continuous-valued SDF encodes both the mask geometry and boundary proximity, making it far more amenable to PCA

decomposition than raw binary images.

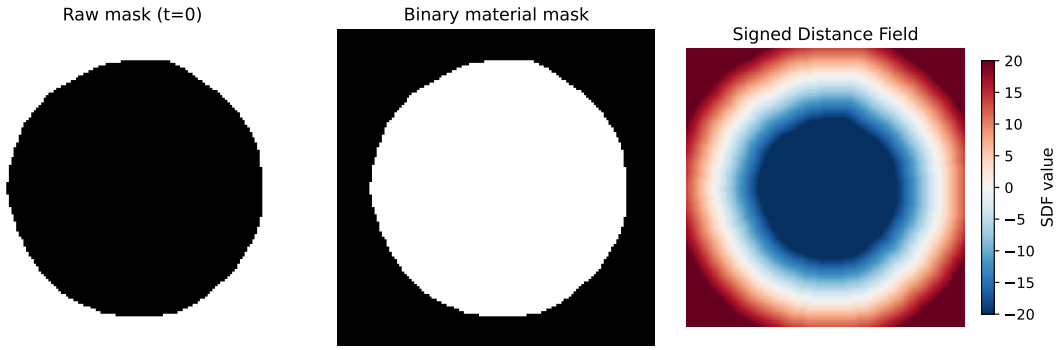

Figure 4: From left to right: raw greyscale mask, binary material mask (black = material), and the corresponding SDF. In the SDF, negative values (red) indicate material interior, positive values (blue) indicate the exterior, and the zero level-set defines the boundary.

14 PCA Basis Analysis

A global PCA basis was fitted on 400 training slices (4000 SDF fields in total: 400 slices × 10 time-steps) using randomised SVD with $K_{\max} = 80$ components.

14.1 Explained Variance

Figure 5 shows that 99% of the variance is captured with only $K = 24$ components, and 99.9% with $K = 64$. This confirms that the SDF representation of the corrosion masks lies on a low-dimensional manifold well suited for PCA compression.

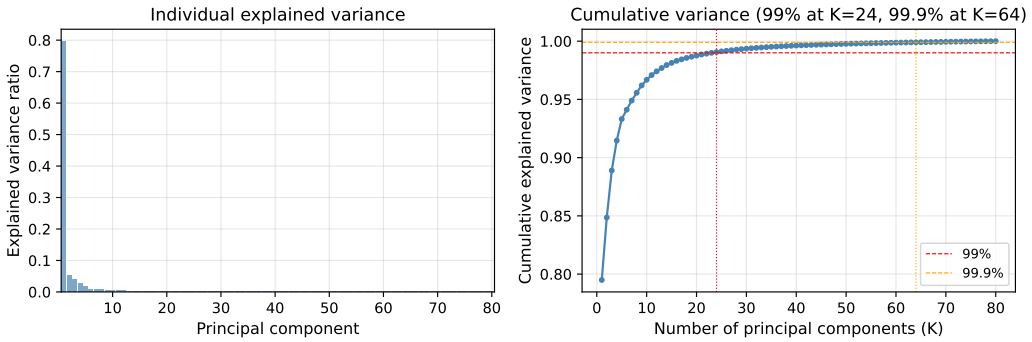

Figure 5: Left: individual explained variance per principal component. Right: cumulative explained variance. The 99% threshold is reached at $K = 24$; 99.9% at $K = 64$.

14.2 Principal Component Images

Figure 6 visualises the first eight principal components ("corrosion modes") reshaped to the original image dimensions. PC 1 captures the overall material presence and dominates with 50.2% of the variance. Higher-order components encode increasingly fine boundary deformations and localised corrosion patterns.

14.3 Oracle Reconstruction Quality

Figure 7 evaluates the oracle reconstruction quality (project the true test field into the PCA basis and reconstruct) as a function of K. At $K = 40$, oracle IoU reaches 0.9946, Dice reaches 0.9973,

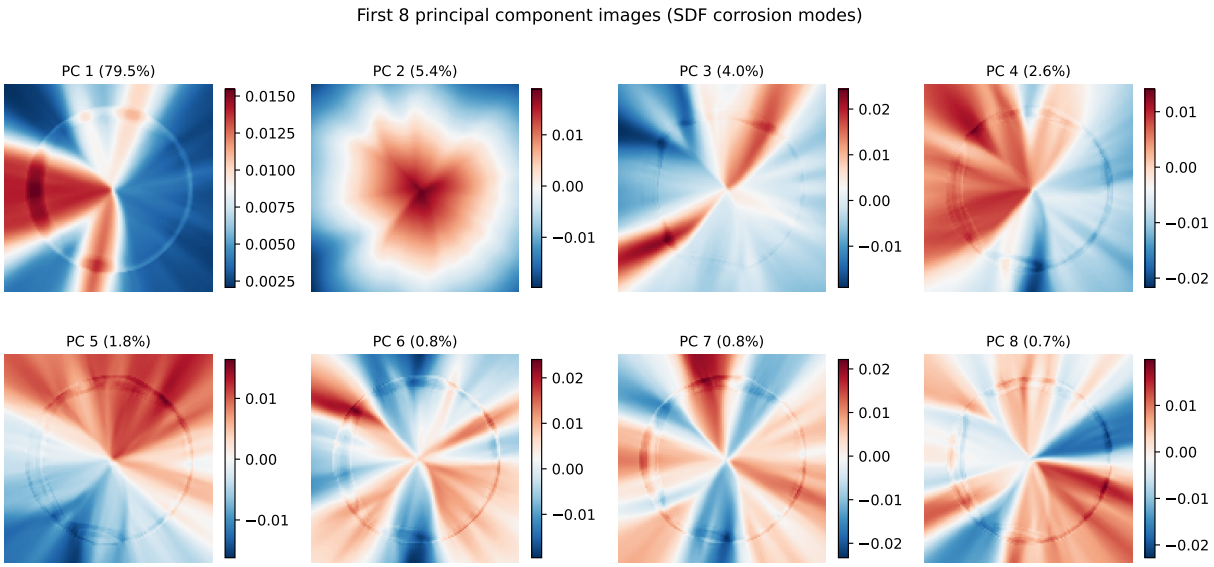

Figure 6: First eight principal component images (SDF "corrosion modes"). Each image is normalised individually for visualisation. The percentage indicates the fraction of total variance explained by that component.

and BF1 reaches 0.9998. This sets an upper bound on forecasting performance: no forecast can exceed the oracle reconstruction quality for the chosen K.

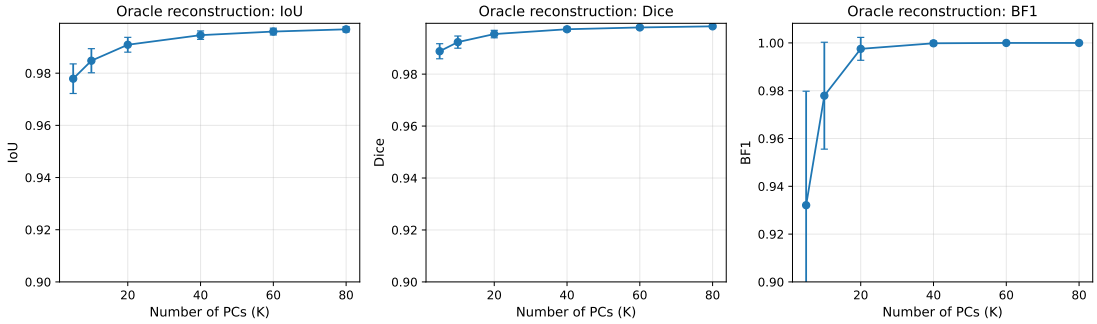

Figure 7: Oracle reconstruction quality (mean ± std) on 50 validation slices, evaluated at four time-steps per slice. IoU, Dice, and BF1 all converge rapidly; $K = 40$ achieves near-perfect reconstruction.

Figure 8 provides a visual comparison of oracle reconstructions at different K. Even at $K = 10$, the overall shape is captured; $K = 40$ is visually indistinguishable from the ground truth.

15 Forecast Results

Forecasting was evaluated on 4 test slices (indices 1154, 1109, 1237, 1222), each from two starting time-steps ($t_0 = 3$ and $t_0 = 5$), yielding 8 forecasting experiments. The MC LLM+kNN variant used 4 rollouts per experiment.

15.1 Forecast Quality vs. Horizon

Figure 9 shows IoU, Dice, and BF1 as a function of forecast horizon (number of autoregressive steps into the future). IoU degrades gracefully from 0.98 at horizon 1 to approximately 0.91 at horizon 7. Dice remains above 0.95 throughout. BF1 (boundary accuracy) degrades more rapidly,

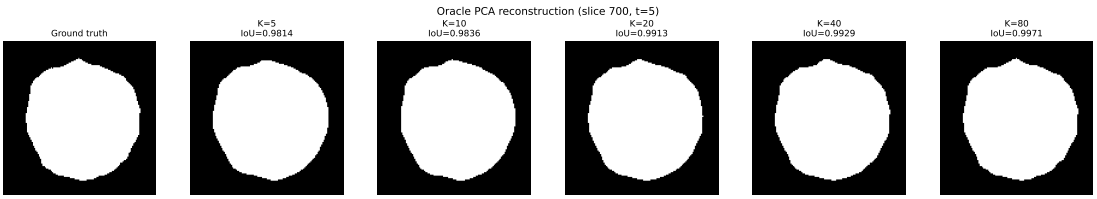

Figure 8: Visual comparison of oracle PCA reconstruction at different numbers of components K vs. the ground-truth material mask.

dropping from 0.97 at horizon 1 to about 0.48 at horizon 6, indicating that the model preserves overall volume well but loses fine boundary detail at longer horizons.

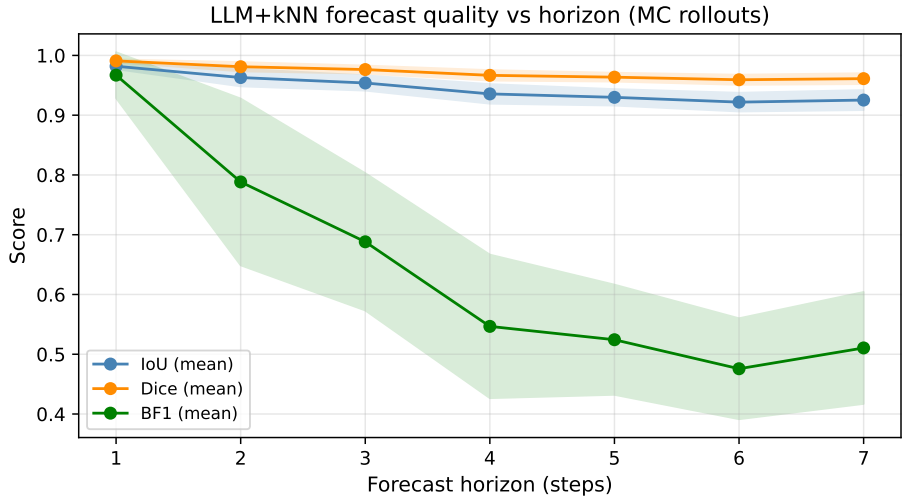

Figure 9: Forecast quality (IoU, Dice, BF1) vs. forecast horizon for the MC LLM+kNN pipeline. Shaded regions show $\pm 1\sigma$ across rollouts and test slices. IoU and Dice remain above 0.91 and 0.95 respectively even at horizon 7.

15.2 Area Error vs. Horizon

Figure 10 shows the absolute pixel-count error in the predicted material area. The mean error grows roughly linearly with horizon, reflecting the accumulated prediction drift.

16 Ablation Study

Three pipeline variants were compared:

- **MC LLM+kNN**: 4 rollouts, temperature = 0.6, residual scale = 0.7
- **Deterministic (T=0)**: single rollout, temperature = 0
- **kNN-only**: no LLM calls; uses only the kNN prior as the predicted delta

16.1 Ablation: IoU and Dice Comparison

Figure 11 compares the three variants. Table 4 reports the aggregated scores. The **Deterministic LLM (T=0) achieves the best overall IoU** (0.9512), outperforming both kNN-only (0.9492) and MC LLM+kNN (0.9480). While the differences are within one standard deviation, the consistent improvement of the deterministic LLM variant indicates that the LLM residual correction provides a measurable benefit when the stochastic sampling noise is eliminated.

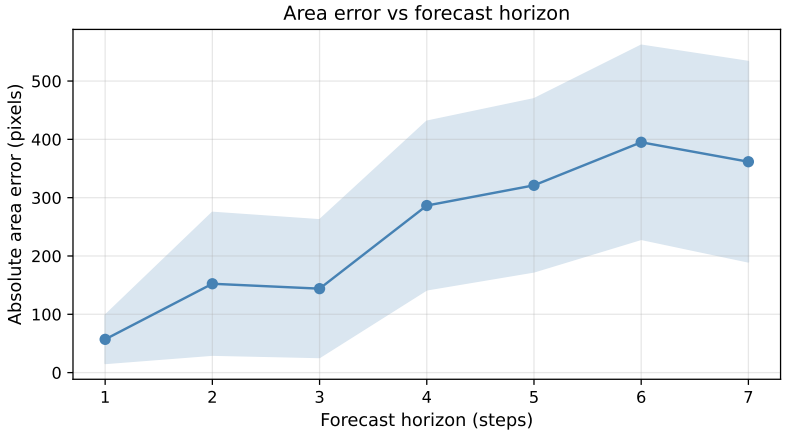

Figure 10: Absolute area error (pixels) vs. forecast horizon. Error grows approximately linearly with horizon.

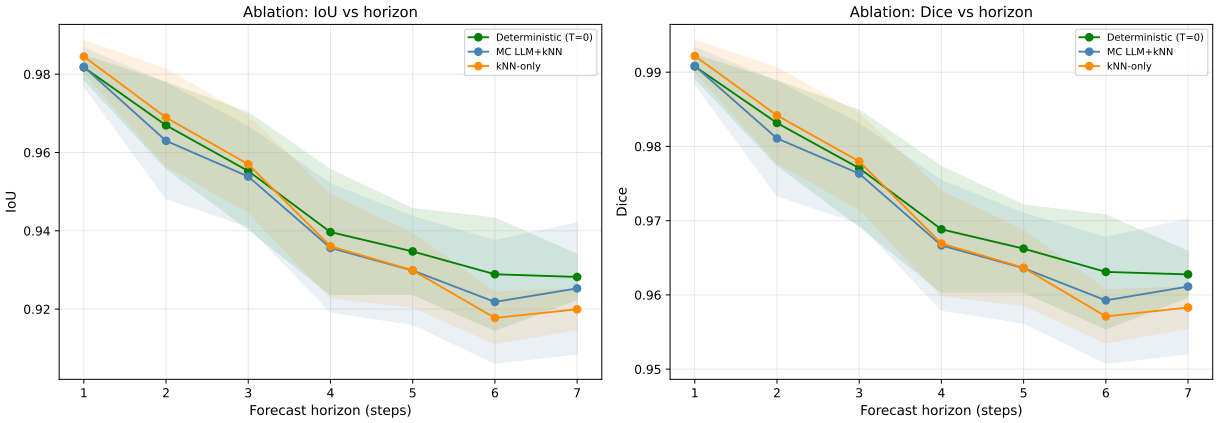

Figure 11: Ablation comparison: IoU (left) and Dice (right) vs. horizon for the three pipeline variants. All variants show similar performance trajectories with overlapping confidence bands.

16.2 Added Value of LLM Residual (ΔIoU)

Figure 12 shows the IoU difference relative to the kNN-only baseline. A clear horizon-dependent pattern emerges: the LLM residual is slightly detrimental at short horizons ($h \leq 3$, where the kNN prior is already accurate and the LLM adds noise), but **provides increasing benefit at longer horizons** ($h \geq 4$), with the deterministic variant reaching +1.1% IoU improvement at horizon 6.

This suggests that the LLM residual correction is most valuable for correcting accumulated drift in the autoregressive rollout. At short horizons, the kNN prior already captures the dominant dynamics and the LLM perturbation is counterproductive. At longer horizons, however, the LLM's ability to model non-linear trends in the latent trajectory provides a genuine correction. The deterministic variant consistently outperforms MC LLM+kNN, suggesting that the stochastic sampling noise ($T = 0.6$) dilutes the LLM's signal. Future work could explore adaptive temperature schedules or larger ensembles.

17 Qualitative Results

Figure 13 shows side-by-side comparisons of ground-truth final masks, mean MC predictions, XOR difference maps, and pixel-wise probability maps for all 8 test experiments. The predictions capture the overall material geometry well but tend to produce smoother, more circular boundaries than the true irregular corrosion fronts — a consequence of the SDF smoothing and monotonic shrinkage

Table 4: Aggregated results across all horizons and test slices (mean ± std).

Variant	IoU	Dice	BF1
MC LLM+kNN	0.9480 ± 0.024	0.9731 ± 0.013	0.668 ± 0.198
Deterministic (T=0)	0.9512 ± 0.022	0.9748 ± 0.012	0.683 ± 0.198
kNN-only	0.9492 ± 0.025	0.9774 ± 0.013	0.680 ± 0.207

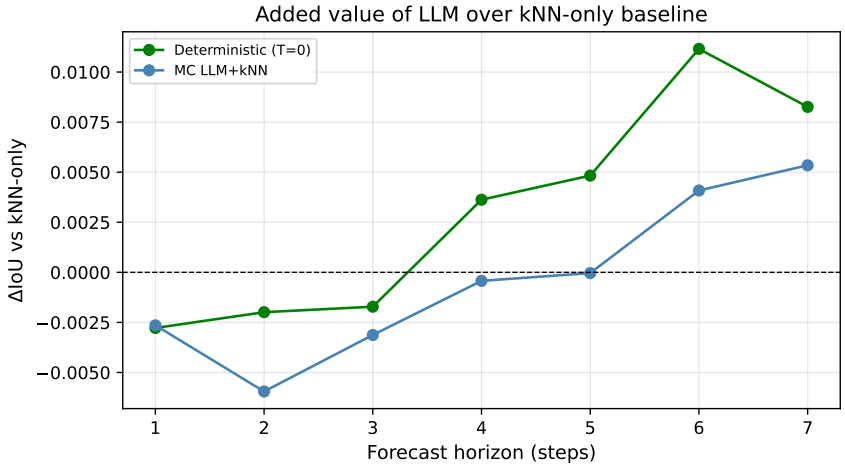

Figure 12: Added IoU value of each variant relative to kNN-only. Values below zero indicate that the variant underperforms the pure kNN prior. The deterministic LLM provides clear benefit at longer horizons ($h \geq 4$), reaching +1.1% IoU at horizon 6.

constraints. The probability maps show high confidence (yellow) in the interior and elevated uncertainty (cyan/purple) at the boundaries, which is physically appropriate since boundary pixels are most affected by corrosion dynamics.

18 Uncertainty Calibration

The MC probability maps were assessed using three calibration metrics and two diagnostic plots (Figure 14).

- **Brier score** = 0.0246 (lower is better; perfect = 0)
- **NLL** = 0.260 (lower is better)
- **ECE** = 0.026 (lower is better; perfect = 0)

The reliability diagram (left panel) shows good calibration at the extremes ($P \approx 0$ and $P \approx 1$) where most pixels reside, with some overconfidence in the 0.3–0.8 range. The risk–coverage curve (right panel) confirms that uncertainty is informative: the 10% of pixels with the lowest uncertainty have only 0.9% error, while the full dataset has 2.6% error. This validates the MC rollout strategy as a meaningful source of epistemic uncertainty.

19 Summary of Key Findings

1. **SDF-PCA is highly efficient**: 99% variance captured at $K = 24$; oracle reconstruction IoU > 0.99 at $K = 40$.

2. **Forecast quality is strong**: IoU ≈ 0.98 at horizon 1, gracefully degrading to ≈ 0.91 at horizon 7.

3. **kNN prior is a strong baseline**: the inverse-distance weighted kNN predictor in normalised PCA space alone achieves IoU = 0.949, providing a competitive baseline.

4. **Deterministic LLM outperforms kNN**: the deterministic LLM residual correction (T=0) achieves the best overall IoU (0.951), with the benefit concentrated at longer horizons ($h \geq 4$, up to +1.1% IoU at $h = 6$). The stochastic MC variant slightly underperforms due to sampling noise at short horizons.

5. **MC uncertainty is well-calibrated**: Brier = 0.025, ECE = 0.026; the risk–coverage curve confirms that uncertainty tracks prediction error.

6. **Boundary accuracy degrades faster than volume**: BF1 drops more rapidly than IoU/Dice, indicating the model preserves overall shape but loses fine boundary detail at longer horizons.

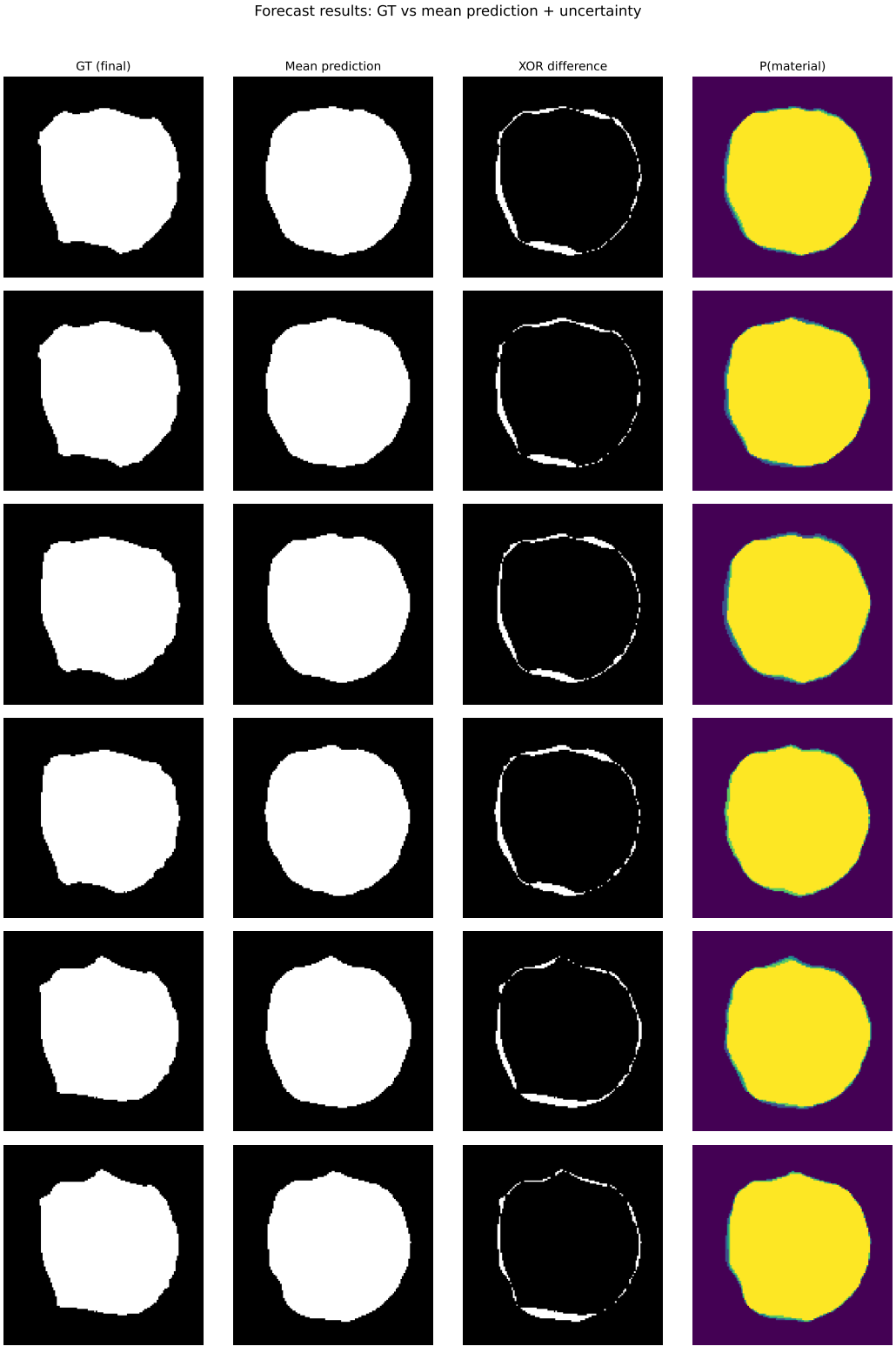

Figure 13: Prediction gallery for all test experiments. Columns: ground-truth final mask, mean MC prediction, XOR difference (white = disagreement), and probability map (yellow = high P(material), purple = low). Rows correspond to different (slice, t_0) combinations.

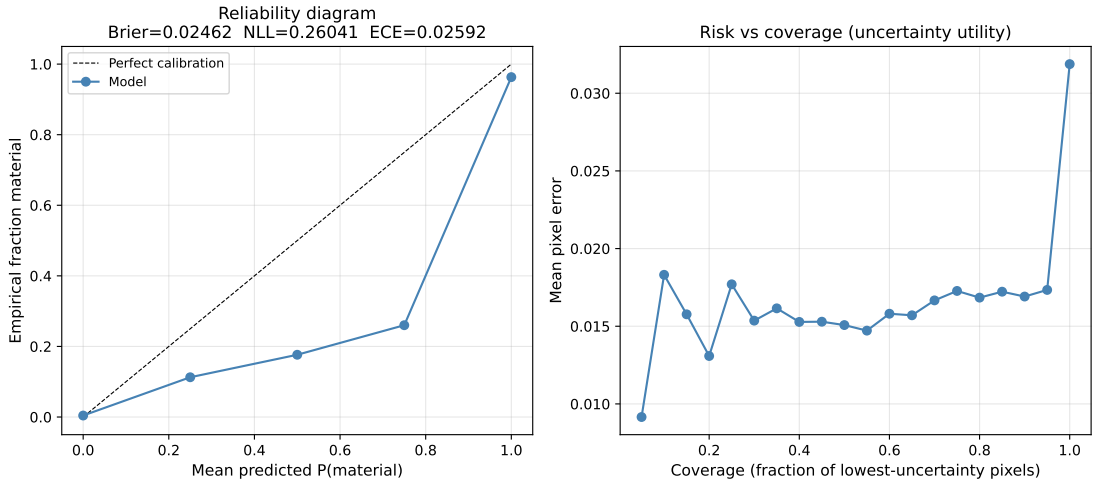

Figure 14: Left: reliability diagram showing calibration of MC probability estimates (ECE = 0.027). Right: risk–coverage curve demonstrating that low-uncertainty pixels have systematically lower prediction error.