

Build Models and Analyse FD003

```
In [1]: 1 import os
        2 os.environ["KERAS_BACKEND"] = "torch"
        3 os.environ["PYTORCH_ENABLE_MPS_FALLBACK"] = "1"
        4 os.environ["PYTORCH_MPS_HIGH_WATERMARK_RATIO"] = "0.0" # optional: reduce MPS memory pressure
        5
        6 from keras.models import load_model
        7 import numpy as np
        8 import pandas as pd
        9 import matplotlib.pyplot as plt
       10 import seaborn as sns
       11 from pathlib import Path
       12 import joblib
       13 from keras.models import load_model as keras_load_model
       14 import random
       15 from typing import Iterable, Dict, List, Optional, Tuple
```

```
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
WARNING:root:Limited tf.summary API due to missing TensorBoard installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
```


In [2]:

```
1 # --- functions---
2
3 def load_any_model(path):
4     """
5     Load a model given its file path, handling both Keras (.keras/.h5) and joblib (.joblib/.pkl).
6     """
7     path = Path(path)
8     suf = path.suffix.lower()
9     if suf in {".keras", ".h5", ".hdf5"}:
10         return keras_load_model(path)
11     if suf in {".joblib", ".pkl"}:
12         return joblib.load(path)
13     raise ValueError(f"Unrecognized model suffix for {path.name}")
14
15 def load_models(model_paths: dict):
16     """
17     model_paths: dict like {"Base": "...joblib", "LSTM": "...keras", "CNN": "...keras", "CNN-LSTM": "...keras"}
18     returns: dict {name: model_object}
19     """
20     models = {}
21     for name, path in model_paths.items():
22         models[name] = load_any_model(path)
23     return models
24
25 def build_model_paths(dataset=None, seq_len=None, strategy="last", art_dir=None,
26                       include=("Base", "LSTM", "CNN", "CNN-LSTM")):
27     """
28     Build default file paths for your saved models in <ART_DIR>/models/...
29     Base:      base_linear_{dataset.lower()}_seq{seq_len}_{strategy}.joblib
30     LSTM:      lstm_{dataset.lower()}_seq{seq_len}.keras
31     CNN:       cnn_{dataset.lower()}_seq{seq_len}.keras
32     CNN-LSTM:  cnn_lstm_{dataset.lower()}_seq{seq_len}.keras
33     """
34     if dataset is None:
35         dataset = globals().get("DATASET", "FD001")
36     if seq_len is None:
37         seq_len = globals().get("SEQ_LEN", 30)
38     if art_dir is None:
39         art_dir = globals().get("ART_DIR", Path.cwd() / f"{dataset} data & artefacts")
40
41     models_dir = Path(art_dir) / "models"
```

```

42     ds = dataset.lower()
43
44     paths = {}
45     if "Base" in include:
46         paths["Base"] = str(models_dir / f"base_linear_{ds}_seq{seq_len}_{strategy}.joblib")
47     if "LSTM" in include:
48         paths["LSTM"] = str(models_dir / f"lstm_{ds}_seq{seq_len}.keras")
49     if "CNN" in include:
50         paths["CNN"] = str(models_dir / f"cnn_{ds}_seq{seq_len}.keras")
51     if "CNN-LSTM" in include:
52         paths["CNN-LSTM"] = str(models_dir / f"cnn_lstm_{ds}_seq{seq_len}.keras")
53     return paths
54
55
56 def plot_true_vs_pred(y_true, y_pred, model_name="Model", savepath=None):
57     """
58     Scatter plot: True RUL vs Predicted RUL for a single model.
59
60     Args:
61         y_true (array-like): Ground truth RUL values.
62         y_pred (array-like): Predicted RUL values.
63         model_name (str): Name of the model for labeling.
64         savepath (str or None): If given, save the figure to this path.
65     """
66     plt.figure(figsize=(6, 6))
67     plt.scatter(y_true, y_pred, alpha=0.6, edgecolor="k")
68     max_val = max(y_true.max(), y_pred.max())
69     plt.plot([0, max_val], [0, max_val], "r--", lw=2, label="Ideal")
70     plt.xlabel("True RUL")
71     plt.ylabel("Predicted RUL")
72     plt.title(f"True vs Predicted RUL ({model_name})")
73     plt.legend()
74     plt.grid(True)
75     if savepath:
76         plt.savefig(savepath, dpi=300, bbox_inches="tight")
77     plt.show()
78
79
80
81 def plot_residuals(y_true, y_pred, model_name="Model", kind="box", savepath=None):
82     """
83     Plot residuals (Predicted - True) for a single model.

```

```

84
85     Args:
86         y_true (array-like): Ground truth RUL values.
87         y_pred (array-like): Predicted RUL values.
88         model_name (str): Label for the model.
89         kind (str): "box" or "violin" for plot type.
90         savepath (str or None): If given, save the figure.
91     """
92     errors = y_pred - y_true
93     plt.figure(figsize=(6, 4))
94
95     if kind == "violin":
96         sns.violinplot(y=errors)
97     else:
98         sns.boxplot(y=errors)
99
100    plt.axhline(0, color="r", linestyle="--", lw=2, label="Zero Error")
101    plt.ylabel("Residual (Predicted - True)")
102    plt.title(f"Residual Distribution ({model_name})")
103    plt.legend()
104    plt.grid(True, axis="y")
105    if savepath:
106        plt.savefig(savepath, dpi=300, bbox_inches="tight")
107    plt.show()
108
109 def plot_per_engineBars(y_true, y_pred, unit_ids, model_name="Model", n_samples=30, savepath=None, seed=42):
110     """
111     Bar plot comparing Actual vs Predicted RUL for a random sample of engines.
112
113     Args:
114         y_true (array-like): Ground truth RUL values (aligned with unit_ids).
115         y_pred (array-like): Predicted RUL values (aligned with unit_ids).
116         unit_ids (array-like): Engine/unit identifiers for each sample.
117         model_name (str): Name of the model for labeling.
118         n_samples (int): Number of engines to randomly sample.
119         savepath (str or None): If given, save the figure.
120         seed (int): Random seed for reproducibility.
121     """
122     random.seed(seed)
123     unique_ids = np.unique(unit_ids)
124     chosen_ids = random.sample(list(unique_ids), min(n_samples, len(unique_ids)))
125

```

```

126     # Collect true & pred for chosen engines
127     true_sample, pred_sample, labels = [], [], []
128     for uid in chosen_ids:
129         mask = unit_ids == uid
130         # Last entry corresponds to the test RUL Label
131         true_sample.append(y_true[mask][-1])
132         pred_sample.append(y_pred[mask][-1])
133         labels.append(str(uid))
134
135     x = np.arange(len(chosen_ids))
136     width = 0.35
137
138     plt.figure(figsize=(12, 6))
139     plt.bar(x - width/2, true_sample, width, label="Actual")
140     plt.bar(x + width/2, pred_sample, width, label="Predicted")
141     plt.xticks(x, labels, rotation=45)
142     plt.xlabel("Engine ID (sampled)")
143     plt.ylabel("RUL")
144     plt.title(f"Actual vs Predicted RUL (Sampled Engines) - {model_name}")
145     plt.legend()
146     plt.tight_layout()
147     if savepath:
148         plt.savefig(savepath, dpi=300, bbox_inches="tight")
149     plt.show()
150 def plot_metric_comparison(metrics_list,
151                             dataset_name: str = "FD001",
152                             savepath: Optional[str] = None):
153     """
154     Grouped bar chart of RMSE/MAE across models.
155     Accepts either:
156     - list of dicts: [{"model": "LSTM", "RMSE": 15.2, "MAE": 11.3}, ...]
157     - DataFrame with columns: model/Model, RMSE, MAE
158     """
159     # Build DataFrame
160     df = metrics_list.copy() if isinstance(metrics_list, pd.DataFrame) else pd.DataFrame(metrics_list)
161     if df.empty:
162         print("No metrics to plot.")
163         return
164
165     # Normalise column names
166     if "model" not in df.columns and "Model" in df.columns:
167         df = df.rename(columns={"Model": "model"})

```

```

168
169     # Validate required columns
170     required = {"model", "RMSE", "MAE"}
171     missing = required - set(df.columns)
172     if missing:
173         raise ValueError(f"Missing columns for plotting: {missing}")
174
175     # Keep only what we need, coerce to numeric
176     df = df[["model", "RMSE", "MAE"]].copy()
177     df[["RMSE", "MAE"]] = df[["RMSE", "MAE"]].astype(float)
178     df = df.set_index("model")
179
180     # Plot
181     ax = df.plot(kind="bar", figsize=(10, 6))
182     ax.set_title(f"RMSE / MAE Comparison - {dataset_name}")
183     ax.set_ylabel("Error")
184     ax.set_xlabel("")
185     ax.grid(True, axis="y", alpha=0.3)
186     plt.xticks(rotation=0)
187     plt.tight_layout()
188     plt.show()

```

Project Module and base set up

In [3]:

```
1  # Project modules
2  import data_loader as dl
3  import pre_processing as pp
4  import evaluator as ev
5  import base_model as base
6  import lstm_model as lstm
7  import cnn_model as cnn
8  import cnn_lstm_model as cnnlstm
9  import plots
10
11  # ---- Paths ----
12  ROOT = Path.cwd()
13  CMAPS = ROOT / "CMaps" # keep correct folder case
14  # ==== Minimal config you tweak next time ====
15  DATASET = "FD003" # <- change this to FD002/FD003/FD004 Later
16  SEQ_LEN = 30 # sliding window
17  MAX_RUL = 130 # RUL clipping
18  VAL_SPLIT = 0.30 # val split by unit
19
20  # Files derived from DATASET (so you edit one line only)
21  TRAIN_PATH = CMAPS / f"train_{DATASET}.txt"
22  TEST_PATH = CMAPS / f"test_{DATASET}.txt"
23  RUL_PATH = CMAPS / f"RUL_{DATASET}.txt"
24
25  # Artifacts folder for this dataset
26  ART_DIR = ROOT / f"{DATASET} data & artefacts"
27  ART_DIR.mkdir(exist_ok=True)
28
29  print(f"backend: torch | dataset: {DATASET}")
30  print("Train:", TRAIN_PATH.name, "| Test:", TEST_PATH.name, "| RUL:", RUL_PATH.name)
```

backend: torch | dataset: FD003

Train: train_FD003.txt | Test: test_FD003.txt | RUL: RUL_FD003.txt

Load & Preprocessing Data

In [4]:

```
1 # --- Load FD001 ---
2 train_df = dl.load_raw_data(CMAPS / f"train_{DATASET}.txt")
3 test_df, rul_df = dl.load_test_data(
4     CMAPS / f"test_{DATASET}.txt",
5     CMAPS / f"RUL_{DATASET}.txt"
6 )
7
8 unit_ids = test_df["unit_number"].values
9
10 print("Loaded.")
11 print("  train_df:", train_df.shape, "  test_df:", test_df.shape, "  rul_df:", rul_df.shape)
12 assert train_df.shape[1] == 26 and test_df.shape[1] == 26
13
14 dl.inspect_data(train_df)
15 pp.summarise_engine_lifespans(train_df, dataset_name=DATASET)
16
17 # -----
18 # 1) TRAIN: make targets first (no Leakage)
19 # -----
20 train_rul = pp.calculate_rul(train_df, max_rul=MAX_RUL)
21
22 # -----
23 # 2) Split by unit BEFORE deciding features/scaling
24 # -----
25 train_split, val_split = pp.split_by_unit(train_rul, test_size=0.2, random_state=42)
26
27 # -----
28 # 3) Decide flat sensors using TRAIN ONLY, then drop same cols from val/test
29 # -----
30 before_cols = list(train_split.columns)
31 train_split_clean = pp.drop_flat_sensors(train_split.copy())
32 after_cols = list(train_split_clean.columns)
33 dropped_cols = [c for c in before_cols if c not in after_cols]
34
35 val_split_clean = val_split.drop(columns=[c for c in dropped_cols if c in val_split.columns]).reset_index(drop=True)
36 test_df_clean = test_df.drop(columns=[c for c in dropped_cols if c in test_df.columns]).reset_index(drop=True)
37
38 # -----
39 # 4) TEST: build true RUL from RUL file, then clip like train
40 # -----
41 last_cycles = test_df_clean.groupby("unit_number")["time_in_cycles"].max()
```

```

42 rul_map = dict(zip(sorted(test_df_clean["unit_number"].unique()), rul_df["RUL"].values))
43 test_df_clean = test_df_clean.copy()
44 test_df_clean["RUL"] = test_df_clean.apply(
45     lambda r: (last_cycles.loc[r["unit_number"]] - r["time_in_cycles"]) + rul_map[r["unit_number"]],
46     axis=1
47 )
48 test_df_clean["RUL"] = np.minimum(test_df_clean["RUL"], MAX_RUL)
49
50 # -----
51 # 5) Scale sensors (per-condition for FD002/FD004, global for FD001/FD003)
52 # -----
53 from sklearn.preprocessing import StandardScaler
54
55 sensor_cols = [c for c in train_split_clean.columns if c.startswith("sensor_measurement")]
56 op_cols = [c for c in train_split_clean.columns
57            if c.startswith("operational_setting") or c.startswith("op_setting")]
58
59 def _cond_keys(df, op_cols, ndigits=3):
60     # Round op settings to make discrete condition keys robust to float jitter
61     return list(map(tuple, df[op_cols].round(ndigits).to_numpy()))
62
63 def fit_per_condition_scalers(df, sensor_cols, op_cols, ndigits=3):
64     df = df.copy()
65     df["_cond_key"] = _cond_keys(df, op_cols, ndigits)
66     scalers = {}
67     for k, g in df.groupby("_cond_key", sort=False):
68         scalers[k] = StandardScaler().fit(g[sensor_cols])
69     return scalers
70
71 def transform_with_condition_scalers(df, sensor_cols, op_cols, scalers, fallback_scaler, ndigits=3):
72     out = df.copy()
73     out["_cond_key"] = _cond_keys(out, op_cols, ndigits)
74     unseen_rows = 0
75     for k, idx in out.groupby("_cond_key").groups.items():
76         sc = scalers.get(k, fallback_scaler)
77         if k not in scalers:
78             unseen_rows += len(idx)
79         out.loc[idx, sensor_cols] = sc.transform(out.loc[idx, sensor_cols])
80     if unseen_rows:
81         print(f"[WARN] {unseen_rows} rows used fallback scaler due to unseen condition(s).")
82     return out.drop(columns=["_cond_key"])
83

```

```

84 multi_condition = DATASET in {"FD002", "FD004"}
85
86 if multi_condition:
87     # Fit: one scaler per operating condition on TRAIN ONLY
88     cond_scalers = fit_per_condition_scalers(train_split_clean, sensor_cols, op_cols, ndigits=3)
89     global_scaler = StandardScaler().fit(train_split_clean[sensor_cols]) # safe fallback
90     print(f"Fitted {len(cond_scalers)} condition-specific scalers on training set.")
91
92     # Apply the same condition scalers to train/val/test
93     train_scaled = transform_with_condition_scalers(train_split_clean, sensor_cols, op_cols, cond_scalers, global_sc
94     val_scaled = transform_with_condition_scalers(val_split_clean, sensor_cols, op_cols, cond_scalers, global_sc
95     test_scaled = transform_with_condition_scalers(test_df_clean, sensor_cols, op_cols, cond_scalers, global_sc
96 else:
97     # Single-condition datasets (FD001, FD003): one global scaler
98     scaler = StandardScaler().fit(train_split_clean[sensor_cols])
99     train_scaled = train_split_clean.copy()
100    val_scaled = val_split_clean.copy()
101    test_scaled = test_df_clean.copy()
102    train_scaled[sensor_cols] = scaler.transform(train_scaled[sensor_cols])
103    val_scaled[sensor_cols] = scaler.transform(val_scaled[sensor_cols])
104    test_scaled[sensor_cols] = scaler.transform(test_scaled[sensor_cols])
105
106    # -----
107    # 6) Windowing with your helper
108    # -----
109    X_train, y_train = pp.generate_sliding_windows(train_scaled, seq_len=SEQ_LEN)
110    X_val, y_val = pp.generate_sliding_windows(val_scaled, seq_len=SEQ_LEN)
111    X_test, y_test = pp.generate_sliding_windows(test_scaled, seq_len=SEQ_LEN)
112
113    print("After preprocessing (FD001, no new pp funcs):")
114    print(" Train engines :", train_scaled['unit_number'].nunique())
115    print(" Val engines :", val_scaled['unit_number'].nunique())
116    print(" X_train shape :", X_train.shape, " y_train:", y_train.shape)
117    print(" X_val shape :", X_val.shape, " y_val :", y_val.shape)
118    print(" X_test shape :", X_test.shape, " y_test :", y_test.shape)
119    print(" Dropped sensors:", dropped_cols)
120
121    # --- Save to use in model ---
122    out_npz = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
123    pp.save_preprocessed_data(X_train, y_train, X_val, y_val, X_test, y_test, filename=str(out_npz))
124

```

Loaded.

train_df: (24720, 26) test_df: (16596, 26) rul_df: (100, 1)
Shape: (24720, 26)

Unique engines: 100

Missing values:
0

Max cycles per engine:

count 100.00000
mean 247.20000
std 86.48384
min 145.00000
25% 189.75000
50% 220.50000
75% 279.75000
max 525.00000

Name: time_in_cycles, dtype: float64

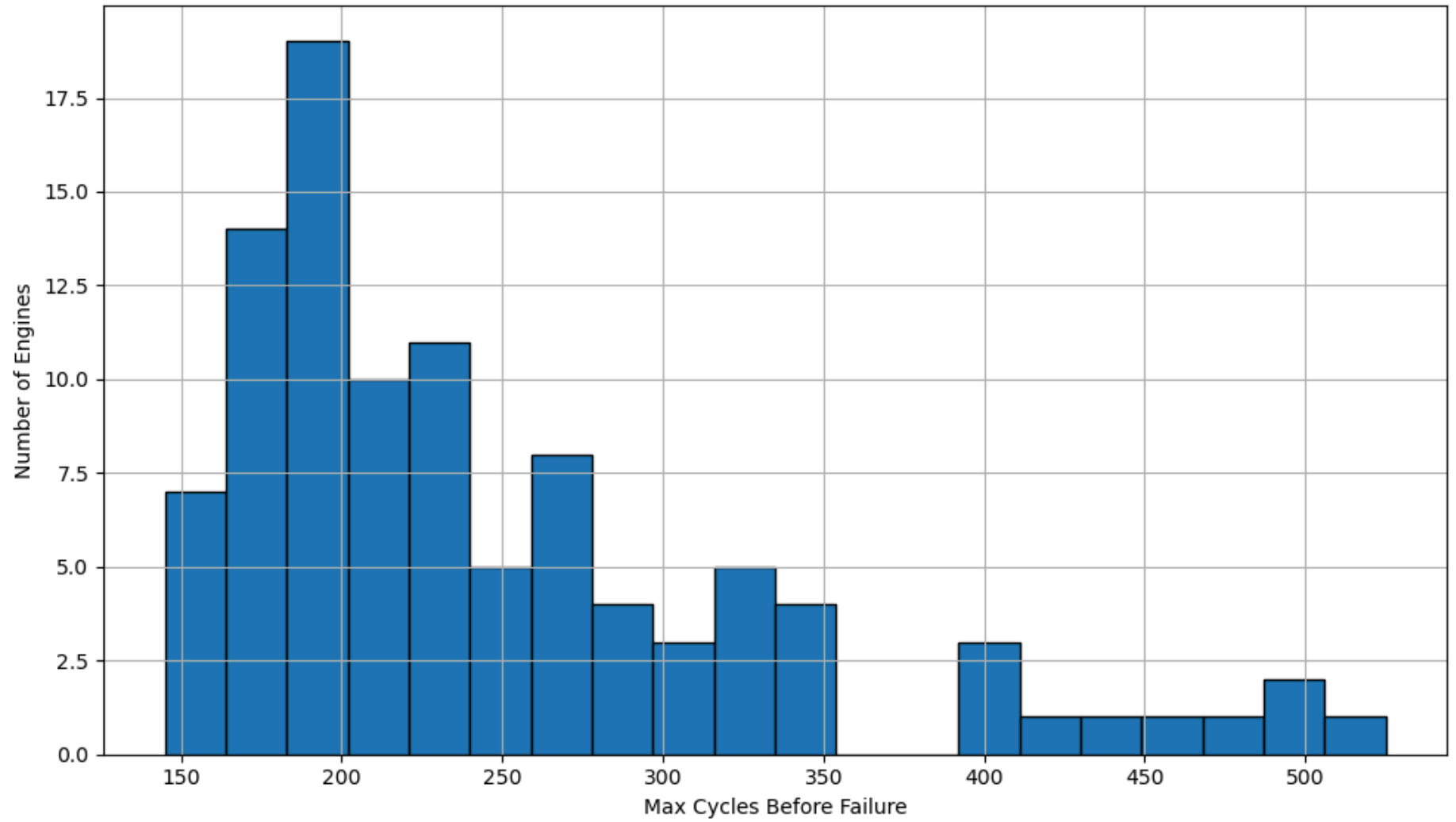
First 5 rows:

	unit_number	time_in_cycles	op_setting_1	op_setting_2	op_setting_3	sensor_measurement_1	sensor_measurement_2	sensor_measurement_3	se
0	1	1	-0.0005	0.0004	100.0	518.67	642.36	1583.23	
1	1	2	0.0008	-0.0003	100.0	518.67	642.50	1584.69	
2	1	3	-0.0014	-0.0002	100.0	518.67	642.18	1582.35	
3	1	4	-0.0020	0.0001	100.0	518.67	642.92	1585.61	
4	1	5	0.0016	0.0000	100.0	518.67	641.68	1588.63	

5 rows × 26 columns



Distribution of Engine Lifespans (FD003)



Dataset: FD003
Mean cycles to failure: 247.20
Standard deviation: 86.48
Minimum: 145
Maximum: 525
After preprocessing (FD001, no new pp funcs):
 Train engines : 80
 Val engines : 20
 X_train shape : (17692, 30, 16) y_train: (17692,)
 X_val shape : (4128, 30, 16) y_val : (4128,)
 X_test shape : (13696, 30, 16) y_test : (13696,)
 Dropped sensors: ['sensor_measurement_1', 'sensor_measurement_5', 'sensor_measurement_16', 'sensor_measurement_18',
 'sensor_measurement_19']
Data saved to C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD003 data & artefacts\fd003_seq30.npz

=====Train Models=====

Base Model

```
In [5]: 1 from base_model import train_linear_model
2
3 # 1) Load cached windows
4 npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
5 X_train, y_train, X_val, y_val, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
6
7 # 2) Convert 3D windows -> 2D feature vectors for baseline
8 # (stick to your existing helper; default strategy='last')
9 X_train_feat = pp.make_feature_vectors_from_windows(X_train, strategy='last')
10 X_val_feat = pp.make_feature_vectors_from_windows(X_val, strategy='last')
11
12 print("Feature shapes (base model):")
13 print(" X_train:", X_train_feat.shape, " y_train:", y_train.shape)
14 print(" X_val  :", X_val_feat.shape, " y_val  :", y_val.shape)
15
16 # 3) Train simple Linear Regression
17 base_model = train_linear_model(X_train_feat, y_train)
18
19 # 4) Save the trained model
20 models_dir = ART_DIR / "models"
21 models_dir.mkdir(parents=True, exist_ok=True)
22
23 model_path = models_dir / f"base_linear_{DATASET.lower()}_seq{SEQ_LEN}_last.joblib"
24 joblib.dump(base_model, model_path)
25
26 print("Saved base model to:", model_path)
```

Feature shapes (base model):

X_train: (17692, 16) y_train: (17692,)

X_val : (4128, 16) y_val : (4128,)

Saved base model to: C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD003 data & artefacts\models\base_linear_fd003_seq30_last.joblib

CNN Model

In [6]:

```
1  # --- Train / Save: CNN (FD001) ---
2
3  from cnn_model import build_cnn_model, train_cnn_model
4  import json
5
6  # 1) Load cached 3D windows
7  npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
8  X_train, y_train, X_val, y_val, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
9
10 # 2) Build CNN
11 input_shape = (SEQ_LEN, X_train.shape[2])
12 cnn = build_cnn_model(input_shape)
13
14 # 3) Train (only pass what your module expects!)
15 cnn, history = train_cnn_model(cnn, X_train, y_train, X_val, y_val, epochs=30, batch_size=128)
16
17 # 4) Save model + meta
18 models_dir = ART_DIR / "models"
19 models_dir.mkdir(parents=True, exist_ok=True)
20 model_path = models_dir / f"cnn_{DATASET.lower()}_seq{SEQ_LEN}.keras"
21 cnn.save(model_path)
22
23 meta = {
24     "model_type": "cnn",
25     "dataset": DATASET,
26     "seq_len": int(SEQ_LEN),
27     "features": int(X_train.shape[2]),
28     "epochs": 30,
29     "batch_size": 128
30 }
31 with open(models_dir / f"cnn_{DATASET.lower()}_seq{SEQ_LEN}.meta.json", "w") as f:
32     json.dump(meta, f, indent=2)
33
34 print("Saved CNN model to:", model_path)
```

Epoch 1/30
139/139 [=====] - 2s 9ms/step - loss: 2286.2473 - val_loss: 518.1725
Epoch 2/30
139/139 [=====] - 1s 8ms/step - loss: 587.1365 - val_loss: 420.0499
Epoch 3/30
139/139 [=====] - 1s 8ms/step - loss: 529.0831 - val_loss: 416.2576
Epoch 4/30
139/139 [=====] - 1s 8ms/step - loss: 505.2010 - val_loss: 418.7025
Epoch 5/30
139/139 [=====] - 1s 7ms/step - loss: 481.3220 - val_loss: 386.4347
Epoch 6/30
139/139 [=====] - 1s 6ms/step - loss: 453.9855 - val_loss: 357.2578
Epoch 7/30
139/139 [=====] - 1s 6ms/step - loss: 436.6833 - val_loss: 329.8915
Epoch 8/30
139/139 [=====] - 1s 6ms/step - loss: 416.7867 - val_loss: 385.3098
Epoch 9/30
139/139 [=====] - 1s 6ms/step - loss: 383.8130 - val_loss: 327.4228
Epoch 10/30
139/139 [=====] - 1s 7ms/step - loss: 369.5308 - val_loss: 314.5972
Epoch 11/30
139/139 [=====] - 1s 6ms/step - loss: 347.0515 - val_loss: 301.8029
Epoch 12/30
139/139 [=====] - 1s 6ms/step - loss: 329.7316 - val_loss: 277.8990
Epoch 13/30
139/139 [=====] - 1s 6ms/step - loss: 311.6165 - val_loss: 330.2468
Epoch 14/30
139/139 [=====] - 1s 6ms/step - loss: 291.6809 - val_loss: 279.0862
Epoch 15/30
139/139 [=====] - 1s 6ms/step - loss: 277.8235 - val_loss: 277.5264
Epoch 16/30
139/139 [=====] - 1s 7ms/step - loss: 275.2344 - val_loss: 265.4743
Epoch 17/30
139/139 [=====] - 1s 7ms/step - loss: 264.2656 - val_loss: 260.2368
Epoch 18/30
139/139 [=====] - 1s 7ms/step - loss: 245.6158 - val_loss: 271.9914
Epoch 19/30
139/139 [=====] - 1s 7ms/step - loss: 240.7386 - val_loss: 281.8378
Epoch 20/30
139/139 [=====] - 1s 7ms/step - loss: 236.0646 - val_loss: 262.6802
Epoch 21/30

```
139/139 [=====] - 1s 7ms/step - loss: 221.4383 - val_loss: 266.7111
Epoch 22/30
139/139 [=====] - 1s 7ms/step - loss: 217.4519 - val_loss: 277.5710
Epoch 23/30
139/139 [=====] - 1s 7ms/step - loss: 206.8189 - val_loss: 292.8675
Epoch 24/30
139/139 [=====] - 1s 6ms/step - loss: 199.3513 - val_loss: 244.5379
Epoch 25/30
139/139 [=====] - 1s 6ms/step - loss: 190.6853 - val_loss: 259.7134
Epoch 26/30
139/139 [=====] - 1s 6ms/step - loss: 185.7790 - val_loss: 245.3492
Epoch 27/30
139/139 [=====] - 1s 7ms/step - loss: 186.6141 - val_loss: 286.3978
Epoch 28/30
139/139 [=====] - 1s 7ms/step - loss: 180.2722 - val_loss: 291.8390
Epoch 29/30
139/139 [=====] - 1s 6ms/step - loss: 179.4201 - val_loss: 261.4296
Epoch 30/30
139/139 [=====] - 1s 6ms/step - loss: 169.4102 - val_loss: 274.2621
Saved CNN model to: C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD003 data & artefacts\models\cnn_fd003_se
q30.keras
```

LSTM MODEL

In [7]:

```
1 # --- LSTM model (train + save) ---
2
3 from lstm_model import build_lstm_model, train_lstm_model
4 import json
5
6 # 1) Load cached 3D windows
7 npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
8 X_train, y_train, X_val, y_val, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
9
10 # 2) Build LSTM (input: [seq_len, n_features])
11 input_shape = (SEQ_LEN, X_train.shape[2])
12 lstm = build_lstm_model(input_shape)
13
14 # 3) Train (pass only what your module expects)
15 # If your train_lstm_model returns (model, history), keep both; if it returns only model, handle that too.
16 result = train_lstm_model(lstm, X_train, y_train, X_val, y_val, epochs=50, batch_size=128)
17 if isinstance(result, tuple):
18     lstm, lstm_history = result
19 else:
20     lstm = result
21     lstm_history = None
22
23 # 4) Save model + minimal metadata
24 models_dir = ART_DIR / "models"
25 models_dir.mkdir(parents=True, exist_ok=True)
26
27 model_path = models_dir / f"lstm_{DATASET.lower()}_seq{SEQ_LEN}.keras"
28 lstm.save(model_path)
29
30 meta = {
31     "model_type": "lstm",
32     "dataset": DATASET,
33     "seq_len": int(SEQ_LEN),
34     "features": int(X_train.shape[2]),
35     "epochs": 50,
36     "batch_size": 128
37 }
38 with open(models_dir / f"lstm_{DATASET.lower()}_seq{SEQ_LEN}.meta.json", "w") as f:
39     json.dump(meta, f, indent=2)
40
```

```
41 print("Saved LSTM model to:", model_path)
```

Epoch 1/50
139/139 [=====] - 6s 32ms/step - loss: 9009.2578 - val_loss: 7946.7007
Epoch 2/50
139/139 [=====] - 4s 26ms/step - loss: 7985.2632 - val_loss: 7221.5098
Epoch 3/50
139/139 [=====] - 3s 24ms/step - loss: 7278.3848 - val_loss: 6577.6685
Epoch 4/50
139/139 [=====] - 3s 24ms/step - loss: 6639.9668 - val_loss: 5990.5103
Epoch 5/50
139/139 [=====] - 3s 23ms/step - loss: 6063.1558 - val_loss: 5456.3276
Epoch 6/50
139/139 [=====] - 3s 24ms/step - loss: 5530.3589 - val_loss: 4968.9062
Epoch 7/50
139/139 [=====] - 3s 23ms/step - loss: 5044.4531 - val_loss: 4515.5801
Epoch 8/50
139/139 [=====] - 3s 23ms/step - loss: 4593.1997 - val_loss: 4103.1821
Epoch 9/50
139/139 [=====] - 3s 25ms/step - loss: 4189.4971 - val_loss: 3726.2859
Epoch 10/50
139/139 [=====] - 3s 24ms/step - loss: 3800.8279 - val_loss: 3374.3955
Epoch 11/50
139/139 [=====] - 3s 24ms/step - loss: 3447.3911 - val_loss: 3052.4436
Epoch 12/50
139/139 [=====] - 3s 24ms/step - loss: 3116.2444 - val_loss: 2762.2205
Epoch 13/50
139/139 [=====] - 3s 24ms/step - loss: 2831.6741 - val_loss: 2487.1804
Epoch 14/50
139/139 [=====] - 3s 23ms/step - loss: 2560.1147 - val_loss: 2237.3413
Epoch 15/50
139/139 [=====] - 3s 23ms/step - loss: 2304.8499 - val_loss: 2010.0829
Epoch 16/50
139/139 [=====] - 3s 23ms/step - loss: 2084.2686 - val_loss: 1802.2012
Epoch 17/50
139/139 [=====] - 3s 23ms/step - loss: 1874.3990 - val_loss: 1614.1465
Epoch 18/50
139/139 [=====] - 3s 23ms/step - loss: 1685.9531 - val_loss: 1443.7773
Epoch 19/50
139/139 [=====] - 3s 23ms/step - loss: 1509.5267 - val_loss: 1291.2517
Epoch 20/50
139/139 [=====] - 3s 23ms/step - loss: 1359.2804 - val_loss: 1153.6637
Epoch 21/50

139/139 [=====] - 3s 23ms/step - loss: 1211.1892 - val_loss: 1016.3416
Epoch 22/50
139/139 [=====] - 3s 23ms/step - loss: 1075.2662 - val_loss: 907.0706
Epoch 23/50
139/139 [=====] - 3s 23ms/step - loss: 961.4497 - val_loss: 804.8181
Epoch 24/50
139/139 [=====] - 3s 23ms/step - loss: 860.8553 - val_loss: 721.3382
Epoch 25/50
139/139 [=====] - 3s 23ms/step - loss: 761.2798 - val_loss: 652.1026
Epoch 26/50
139/139 [=====] - 3s 24ms/step - loss: 680.7581 - val_loss: 566.5023
Epoch 27/50
139/139 [=====] - 3s 23ms/step - loss: 609.7139 - val_loss: 513.1118
Epoch 28/50
139/139 [=====] - 3s 23ms/step - loss: 545.2030 - val_loss: 441.9966
Epoch 29/50
139/139 [=====] - 3s 23ms/step - loss: 489.7608 - val_loss: 404.6142
Epoch 30/50
139/139 [=====] - 3s 23ms/step - loss: 438.2371 - val_loss: 355.6134
Epoch 31/50
139/139 [=====] - 3s 23ms/step - loss: 392.7744 - val_loss: 325.0167
Epoch 32/50
139/139 [=====] - 3s 23ms/step - loss: 355.0634 - val_loss: 292.5157
Epoch 33/50
139/139 [=====] - 3s 23ms/step - loss: 318.6096 - val_loss: 266.1568
Epoch 34/50
139/139 [=====] - 3s 23ms/step - loss: 292.8992 - val_loss: 262.9375
Epoch 35/50
139/139 [=====] - 3s 23ms/step - loss: 266.7509 - val_loss: 250.2795
Epoch 36/50
139/139 [=====] - 3s 24ms/step - loss: 248.1743 - val_loss: 246.7417
Epoch 37/50
139/139 [=====] - 3s 23ms/step - loss: 227.6380 - val_loss: 203.8736
Epoch 38/50
139/139 [=====] - 3s 23ms/step - loss: 215.2697 - val_loss: 200.8011
Epoch 39/50
139/139 [=====] - 3s 23ms/step - loss: 199.6657 - val_loss: 254.2719
Epoch 40/50
139/139 [=====] - 3s 23ms/step - loss: 189.4465 - val_loss: 227.1487
Epoch 41/50
139/139 [=====] - 3s 23ms/step - loss: 178.3645 - val_loss: 189.8092
Epoch 42/50


```
139/139 [=====] - 3s 23ms/step - loss: 168.9283 - val_loss: 221.3502
Epoch 43/50
139/139 [=====] - 3s 23ms/step - loss: 162.4426 - val_loss: 205.7650
Epoch 44/50
139/139 [=====] - 3s 23ms/step - loss: 151.1867 - val_loss: 239.7700
Epoch 45/50
139/139 [=====] - 3s 23ms/step - loss: 147.5399 - val_loss: 202.5762
Epoch 46/50
139/139 [=====] - 3s 23ms/step - loss: 145.6921 - val_loss: 211.4368
Epoch 47/50
139/139 [=====] - 3s 23ms/step - loss: 140.5016 - val_loss: 208.9493
Epoch 48/50
139/139 [=====] - 3s 23ms/step - loss: 136.7758 - val_loss: 215.6450
Epoch 49/50
139/139 [=====] - 3s 23ms/step - loss: 133.5197 - val_loss: 217.7166
Epoch 50/50
139/139 [=====] - 3s 23ms/step - loss: 131.3537 - val_loss: 208.7234
Saved LSTM model to: C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD003 data & artefacts\models\lstm_fd003_
seq30.keras
```

CNN_LSTM Model

In [8]:

```
1  # --- CNN-LSTM model (train + save) ---
2
3  from cnn_lstm_model import build_cnn_lstm_model, train_cnn_lstm_model
4  import json
5
6  # 1) Load cached 3D windows
7  npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
8  X_train, y_train, X_val, y_val, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
9
10 # 2) Build model (input: [seq_len, n_features])
11 input_shape = (SEQ_LEN, X_train.shape[2])
12 cnnlstm = build_cnn_lstm_model(input_shape)
13
14 # 3) Train (handle: returns History, returns Model, or returns (Model, History))
15 train_ret = train_cnn_lstm_model(
16     cnnlstm, X_train, y_train, X_val, y_val,
17     epochs=50, batch_size=128
18 )
19
20 # Normalize outputs
21 cnnlstm_history = None
22 model_to_save = cnnlstm # fallback: the model we built
23
24 # Case A: (model, history)
25 if isinstance(train_ret, tuple) and len(train_ret) >= 1:
26     if hasattr(train_ret[0], "save"):
27         model_to_save = train_ret[0]
28     if len(train_ret) >= 2 and hasattr(train_ret[1], "history"):
29         cnnlstm_history = train_ret[1]
30
31 # Case B: just a model
32 elif hasattr(train_ret, "save"):
33     model_to_save = train_ret
34
35 # Case C: just a History
36 elif hasattr(train_ret, "history"):
37     cnnlstm_history = train_ret
38
39 # 4) Save model + minimal metadata
40 models_dir = ART_DIR / "models"
41 models_dir.mkdir(parents=True, exist_ok=True)
```

```
42
43 model_path = models_dir / f"cnn_lstm_{DATASET.lower()}_seq{SEQ_LEN}.keras"
44 model_to_save.save(model_path)
45
46 meta = {
47     "model_type": "cnn_lstm",
48     "dataset": DATASET,
49     "seq_len": int(SEQ_LEN),
50     "features": int(X_train.shape[2]),
51     "epochs": 50,           # keep in sync with your fit(...)
52     "batch_size": 128      # keep in sync with your fit(...)
53 }
54 with open(models_dir / f"cnn_lstm_{DATASET.lower()}_seq{SEQ_LEN}.meta.json", "w") as f:
55     json.dump(meta, f, indent=2)
56
57 # (Optional) persist training history if available
58 if cnnlstm_history:
59     hist_path = models_dir / f"cnn_lstm_{DATASET.lower()}_seq{SEQ_LEN}.history.json"
60     with open(hist_path, "w") as f:
61         json.dump(cnnlstm_history.history, f, indent=2)
62
63
64 print("Saved CNN-LSTM model to:", model_path)
```

Epoch 1/50
139/139 [=====] - 4s 17ms/step - loss: 5317.8711 - mae: 61.2980 - val_loss: 1627.5839 - val_mae: 35.1610

Epoch 2/50
139/139 [=====] - 2s 15ms/step - loss: 740.8433 - mae: 21.8212 - val_loss: 315.1867 - val_mae: 13.7657

Epoch 3/50
139/139 [=====] - 2s 15ms/step - loss: 486.6523 - mae: 17.0407 - val_loss: 340.9031 - val_mae: 13.6077

Epoch 4/50
139/139 [=====] - 2s 15ms/step - loss: 445.0238 - mae: 16.1343 - val_loss: 291.4069 - val_mae: 12.1913

Epoch 5/50
139/139 [=====] - 2s 15ms/step - loss: 407.1817 - mae: 15.3726 - val_loss: 298.0509 - val_mae: 13.9845

Epoch 6/50
139/139 [=====] - 2s 15ms/step - loss: 364.8745 - mae: 14.5724 - val_loss: 470.7306 - val_mae: 17.3392

Epoch 7/50
139/139 [=====] - 2s 15ms/step - loss: 321.0126 - mae: 13.7285 - val_loss: 276.8845 - val_mae: 12.5189

Epoch 8/50
139/139 [=====] - 2s 15ms/step - loss: 283.8782 - mae: 12.9082 - val_loss: 362.3007 - val_mae: 13.1953

Epoch 9/50
139/139 [=====] - 2s 15ms/step - loss: 273.7037 - mae: 12.6662 - val_loss: 329.0597 - val_mae: 12.3519

Epoch 10/50
139/139 [=====] - 2s 15ms/step - loss: 260.7218 - mae: 12.4129 - val_loss: 417.2903 - val_mae: 16.8161

Epoch 11/50
139/139 [=====] - 2s 15ms/step - loss: 265.3242 - mae: 12.5077 - val_loss: 349.3683 - val_mae: 13.0723

Epoch 12/50
139/139 [=====] - 2s 15ms/step - loss: 242.5362 - mae: 11.9520 - val_loss: 367.1696 - val_mae: 12.9347

Epoch 13/50
139/139 [=====] - 2s 15ms/step - loss: 246.8730 - mae: 12.1147 - val_loss: 358.2558 - val_mae: 12.8235

Epoch 14/50
139/139 [=====] - 2s 15ms/step - loss: 236.6196 - mae: 11.7461 - val_loss: 305.1389 - val_mae:

```

12.6500
Epoch 15/50
139/139 [=====] - 2s 15ms/step - loss: 230.9341 - mae: 11.7068 - val_loss: 381.4654 - val_mae:
14.7925
Epoch 16/50
139/139 [=====] - 2s 15ms/step - loss: 226.9448 - mae: 11.4881 - val_loss: 363.6549 - val_mae:
13.4198
Epoch 17/50
139/139 [=====] - 2s 15ms/step - loss: 228.4237 - mae: 11.5835 - val_loss: 289.4308 - val_mae:
12.0695
Saved CNN-LSTM model to: C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD003 data & artefacts\models\cnn_lstm_fd003_seq30.keras

```

===== End of Training =====

===== Model Analysis =====

LOAD MODEL

In [9]:

```

1 # LOAD baseline MODEL
2 model_paths = build_model_paths() # will only load what exists
3 print("Will load:", model_paths)
4
5 models = load_models(model_paths)
6 print("Loaded models:", list(models.keys()))

```

```

Will load: {'Base': 'C:\\Users\\mg020649\\Documents\\15 - Coding\\Msc-Project-main\\FD003 data & artefacts\\models\\base_linear_fd003_seq30_last.joblib', 'LSTM': 'C:\\Users\\mg020649\\Documents\\15 - Coding\\Msc-Project-main\\FD003 data & artefacts\\models\\lstm_fd003_seq30.keras', 'CNN': 'C:\\Users\\mg020649\\Documents\\15 - Coding\\Msc-Project-main\\FD003 data & artefacts\\models\\cnn_fd003_seq30.keras', 'CNN-LSTM': 'C:\\Users\\mg020649\\Documents\\15 - Coding\\Msc-Project-main\\FD003 data & artefacts\\models\\cnn_lstm_fd003_seq30.keras'}
Loaded models: ['Base', 'LSTM', 'CNN', 'CNN-LSTM']

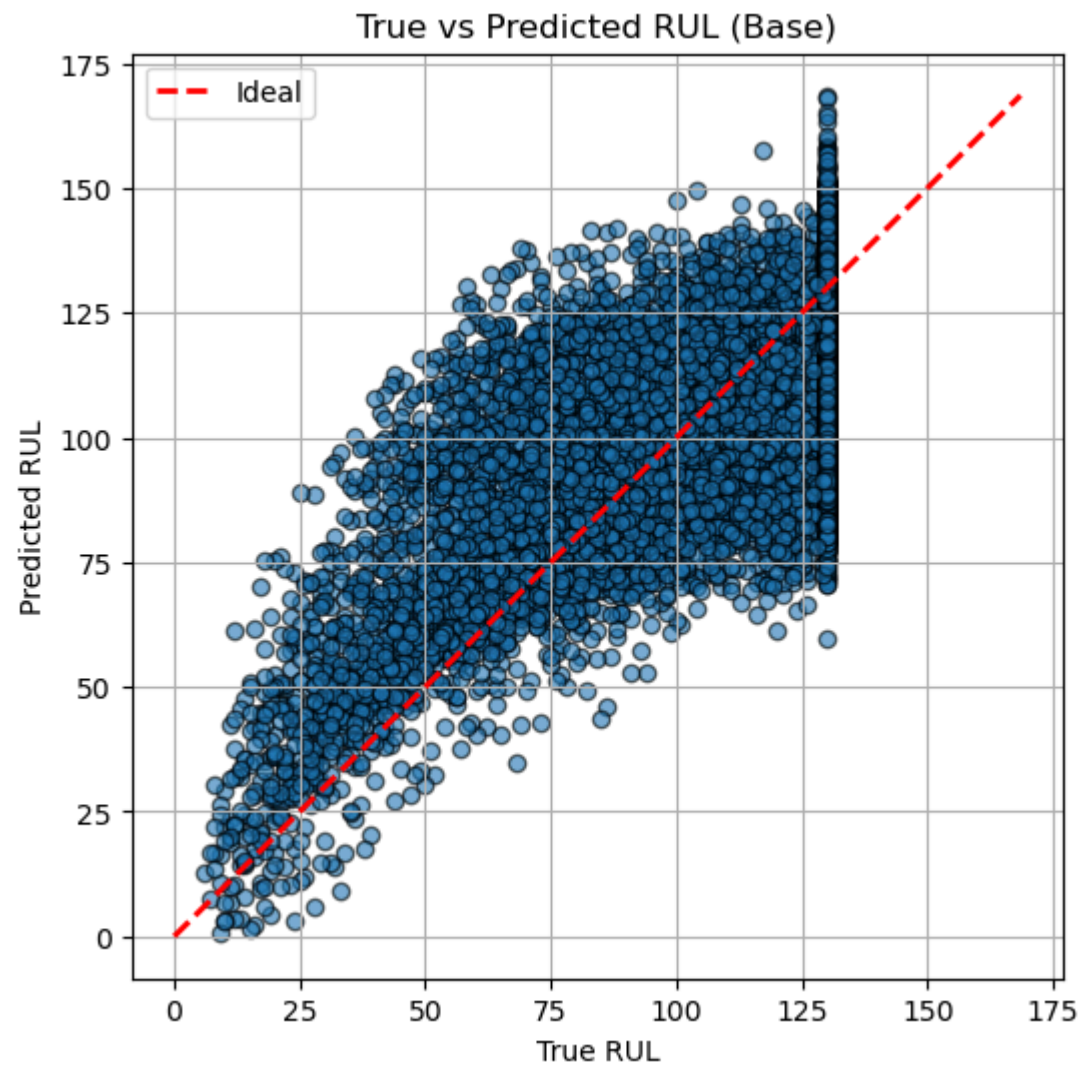
```

=====ALL MODEL LOADED=====

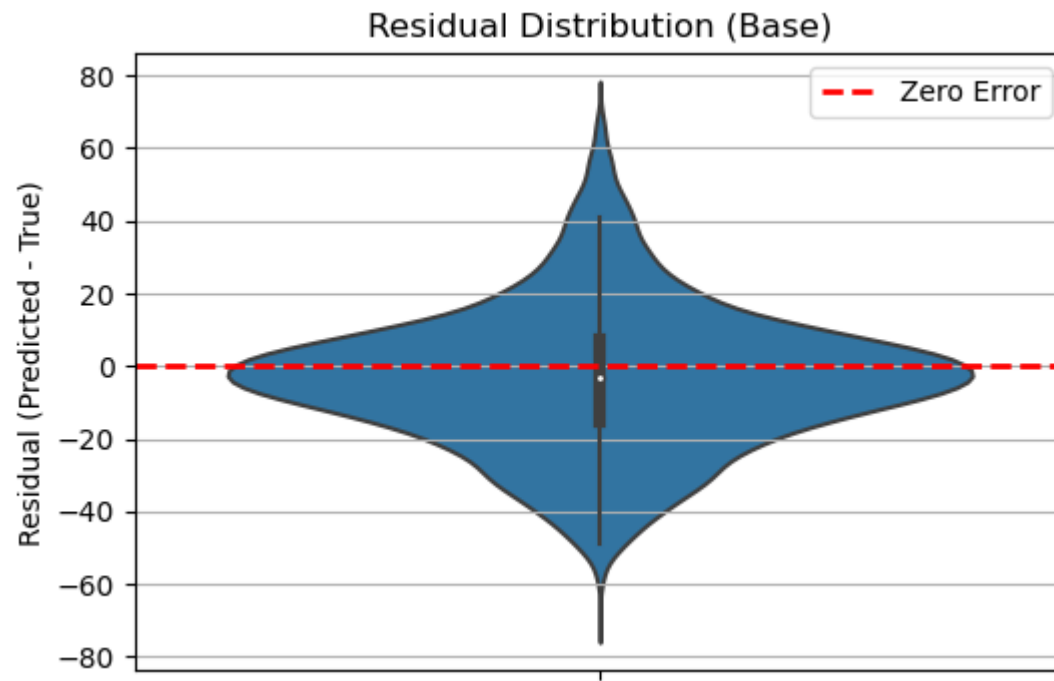
Base build Analysis

In [10]:

```
1  # plot_true_vs_pred
2
3  npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
4
5  _, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
6
7  X_test_base = pp.make_feature_vectors_from_windows(X_test, strategy="last")
8
9  y_pred_base = models["Base"].predict(X_test_base).ravel()
10
11  y_true = y_test
12
13  plot_true_vs_pred(y_true, y_pred_base, model_name="Base")
```

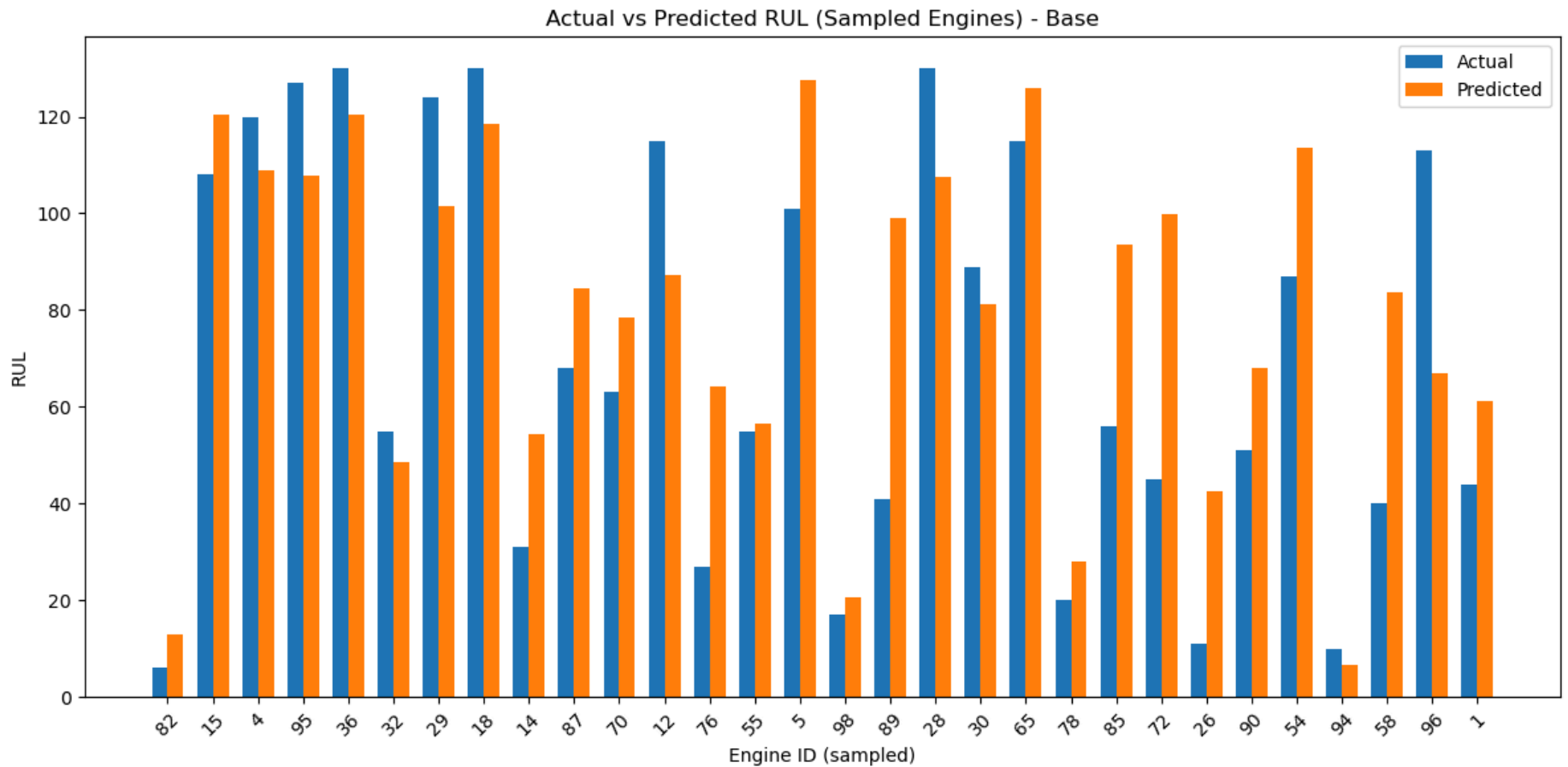
```
In [11]: 1 # Now plot residuals  
2 plot_residuals(y_true, y_pred_base, model_name="Base", kind="violin")
```



```

In [12]: 1 # Build unit IDs aligned to the windowed test data (X_test, y_test)
2 ids = []
3 for uid, grp in test_scaled.groupby("unit_number"):
4     grp = grp.sort_values("time_in_cycles")
5     n = len(grp) - SEQ_LEN + 1
6     if n > 0:
7         ids.extend([uid] * n)
8 unit_ids_aligned = np.array(ids)
9
10 # Now plot
11 plot_per_engine_bars(y_true, y_pred_base, unit_ids_aligned, model_name="Base")
12

```



===== End of Base TEST =====

CNN Model Analysis

In [13]:

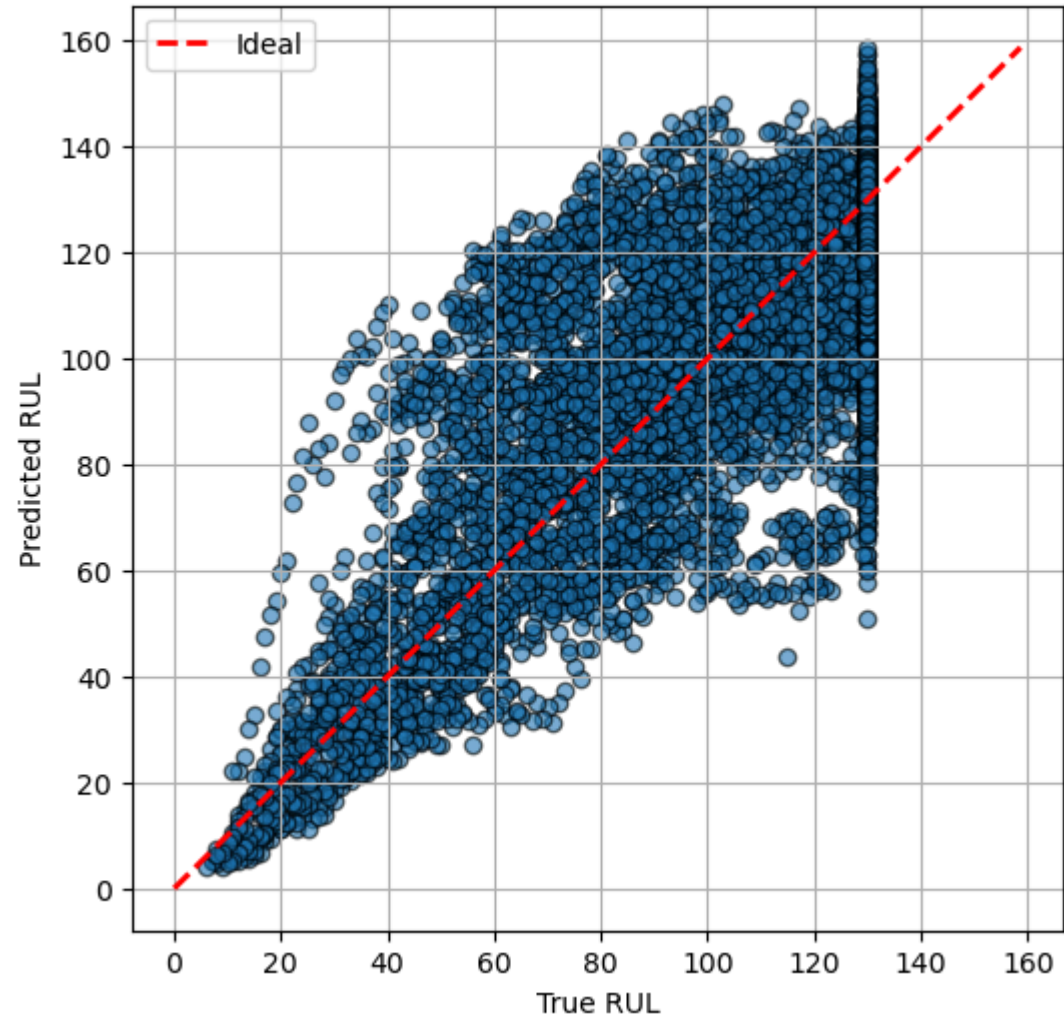
```
1 # CNN Prediction
2 from cnn_model import predict_cnn_model
3 y_pred_cnn = predict_cnn_model(models["CNN"], X_test)
```

428/428 [=====] - 0s 970us/step

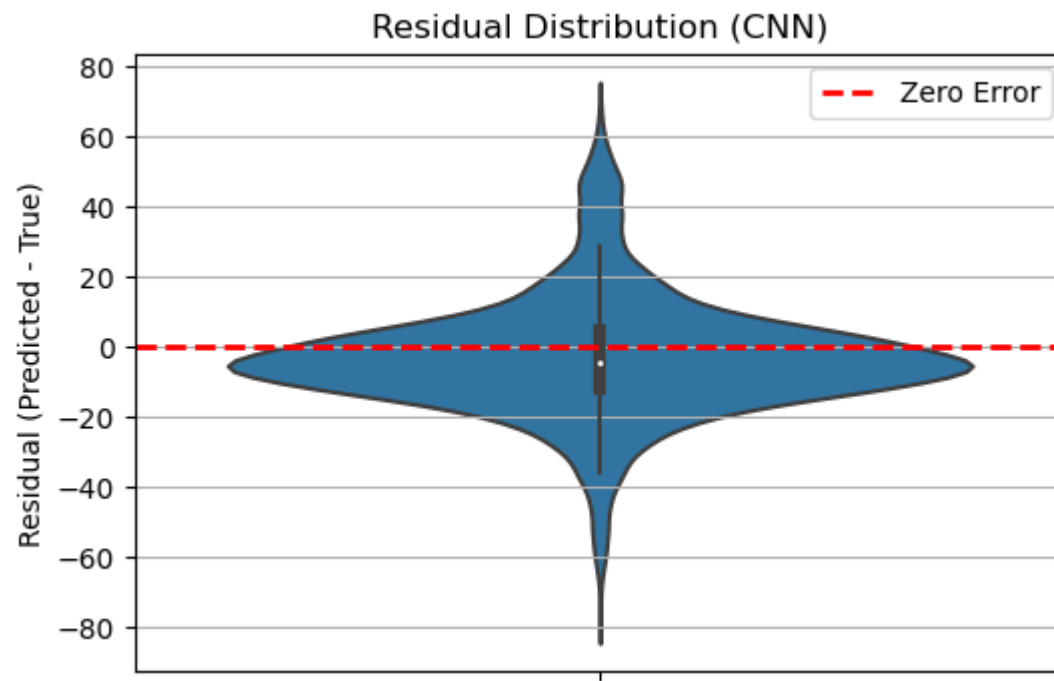
In [14]:

```
1 # plot_true_vs_pred
2
3 # can be rid, brought as don't want to run model build code again
4 npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
5 _, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
6
7 y_true = y_test
8
9 plot_true_vs_pred(y_true, y_pred_cnn, model_name="CNN")
```

True vs Predicted RUL (CNN)



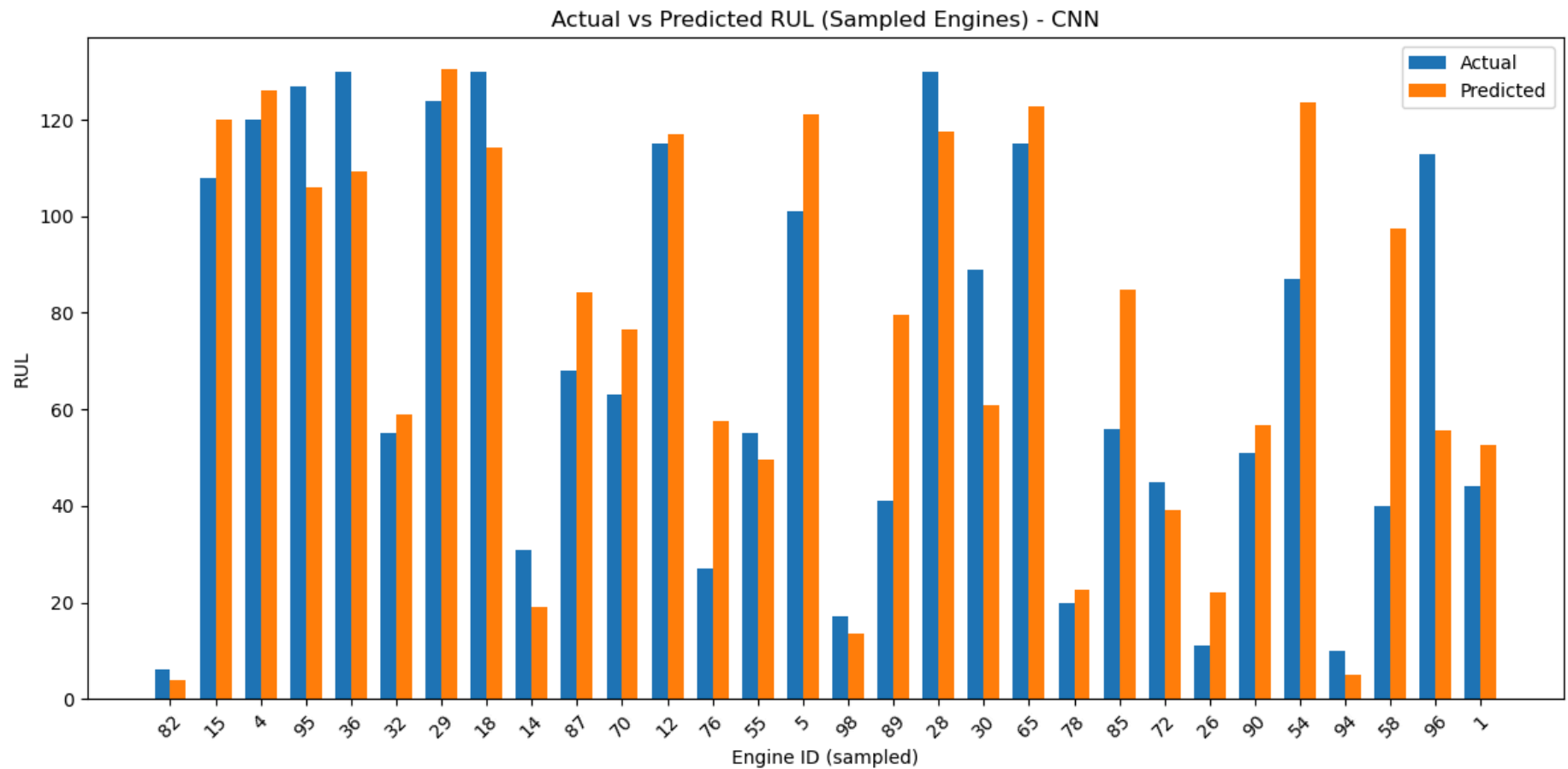
```
In [15]: 1 plot_residuals(y_true, y_pred_cnn, model_name="CNN", kind="violin")
```



```

In [16]: 1 # Build unit IDs aligned to the windowed test data (X_test, y_test)
2 ids = []
3 for uid, grp in test_scaled.groupby("unit_number"):
4     grp = grp.sort_values("time_in_cycles")
5     n = len(grp) - SEQ_LEN + 1
6     if n > 0:
7         ids.extend([uid] * n)
8 unit_ids_aligned = np.array(ids)
9
10 # Now plot
11 plot_per_engine_bars(y_true, y_pred_cnn, unit_ids_aligned, model_name="CNN")

```



===== End of CNN TEST =====

LSTM ANALYSIS

In [17]:

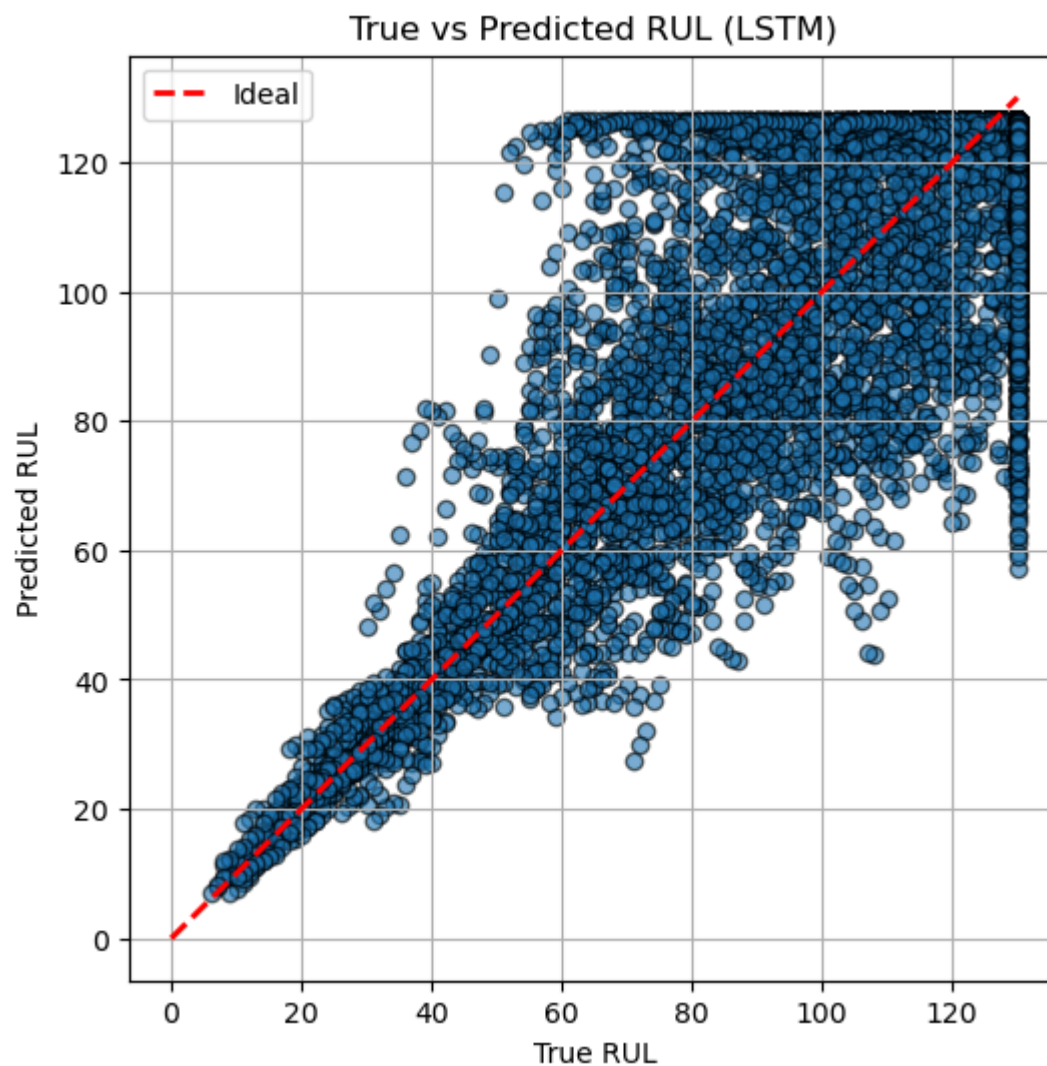
```
1 # CNN Prediction
2 from lstm_model import predict_lstm_model
3 y_pred_lstm = predict_lstm_model(models["LSTM"], X_test)
```

428/428 [=====] - 2s 4ms/step

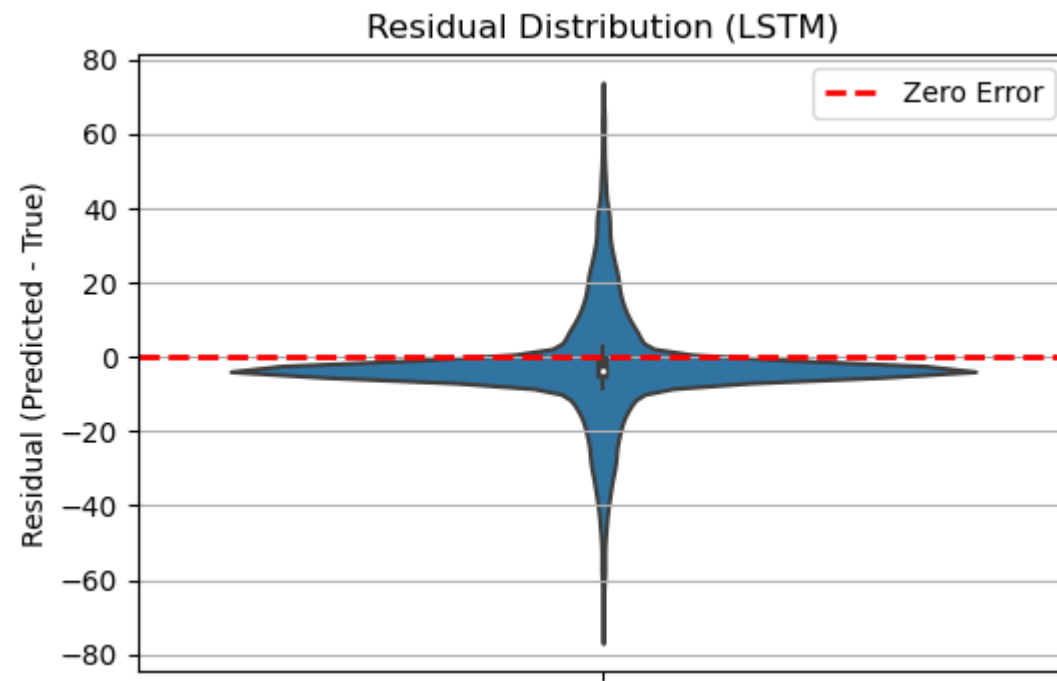
```

In [18]: 1 # plot_true_vs_pred
2
3 # can be rid, brought as don't want to run model build code again
4 npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
5 _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
6
7 y_true = y_test
8 plot_true_vs_pred(y_true, y_pred_lstm, model_name="LSTM")

```



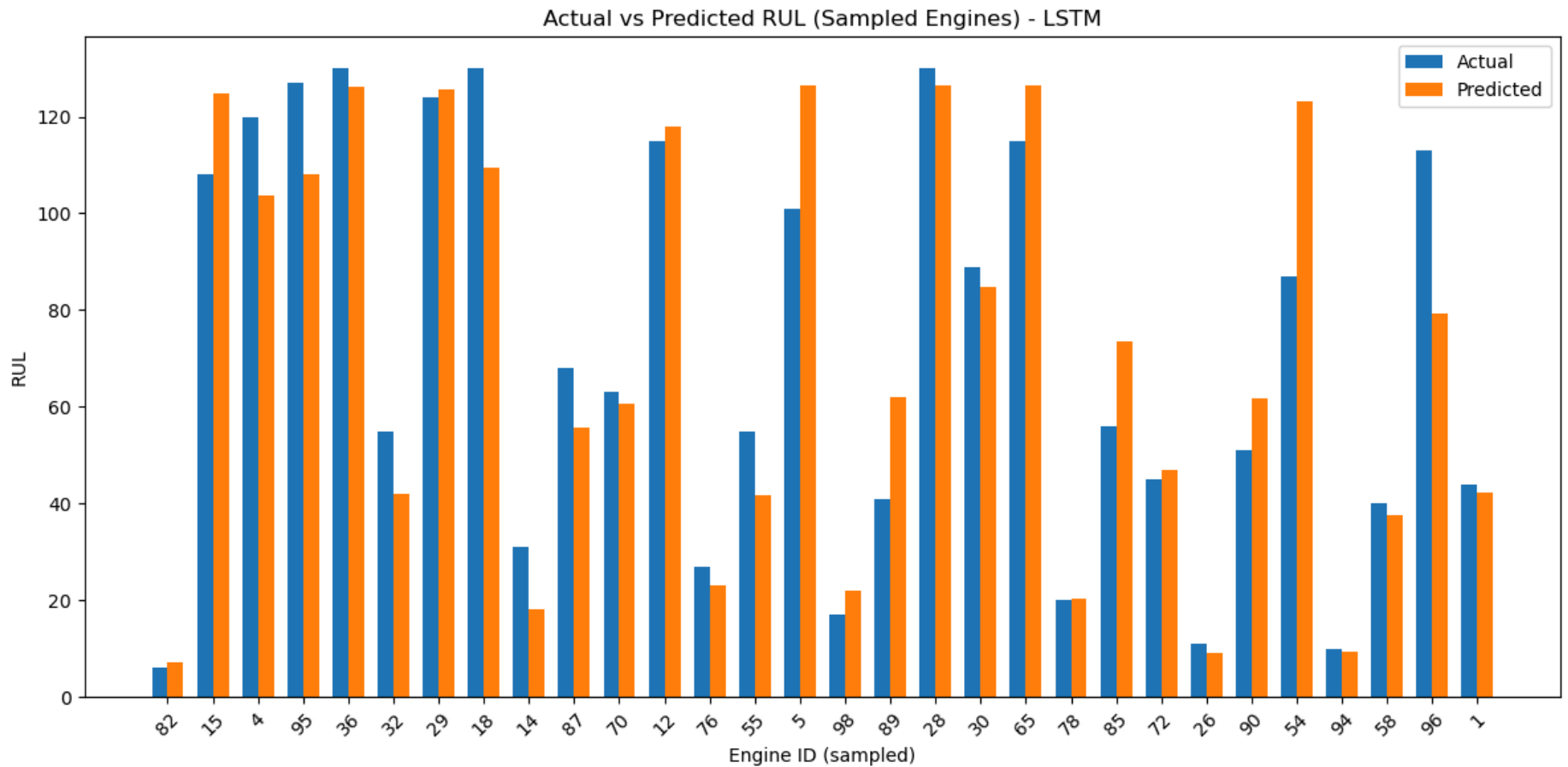
```
In [19]: 1 plot_residuals(y_true, y_pred_lstm, model_name="LSTM", kind="violin")
```



```

In [20]: 1 # Build unit IDs aligned to the windowed test data (X_test, y_test)
2 ids = []
3 for uid, grp in test_scaled.groupby("unit_number"):
4     grp = grp.sort_values("time_in_cycles")
5     n = len(grp) - SEQ_LEN + 1
6     if n > 0:
7         ids.extend([uid] * n)
8 unit_ids_aligned = np.array(ids)
9
10 # Now plot
11 plot_per_engine_bars(y_true, y_pred_lstm, unit_ids_aligned, model_name="LSTM")
12

```



===== End of LSTM TEST =====

LSTM_CNN Build Analysis

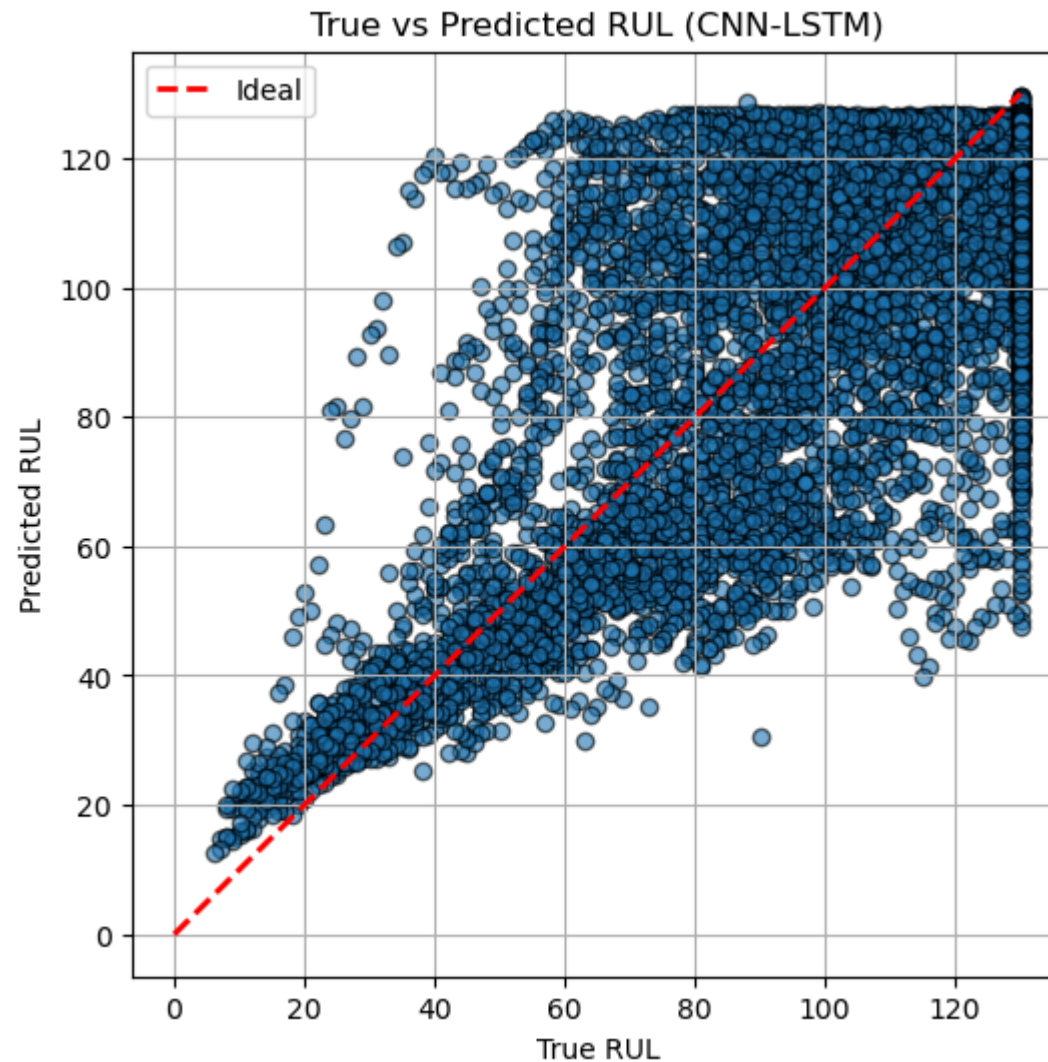
In [21]:

```
1 # CNN Prediction
2 from cnn_lstm_model import predict_cnn_lstm_model
3 y_pred_cnn_lstm = predict_cnn_lstm_model(models["CNN-LSTM"], X_test)
```

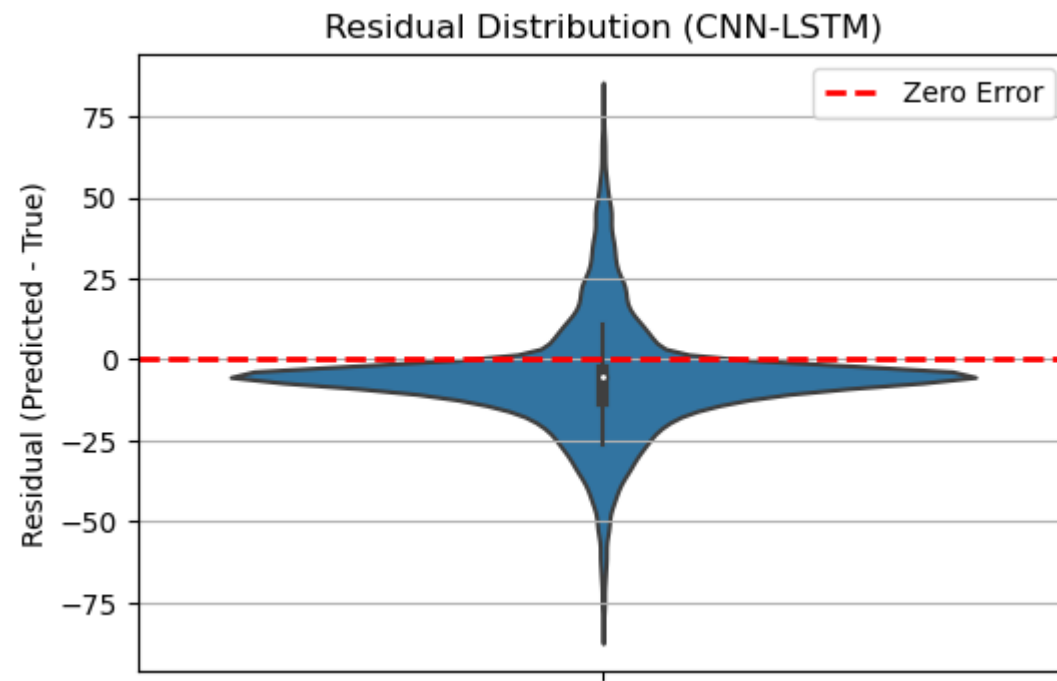
```

In [22]: 1 # plot_true_vs_pred
          2
          3 # can be rid, brought as don't want to run model build code again
          4 npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
          5 _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
          6
          7 y_true = y_test
          8 plot_true_vs_pred(y_true, y_pred_cnn_lstm, model_name="CNN-LSTM")

```



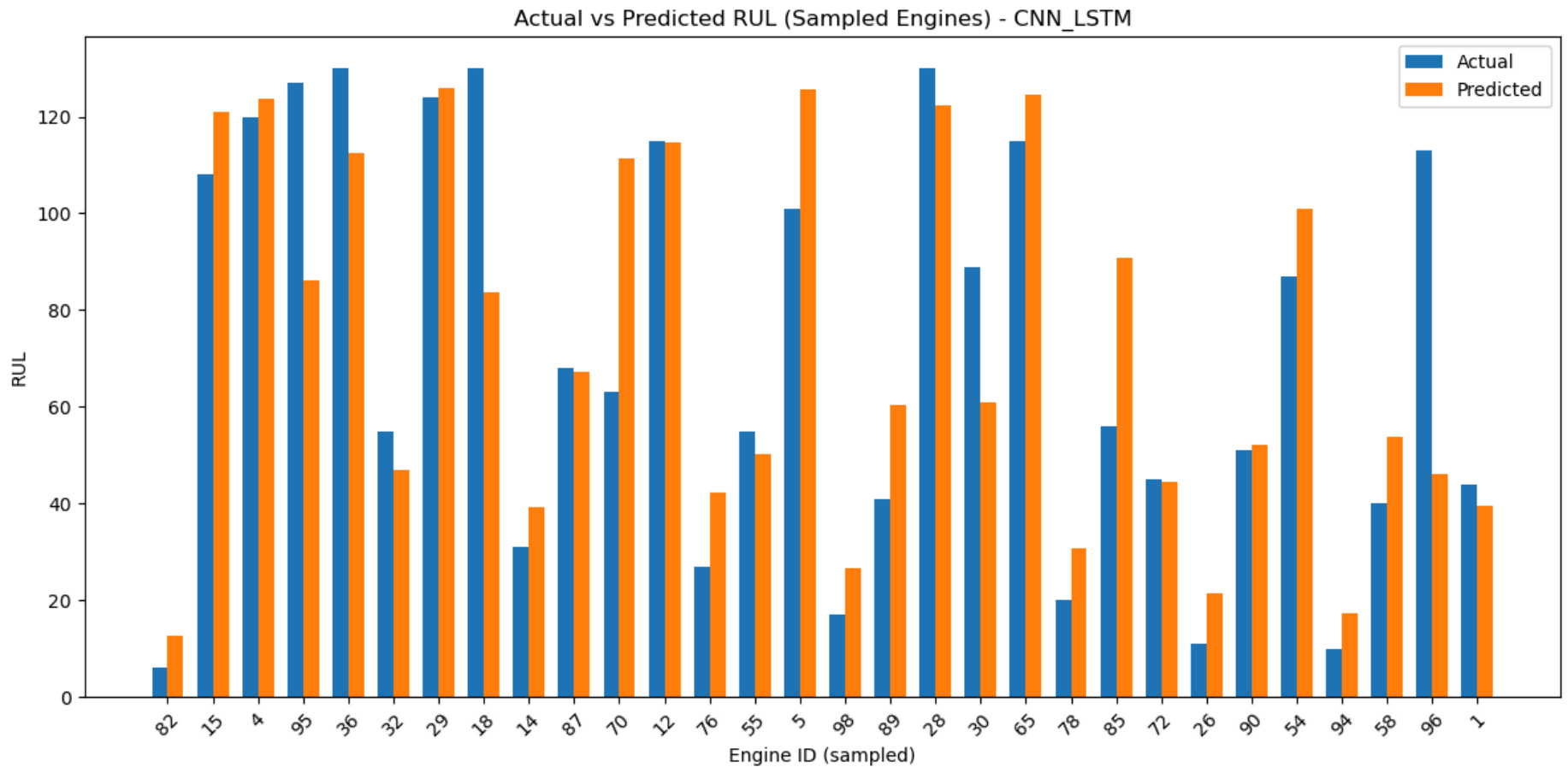
```
In [23]: 1 plot_residuals(y_true, y_pred_cnn_lstm, model_name="CNN-LSTM", kind="violin")
```



```

In [24]: 1 # Build unit IDs aligned to the windowed test data (X_test, y_test)
2 ids = []
3 for uid, grp in test_scaled.groupby("unit_number"):
4     grp = grp.sort_values("time_in_cycles")
5     n = len(grp) - SEQ_LEN + 1
6     if n > 0:
7         ids.extend([uid] * n)
8 unit_ids_aligned = np.array(ids)
9
10 # Now plot
11 plot_per_engine_bars(y_true, y_pred_cnn_lstm, unit_ids_aligned, model_name="CNN_LSTM")
12

```



===== End of CNN_LSTM TEST =====

===== Evaluation Section =====

===== RMSE & MAE =====

base model

```
In [25]: 1 from evaluator import evaluate_model
2
3 # Load preprocessed test set
4 npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
5 _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
6
7 # convert if using base model (needs 2D vectors)
8 X_test_feat = pp.make_feature_vectors_from_windows(X_test, strategy="last")
9
10 # Load trained base model
11 model_path = ART_DIR / "models" / f"base_linear_{DATASET.lower()}_seq{SEQ_LEN}_last.joblib"
12 base_model = joblib.load(model_path)
13
14 # predict & evaluate
15 y_base_pred = base_model.predict(X_test_feat)
16 evaluate_model(y_test, y_base_pred, model_name="Base Linear Model")
```

Base Linear Model Evaluation:

RMSE: 20.1671

MAE : 15.3374

```
Out[25]: {'model': 'Base Linear Model', 'RMSE': 20.167126, 'MAE': 15.337351}
```

CNN model

```
In [26]: 1 # --- Predict & evaluate: CNN on FD001 test ---
2
3 from evaluator import evaluate_model
4
5 # 1) Load cached test windows
6 npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
7 _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
8
9 # 2) Load saved CNN
10 cnn_path = ART_DIR / "models" / f"cnn_{DATASET.lower()}_seq{SEQ_LEN}.keras"
11 cnn_model = load_model(cnn_path)
12
13 # 3) Predict (CNN expects 3D windows)
14 y_cnn_pred = cnn_model.predict(X_test, verbose=0).squeeze()
15
16 # 4) Evaluate
17 evaluate_model(y_test, y_cnn_pred, model_name="CNN")
```

CNN Evaluation:

RMSE: 17.4459

MAE : 12.6242

```
Out[26]: {'model': 'CNN', 'RMSE': 17.44589, 'MAE': 12.624193}
```

LSTM model

```
In [27]: 1 # --- Predict & evaluate: LSTM on FD001 test ---
2
3 from evaluator import evaluate_model
4
5 # 1) Load cached test windows
6 npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
7 _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
8
9 # 2) Load saved LSTM
10 lstm_path = ART_DIR / "models" / f"lstm_{DATASET.lower()}_seq{SEQ_LEN}.keras"
11 lstm_model = load_model(lstm_path)
12
13 # 3) Predict (LSTM expects 3D input)
14 y_lstm_pred = lstm_model.predict(X_test, verbose=0).squeeze()
15
16 # 4) Evaluate
17 evaluate_model(y_test, y_lstm_pred, model_name="LSTM")
```

LSTM Evaluation:

RMSE: 14.3806

MAE : 9.3269

```
Out[27]: {'model': 'LSTM', 'RMSE': 14.3806, 'MAE': 9.326885}
```

CNN_LSTM model

```
In [28]: 1 # --- Predict & evaluate: LSTM on FD001 test ---
2
3 from evaluator import evaluate_model
4
5 # 1) Load cached test windows
6 npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
7 _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))
8
9 # 2) Load saved LSTM
10 lstm_path = ART_DIR / "models" / f"cnn_lstm_{DATASET.lower()}_seq{SEQ_LEN}.keras"
11 lstm_model = load_model(lstm_path)
12
13 # 3) Predict (LSTM expects 3D input)
14 y_cnn_lstm_pred = lstm_model.predict(X_test, verbose=0).squeeze()
15
16 # 4) Evaluate
17 evaluate_model(y_test, y_cnn_lstm_pred, model_name="CNN_LSTM")
```

CNN_LSTM Evaluation:

RMSE: 18.0453

MAE : 12.9718

```
Out[28]: {'model': 'CNN_LSTM', 'RMSE': 18.045258, 'MAE': 12.971771}
```

```
In [29]: 1 from evaluator import evaluate_model
2
3 # build tables
4
5 res_cnn      = evaluate_model(y_test, y_pred_cnn,      model_name="CNN")
6 res_lstm     = evaluate_model(y_test, y_lstm_pred,     model_name="LSTM")
7 res_cnnlstm  = evaluate_model(y_test, y_cnn_lstm_pred, model_name="CNN-LSTM")
8 res_base     = evaluate_model(y_test, y_base_pred,     model_name="Base")
9
10 metrics = [res_cnn, res_lstm, res_cnnlstm, res_base]
11 metrics
```

CNN Evaluation:

RMSE: 17.4459

MAE : 12.6242

LSTM Evaluation:

RMSE: 14.3806

MAE : 9.3269

CNN-LSTM Evaluation:

RMSE: 18.0453

MAE : 12.9718

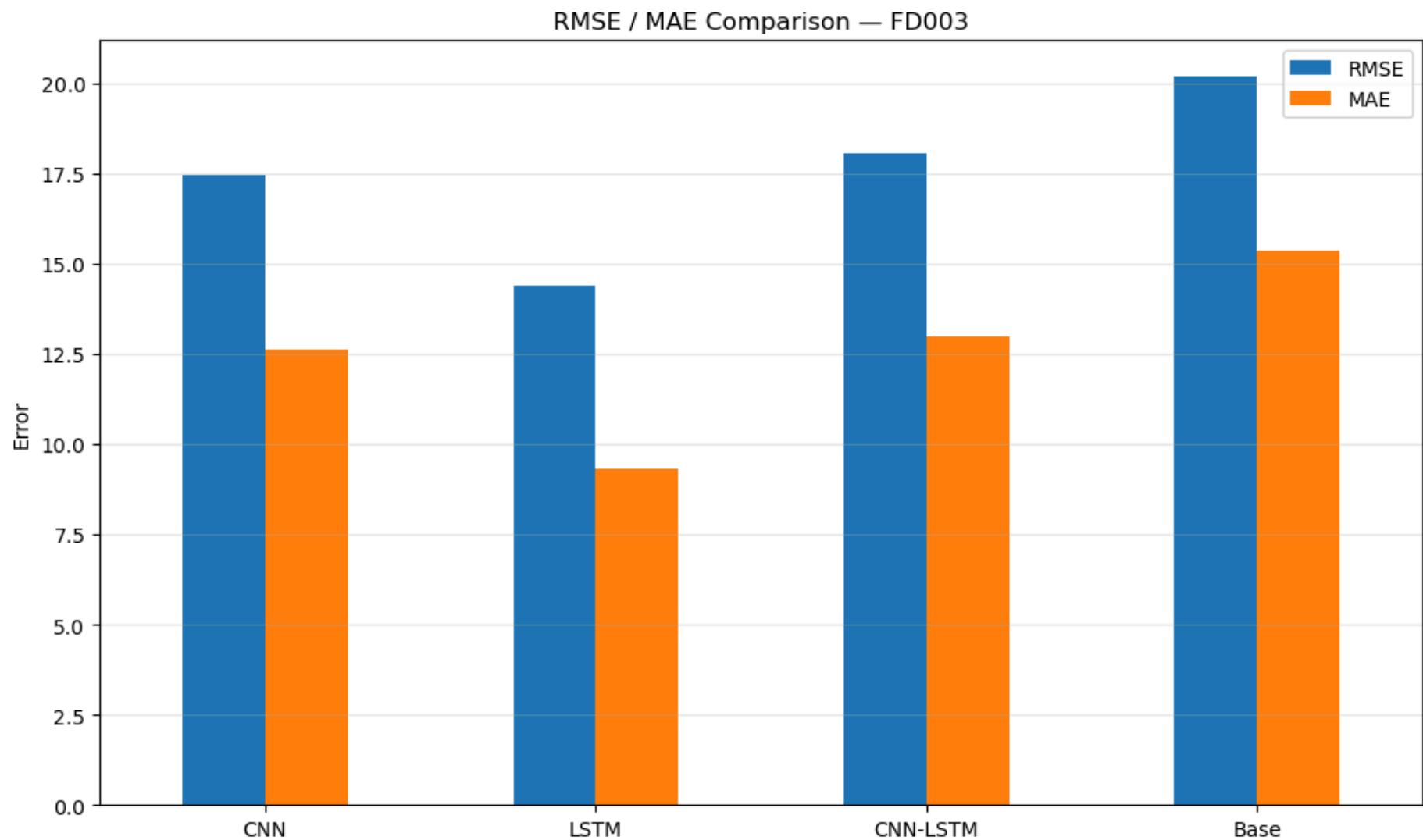
Base Evaluation:

RMSE: 20.1671

MAE : 15.3374

```
Out[29]: [{'model': 'CNN', 'RMSE': 17.44589, 'MAE': 12.624193},
{'model': 'LSTM', 'RMSE': 14.3806, 'MAE': 9.326885},
{'model': 'CNN-LSTM', 'RMSE': 18.045258, 'MAE': 12.971771},
{'model': 'Base', 'RMSE': 20.167126, 'MAE': 15.337351}]
```

```
In [30]: 1 plot_metric_comparison(metrics, dataset_name=DATASET)
```



```
In [31]: 1 metrics_df = pd.DataFrame([res_cnn, res_lstm, res_cnnlstm, res_base])[
2       ["model", "RMSE", "MAE"]
3       ].rename(columns={"model": "Model"})
4
5 metrics_df = metrics_df.sort_values("RMSE").reset_index(drop=True).round(2)
6 display(metrics_df)
```

	Model	RMSE	MAE
0	LSTM	14.380000	9.33
1	CNN	17.450001	12.62
2	CNN-LSTM	18.049999	12.97
3	Base	20.170000	15.34

```
In [32]: 1 metrics_df.to_csv(ART_DIR / f"{DATASET.lower()}_metrics_seq{SEQ_LEN}.csv", index=False)
```