補間手法比較レポート（K-Fold / SNR 実験）

作成日: 2025-12-14
対象リポジトリ: `csi-interpolation`

1. レポートの目的

本レポートでは、以下を整理する。

- **行った実験の概要**
- **Day0（事前学習）/ Day1（ファインチューニング）で使用したデータセット**
- **予測モデル（Day0/Day1）と、補間NN（Interpolator）のモデル仕様**
- **補間手法（スプライン / 低次多項式 / NN補間）の計算方法と、試した全バリエーション**
- **どの補間手法が「補間精度が高い」か／「最終モデル精度を改善する」か**
- **結果の理由（考察）**

2. 実験の入力・出力（参照ファイル）

2.1 K-Fold（5-Fold CV）

- 実行スクリプト: `run_kfold_inside_cell.sh`
- 使用設定: `run_config_real_town.yml` （ログ内で読み込みが確認できる）
- ログ:
  - `logs/deg234_run_kfold_inside_cell_20251212_145514.log`
  - `logs/deg456_k369_run_kfold_inside_cell_20251212_201819.log`
- 集計CSV（ログから抽出して生成）:
  - `results/kfold_summary/kfold_nmse_20251212_145514.csv`
  - `results/kfold_summary/kfold_nmse_20251212_201819.csv`
- 図（mean±stdのエラーバー）:
  - `results/kfold_summary/kfold_nmse_errorbars_20251212_145514.png`
  - `results/kfold_summary/kfold_nmse_errorbars_20251212_201819.png`

2.2 SNR実験（入力にAWGNを付与）

- ログ: `logs/run_snr_experiment_20251213_111533.log`
- 結果CSV: `results/snr_experiment/snr_experiment_results.csv`
- 図: `results/snr_experiment/snr_nmse_plot.png`

3. 使用データセット（Day0 / Day1 / 評価用）

3.1 Day0（事前学習）データセット

`run_config_real_town.yml` より:

- `day0_model_dir`: `results/real_town/day0_model`
- `h5_dirs`: `["data/ikebukuro", "data/shibuya", "data/shinjuku"]`

Day0学習は今回のK-Foldログでは **「Skipping Day0 Base Model training」** となっており、既に学習済みのDay0モデル（チェックポイント）を利用している。

3.2 Day1（ファインチューニング用の元データ）

`run_config_real_town.yml` より:

- `day1_data_dir`: `/media/matsuhashi/データ保管/Documents/matsuhashi2/csi-interpolation/data/1_kinshityou`

K-Foldログでは**Day1の軌跡H5が 204 files** として読み込まれている（foldごとに 164 finetune / 40 eval へ分割）。

3.3 評価データセット（K-Foldのホールドアウト）

K-Foldパイプラインでは foldごとに `data/evaluation_dataset_fold{k}.h5` を生成し、 `scripts/4_evaluate.py` で

- Day0モデル（base）
- Day1ファインチューン後モデル（finetuned）

を同一の評価データで比較している。

評価用H5は「補間を使わない通常の next-step 予測」形式で作られる
（ `scripts/2_1_generate_finetune_data_spline.py` の `create_evaluation_samples` の設計に一致）。

4. 予測に使うモデル（Day0/Day1 predictor）

4.1 モデル種別

設定ファイルで `model_type: "MLP"` 。

実装は `scripts/model.py` の `MLP` 。

- 入力: `x ∈ R^{TI*F}`
- 出力: `y ∈ R^{F}`
- 既定の隠れ層: `[2048, 1024, 512]`
- 活性化: ReLU
- Dropout: 0.1

4.2 学習・ファインチューニングの要点

`scripts/3_finetune_model.py` より:

- **Per-sample 正規化**: 入力の電力から `norm_factor` を作り、 `x / norm_factor` をモデル入力にして予測後に戻す
- **損失**: NMSE（サンプル単位）を `loss_clip_value` でクリップして平均
- **最適化**: Adam, `lr=1e-4` （デフォルト）, `epochs=20` , `batch_size=128`
- **勾配クリッピング**: `gradient_clip_norm=1.0`

※ 評価（ `scripts/4_evaluate.py` ）でも同じ正規化を適用し、クリッピングなしNMSEを算出している。

5. 補間に使うニューラルネットワーク（Interpolator）

K-Foldで `nn_context3` を使う場合、 `scripts/2_2_generate_finetune_data_NN.py` により **補間専用のNN** を使用する。

5.1 Interpolatorの目的

- Day0データで「欠損中心フレーム」を近傍から復元するNNを学習
- Day1データに適用して、補間フレーム `I_center` を生成

5.2 Interpolatorモデル（MLP）

`scripts/2_2_generate_finetune_data_NN.py` の `InterpolatorMLP`：

- 入力: 近傍フレームを結合したベクトル（ `(2*context_size)*F` ）
- 出力: 中心フレーム（ `F` ）
- 構造: Linear → ReLU → Linear → ReLU → Linear（ `2048→1024` を含む）

ログにも `Loaded trained interpolator (MLP)` と出ており、今回の `nn_context3` は **MLPのInterpolator** を使用している。

5.3 context_size の意味（試行した“コンテキスト”）

`create_interpolation_samples` の仕様より:

- `context_size=1`： `[H0, H1(target), H2]` → 入力 `(H0, H2)` で `H1` を推定
- `context_size=2`： `[H0, H1, H2(target), H3, H4]` → 入力 `(H0, H1, H3, H4)` で `H2` を推定
- `context_size=3`： `[H0, H1, H2, H3(target), H4, H5, H6]` → 入力 `(H0, H1, H2, H4, H5, H6)` で `H3` を推定

今回のK-Foldは `kfold_methods: "spline,poly,nn_context3"` のため、NN系は **context=3** のみが評価対象。

6. 補間に使う計算手法（試した全バリエーション）

ここでは「補間で生成するフレーム」を (I_3)、「真値（中心）」を (H_3) とする。

6.1 共通のデータ生成形式（ファインチューニング用H5）

スプライン/多項式のデータ生成スクリプトのコメントより、いずれも **Fine-tune用の入力形式は同じ**。

- Window: `[H0, H1, H2, H3, H4, H5, H6]`
- まず中心 `H3` を補間して `I3` を作る
- Predictorへの入力:
  - `X = [I3, H4, H5]` （3フレーム）
  - `Y = H6` （1フレーム）

※ `nn_context3` も最終的に `X=[I_center, H_{center+1}, H_{center+2}]` , `Y=H_{center+3}` の形式を作る（補間フレームの作り方だけが違う）。

6.2 スプライン補間（Spline）

`scripts/2_1_generate_finetune_data_spline.py` より:

- 近傍: `[-3,-2,-1,+1,+2,+3]` に相当する6フレーム（中心は欠損）
- `CubicSpline(time_knots, y_vals, bc_type="natural")` を時間方向に適用
- `t=0` を評価して `I3 = spline(0)`

6.3 低次多項式補間（Polynomial LS）

`scripts/2_1_generate_finetune_data_poly.py` より:

- 時間方向のみ、Vandermonde行列で **最小二乗**フィット
- (t=0) の推定は切片（intercept）に一致

6.3.1 計算方法の詳細（数式・アルゴリズム）

この実装では、中心フレーム (H_3) を直接使わず、周辺6点から時間方向のみで多項式近似し、 その **t=0 の値**を補間値 (I_3) とする。

- 近傍フレーム（k_true=3 の場合）:
  - 時刻（相対オフセット）: (\mathcal{T}={-3,-2,-1,+1,+2,+3})
  - 観測値: (y(t)) （CSIの各要素。Re/Imを含む各次元を独立に扱う）

多項式次数を (D) とすると、各次元ごとに

[y(t) \approx \sum_{p=0}^{D} a_p t^p]

を最小二乗で推定する。行列表現では、

- 設計行列 (A \in \mathbb{R}^{N \times (D+1)}) （(N=|\mathcal{T}|=6)）:

[A_{i,p} = t_i^p \quad (t_i \in \mathcal{T},\ p=0,1,\dots,D)]

- 観測ベクトル (y \in \mathbb{R}^{N}) （ある1次元成分に対する6点の値）
- 係数ベクトル (a \in \mathbb{R}^{D+1})

として、最小二乗解

[a^* = \arg\min_a |Aa - y|_2^2]

を求める（実装は `np.linalg.lstsq(A, y)`）。そして補間値は

[I_3 = y(0) = \sum_{p=0}^{D} a_p^* 0^p = a_0^*]

となり、**切片 (a_0^*)** がそのまま補間値になる（実装上も `coef[0]` を採用）。

> 重要: この実装は「複素数」を複素として直接扱わず、CSIを `[Re, Im]` の実数配列として保持し、**Re/Imも含めた全成分を"独立な実数次元"として同じ近似を行う。**

6.3.2 poly_deg{D}_k{K} の "k_true" と period の意味

- `k_true` は「中心の左右それぞれ何個の **真値フレーム** を近傍に使うか」
- `period=4` の場合、中心が `3,7,11,...` のように周期的に "欠損（予測点）" とみなされる前提で、 近傍に "予測点" が混ざらないように **±4, ±8, ... をスキップ**して近傍候補を組む。
  - 例: `k_true=3` → ([-3,-2,-1,+1,+2,+3])
  - 例: `k_true=6` → ([-11,-10,-9,-7,-6,-5,-3,-2,-1,+1,+2,+3,+5,+6,+7,+9,+10,+11])

6.3.3 ノイズ付き（SNR実験）での扱い

SNR実験では `*_noisy.py` を使い、**入力側（近傍およびXに入るフレーム）だけ**にAWGNを加える。 一方、ターゲット (Y) と、補間誤差評価の真値 (H_3) はクリーンを維持する。

このとき多項式補間は「ノイズが乗った6点 (y(t))」から最小二乗で係数 (a^*) を推定し、(a_0^*) を採用するため、ノイズは **係数推定（特に切片）** を通じて (I_3) に影響する。

(A) deg2/3/4（k_true=3, period=4）

ログ `deg234_run_kfold_inside_cell_20251212_145514.log` で評価された系列:

- `poly_deg2` （近傍は実質 `[-3,-2,-1,+1,+2,+3]` ）
- `poly_deg3`
- `poly_deg4`

(B) deg4/5/6 と k_true の比較（period=4）

ログ `deg456_k369_run_kfold_inside_cell_20251212_201819.log` で評価された系列:

- `poly_deg4_k3` （基準: 近傍 `[-3,-2,-1,+1,+2,+3]` ）
- `poly_deg5_k3`
- `poly_deg6_k3`
- `poly_deg4_k6` （より遠い “真値” 近傍を追加、period=4なので ±4,±8,... はスキップ）
- `poly_deg4_k9` （さらに遠い近傍を追加）

6.4 NN補間（nn_context3）

`scripts/2_2_generate_finetune_data_NN.py` より:

- `context_size=3` で、近傍 `(H0,H1,H2,H4,H5,H6)` を入力に **中心フレームをMLPで推定**
- 生成した補間フレームを先頭にして `X=[I3, H4, H5]`, `Y=H6` のH5を作る

7. 結果

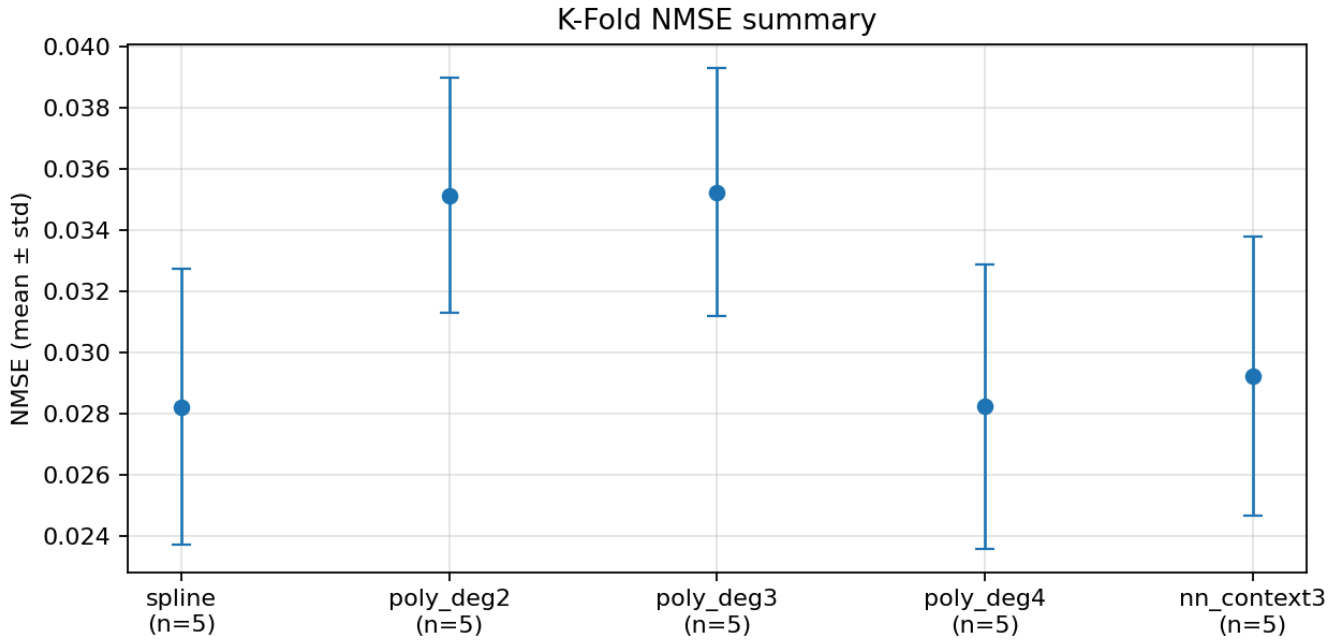
7.1 K-Fold（5-Fold）: 最終モデル NMSE（Fine-tuned Model NMSE）

ここでのNMSEは、 `scripts/4_evaluate.py` が出力する **Fine-tuned Model NMSE** をログから抽出したもの。
（ `scripts/summarize_kfold_results.py` が `Fine-tuned Model NMSE:` 行を抽出）

実験A: deg2/3/4（k_true=3）

対象ログ: `logs/deg234_run_kfold_inside_cell_20251212_145514.log`
集計CSV: `results/kfold_summary/kfold_nmse_20251212_145514.csv`

図（mean ± std）（※ご指定の通り、こちらが実験Aの図）:



method	mean NMSE	std	n
spline	**0.028233**	0.004516	5
poly_deg4	0.028254	0.004649	5
nn_context3	0.029239	0.004561	5
poly_deg2	0.035146	0.003849	5
poly_deg3	0.035249	0.004055	5

結論（実験A）：

- 最良は **spline**（ただし `poly_deg4` と差は極小）
- `deg2/deg3` は明確に悪化（→ 低次すぎる可能性）
- `nn_context3` は spline/deg4 と僅差で次点

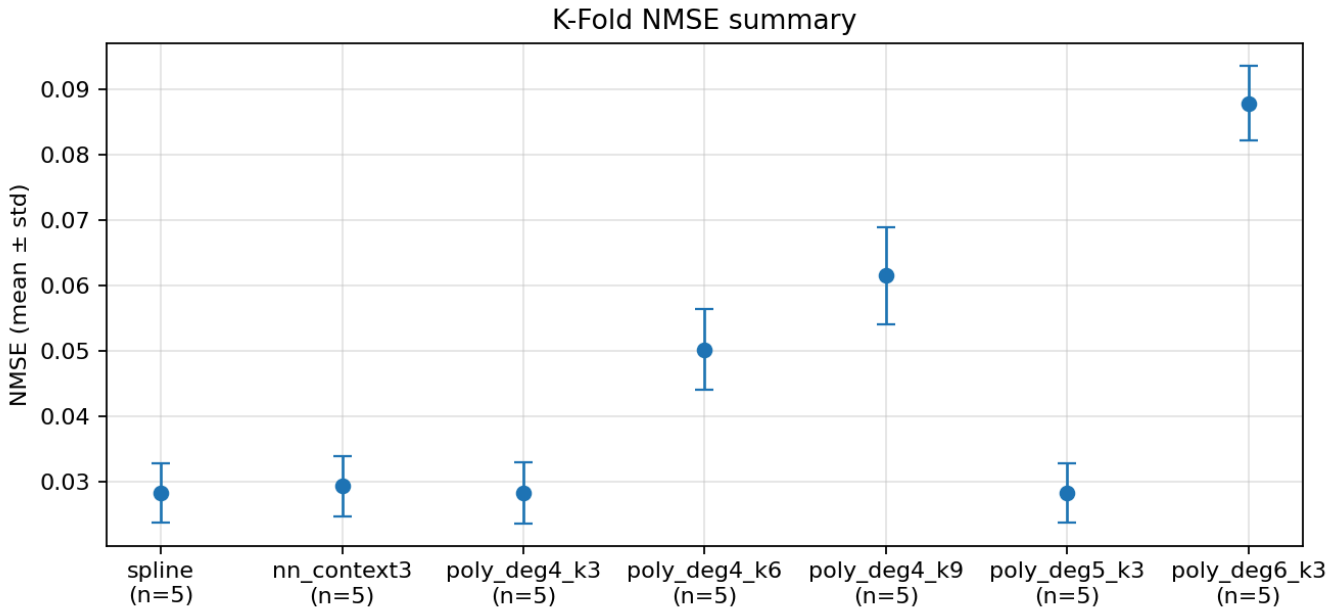
図の読み方（実験A）：

- **spline / poly_deg4 / nn_context3** の誤差棒が大きく重なっており、差は **統計的に見ても僅差**（fold間ばらつきの範囲内）
- **poly_deg2 / poly_deg3** は平均値が上にシフトしており、誤差棒を加味しても **劣化が明確**

実験B: deg4/5/6 と k_true（k=3/6/9）

対象ログ: `logs/deg456_k369_run_kfold_inside_cell_20251212_201819.log`
集計CSV: `results/kfold_summary/kfold_nmse_20251212_201819.csv`

図（mean ± std）：



method	mean NMSE	std	n
spline	**0.028233**	0.004516	5
poly_deg5_k3	0.028247	0.004569	5
poly_deg4_k3	0.028254	0.004649	5
nn_context3	0.029239	0.004561	5
poly_deg4_k6	0.050151	0.006181	5
poly_deg4_k9	0.061435	0.007382	5
poly_deg6_k3	0.087789	0.005713	5

結論（実験B）：

- 最良は **spline**（ `poly_deg5_k3` , `poly_deg4_k3` が僅差で追随）
- `k_true` を大きくして遠方の近傍を増やす（ `k6/k9` ）と **大きく悪化**
- 高次（ `deg6` ）も **大きく悪化**

図の読み方（実験B）：

- `poly_deg4_k3` / `poly_deg5_k3` は **splineとほぼ同等**（誤差棒が強く重なる）
- 一方で `poly_deg4_k6` / `poly_deg4_k9` / `poly_deg6_k3` は **平均値が大幅に悪化**しており、誤差棒の範囲を見ても明確に別グループ

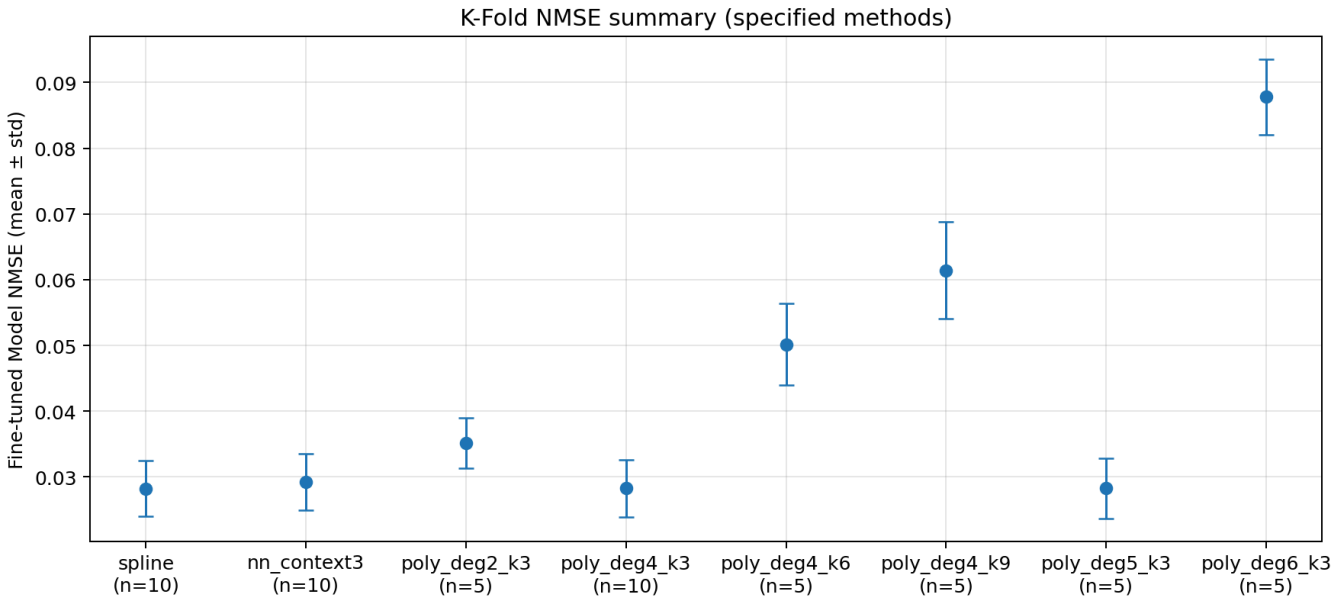
7.1.3 指定手法（質問にある4カテゴリ）を1枚に統合した図と、補間NMSE＋改善率の表

質問で指定された手法カテゴリ:

- (i) **Spline補間（前後3スロット）**： `spline`
- (ii) **4次多項式（前後3/6/9スロット）**： `poly_deg4_k3` , `poly_deg4_k6` , `poly_deg4_k9`
- (iii) **多項式（前後3スロット、2/4/5/6次）**： `poly_deg2_k3` , `poly_deg4_k3` , `poly_deg5_k3` , `poly_deg6_k3`
- (iv) **MLP補間（前後3スロット）**： `nn_context3`

> 注: `poly_deg4_k3` は実験Aログでは `poly_deg4` 表記だが、内容は **k=3** と等価なので本レポートでは `poly_deg4_k3` に正規化して集計している。

K-Fold最終NMSE（Fine-tuned Model NMSE）を1枚に統合（mean ± std）：



表（CSV）：

- 詳細（fold×method）：`results/kfold_summary/kfold_specified_methods_detail.csv`
- 集計（methodのmean±std）：`results/kfold_summary/kfold_specified_methods_summary.csv`

表の列の意味:

- `interp_nmse_mean/std`：**補間そのものの精度**（ログの `Average NMSE (I_n vs h_n)`）
- `finetuned_nmse_mean/std`：**最終評価NMSE**（`4_evaluate.py` の `Fine-tuned Model NMSE`）
- `improvement_pct_mean/std`：**改善率** (100 \times (Base - Fine) / Base)

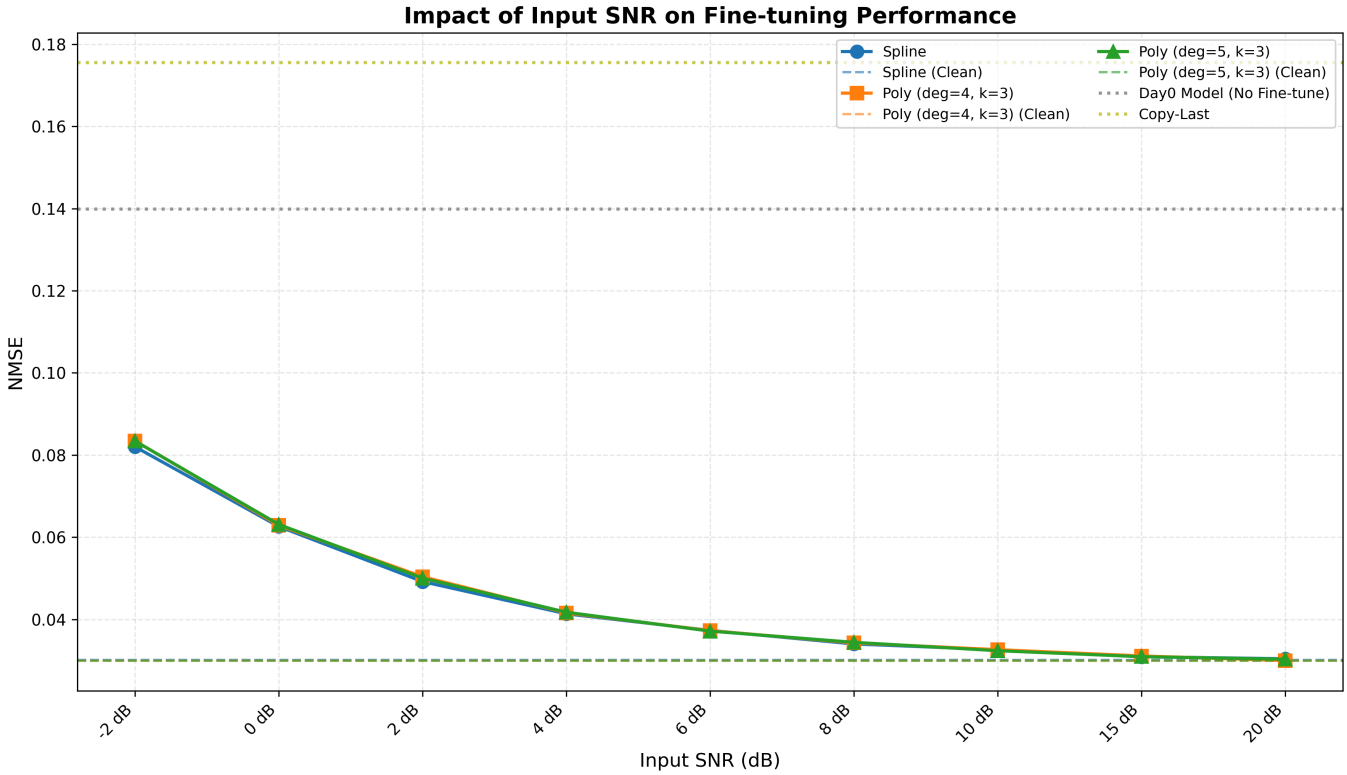
集計結果（`kfold_specified_methods_summary.csv` をMarkdown表に整形）：

method	n	interp NMSE (mean±std)	finetuned NMSE (mean±std)	improvement % (mean±std)
spline	10	0.006111±0.000368	0.028233±0.004257	79.368±2.390
nn_context3	10	0.020046±0.000364	0.029239±0.004300	78.635±2.357
poly_deg2_k3	5	0.051755±0.003383	0.035146±0.003849	74.287±1.759
poly_deg4_k3	10	0.005856±0.000380	0.028254±0.004383	79.353±2.497
poly_deg4_k6	5	0.229095±0.014694	0.050151±0.006181	63.283±3.607
poly_deg4_k9	5	0.258491±0.007546	0.061435±0.007382	54.951±5.172
poly_deg5_k3	5	0.005856±0.000403	0.028247±0.004569	79.358±2.581
poly_deg6_k3	5	0.143227±0.002374	0.087789±0.005713	35.619±3.191

7.2 SNR実験（入力ノイズに対する頑健性）

対象: `results/snr_experiment/snr_experiment_results.csv`（手法: `spline`, `poly_deg4_k3`, `poly_deg5_k3`）

図（SNR vs NMSE）：



ノイズあり（SNR=-2〜20 dB）の総合指標（小さいほど良い）:

- mean NMSE（全SNR平均）: **splineが最良（0.044482）**
- lowSNR平均（-2/0/2 dB）: **splineが最良（0.064592）**
- worst（最悪ケース）: **splineが最良（-2 dBで 0.081998）**

補足: clean（ノイズなし）では `poly_deg5_k3` が僅かに良いが、ノイズが入るとsplineの劣化が最小で総合優位。

図の読み方（SNR）:

- 左（低SNR）ほどノイズが強くNMSEが悪化し、右（高SNR）ほどcleanの破線へ近づく
- 3手法の差は大きくないが、**低SNR域でsplineが最も下（NMSEが小）**に位置しやすく、総合指標でも最良

8. どの補間手法が「補間精度が高い」/「モデルを改善した」か

8.1 「モデルを改善した」＝K-Fold最終NMSEの観点

- **最も安定して良い**: Spline
- **Splineに最も近い**: `poly_deg4_k3` / `poly_deg5_k3`
- **次点**: `nn_context3`（僅差）
- **悪い**: `poly_deg2`, `poly_deg3`, `poly_deg4_k6`, `poly_deg4_k9`, `poly_deg6_k3`

8.2 「ノイズに強い」＝SNR実験の観点

- **最もノイズ耐性が高い**: Spline

9. 理由（考察）

9.1 Spline が強い理由（仮説）

- **局所性 + 滑らかさ（自然スプライン）**により、時間変化が滑らかなCSIに対して過度に振れず安定
- ノイズが入っても「多項式の高次項の暴れ」や「Vandermondeの条件悪化」が起こりにくい

9.1.1 「スプラインは全点を通るのに、なぜノイズで悪化しないの？」への説明

疑問になりやすいポイントは、「補間スプラインは観測点（近傍6点）を **必ず通る** → ノイズがあるとむしろ不利では？」という点。 ここでは誤解が起きやすい点を整理する。

- **“全点を通る”のは「ノイズ付き観測点」を通る**という意味であり、真の（ノイズなしの）軌跡を通るわけではない。
  つまり「点を通ること」自体が、欠損点（中心）の推定に対して常に有利とは限らない。

- それでも今回 spline が優位になったのは、少なくとも以下が効いている可能性が高い。

  - **局所的で形が安定**: 自然3次スプラインは区分的3次で、境界条件 `bc_type="natural"` （端で2階微分0）により 端の暴れが抑制されやすい。中心（t=0）は左右対称の配置で、推定が比較的安定になりやすい。
  - **数値的安定性**: `scipy.interpolate.CubicSpline` はスプライン係数計算が安定なアルゴリズムで実装されている一方、 多項式LSは monomial 基底のVandermonde行列を使うため、次数や点配置によっては **係数推定がノイズに敏感**になり得る。
  - **目的は「中心値を当てること」**: 近傍点を厳密に通ることよりも、t=0 の推定誤差（NMSE）が小さいことが重要。 スプラインは“近傍点への整合”を保ちつつ、中心の推定で過度に振れない形になりやすい。

補足（線形代数の見方）: どちらの手法も、最終的に $\hat{I}_3$ は「近傍値 y の線形結合」 $\hat{I}_3 = w^\top y$ とみなせる（固定の時刻・固定の手法なら重み w は決まる）。 ノイズ $y = y_{\text{true}} + \varepsilon$ に対しては、分散が概ね $\mathrm{Var}(\hat{I}_3) \propto \|w\|_2^2$ で増えるため、 **重みのノルムが大きい（外挿っぽくなる／係数が暴れる）ほどノイズで悪化しやすい**。 今回の設定では、スプラインの重みが相対的に安定で、多項式LS（特に高次・広い近傍）より悪化が小さかった、と解釈できる。

9.2 多項式補間が deg4/5（k3）で良く、deg2/3 で悪い理由（仮説）

- `deg2/deg3` は **表現力が不足**し、曲率・非線形な時間変化を近似しきれない（→補間誤差↑ → fine-tune教師信号の質↓）
- `deg4/deg5` は局所窓（±3）に対して **適度な自由度**で近似でき、Splineに近い性能が出る

9.3 poly_deg6_k3 が悪化する理由（仮説）

- 近傍点が6点しかない状況で高次（6次）を最小二乗すると **過学習/不安定（条件数悪化）**しやすい
- 補間誤差が増え、fine-tuningデータの先頭フレーム `I3` が劣化 → 予測モデルが学習しづらい

9.4 k_true を増やす（k6/k9）と悪化する理由（仮説）

- 近傍が遠方に広がることで **局所近似としての妥当性が崩れる**（時間変化の非定常性に引っ張られる）
- `period=4` により近傍オフセットが不均一（±4,±8…をスキップ）になり、設計行列の条件が悪くなり得る
- 結果として補間誤差が増え、fine-tuningの入力品質が低下

9.5 nn_context3 が spline/deg4 に僅差で負ける理由（仮説）

- Interpolator は **Day0分布で学習**され、Day1へ適用する際にドメインシフトがある可能性
- Interpolatorは「中心復元」最適化であり、「下流の予測モデルの最終NMSE」を直接最適化していない
- MLPの容量/正則化/学習設定次第で spline と同等に届く余地がある（ただし現状は僅差で負け）

10. 結論（短く）

- **K-Fold最終NMSE**の観点では、**Spline が最良**。次点が **poly_deg4_k3 / poly_deg5_k3**、その次が **nn_context3**。
- **ノイズ耐性（SNR実験）**でも **Spline が最良**。
- 多項式は「低次すぎる」「高次すぎる」「近傍が遠すぎる（kが大きい）」で悪化しやすく、Spline/（deg4〜5,k3）程度が妥当。