Build Models and Analyse FD002

In [1]:

```
import os
os.environ["KERAS_BACKEND"] = "torch"
os.environ["PYTORCH_ENABLE_MPS_FALLBACK"] = "1"
os.environ["PYTORCH_MPS_HIGH_WATERMARK_RATIO"] = "0.0"  # optional: reduce MPS memory pressure

from keras.models import load_model
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import joblib
from keras.models import load_model as keras_load_model
import random
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]:

```
# --- functions---

def load_any_model(path):
    """
    Load a model given its file path, handling both Keras (.keras/.h5) and joblib (.joblib/.pkl).
    """
    path = Path(path)
    suf = path.suffix.lower()
    if suf in {".keras", ".h5", ".hdf5"}:
        return keras_load_model(path)
    if suf in {".joblib", ".pkl"}:
        return joblib.load(path)
    raise ValueError(f"Unrecognized model suffix for {path.name}")

def load_models(model_paths: dict):
    """
    model_paths: dict like {"Base": "...joblib", "LSTM": "...keras", "CNN": "...keras", "CNN-LSTM": "...keras"}
    returns: dict {name: model_object}
    """
    models = {}
    for name, path in model_paths.items():
        models[name] = load_any_model(path)
    return models

def build_model_paths(dataset=None, seq_len=None, strategy="last", art_dir=None,
                      include=("Base", "LSTM", "CNN", "CNN-LSTM")):
    """
    Build default file paths for your saved models in <ART_DIR>/models/…
    Base:      base_linear_{dataset.lower()}_seq{seq_len}_{strategy}.joblib
    LSTM:      lstm_{dataset.lower()}_seq{seq_len}.keras
    CNN:       cnn_{dataset.lower()}_seq{seq_len}.keras
    CNN-LSTM:  cnn_lstm_{dataset.lower()}_seq{seq_len}.keras
    """
    if dataset is None:
        dataset = globals().get("DATASET", "FD001")
    if seq_len is None:
        seq_len = globals().get("SEQ_LEN", 30)
    if art_dir is None:
        art_dir = globals().get("ART_DIR", Path.cwd() / f"{dataset} data & artefacts")

    models_dir = Path(art_dir) / "models"
```

```python
    ds = dataset.lower()

    paths = {}
    if "Base" in include:
        paths["Base"] = str(models_dir / f"base_linear_{ds}_seq{seq_len}_{strategy}.joblib")
    if "LSTM" in include:
        paths["LSTM"] = str(models_dir / f"lstm_{ds}_seq{seq_len}.keras")
    if "CNN" in include:
        paths["CNN"] = str(models_dir / f"cnn_{ds}_seq{seq_len}.keras")
    if "CNN-LSTM" in include:
        paths["CNN-LSTM"] = str(models_dir / f"cnn_lstm_{ds}_seq{seq_len}.keras")
    return paths


def plot_true_vs_pred(y_true, y_pred, model_name="Model", savepath=None):
    """
    Scatter plot: True RUL vs Predicted RUL for a single model.

    Args:
        y_true (array-like): Ground truth RUL values.
        y_pred (array-like): Predicted RUL values.
        model_name (str): Name of the model for labeling.
        savepath (str or None): If given, save the figure to this path.
    """
    plt.figure(figsize=(6, 6))
    plt.scatter(y_true, y_pred, alpha=0.6, edgecolor="k")
    max_val = max(y_true.max(), y_pred.max())
    plt.plot([0, max_val], [0, max_val], "r--", lw=2, label="Ideal")
    plt.xlabel("True RUL")
    plt.ylabel("Predicted RUL")
    plt.title(f"True vs Predicted RUL ({model_name})")
    plt.legend()
    plt.grid(True)
    if savepath:
        plt.savefig(savepath, dpi=300, bbox_inches="tight")
    plt.show()



def plot_residuals(y_true, y_pred, model_name="Model", kind="box", savepath=None):
    """
    Plot residuals (Predicted - True) for a single model.
```

```python

    Args:
        y_true (array-like): Ground truth RUL values.
        y_pred (array-like): Predicted RUL values.
        model_name (str): Label for the model.
        kind (str): "box" or "violin" for plot type.
        savepath (str or None): If given, save the figure.
    """
    errors = y_pred - y_true
    plt.figure(figsize=(6, 4))

    if kind == "violin":
        sns.violinplot(y=errors)
    else:
        sns.boxplot(y=errors)

    plt.axhline(0, color="r", linestyle="--", lw=2, label="Zero Error")
    plt.ylabel("Residual (Predicted - True)")
    plt.title(f"Residual Distribution ({model_name})")
    plt.legend()
    plt.grid(True, axis="y")
    if savepath:
        plt.savefig(savepath, dpi=300, bbox_inches="tight")
    plt.show()

def plot_per_engine_bars(y_true, y_pred, unit_ids, model_name="Model", n_samples=30, savepath=None, seed=42):
    """
    Bar plot comparing Actual vs Predicted RUL for a random sample of engines.

    Args:
        y_true (array-like): Ground truth RUL values (aligned with unit_ids).
        y_pred (array-like): Predicted RUL values (aligned with unit_ids).
        unit_ids (array-like): Engine/unit identifiers for each sample.
        model_name (str): Name of the model for labeling.
        n_samples (int): Number of engines to randomly sample.
        savepath (str or None): If given, save the figure.
        seed (int): Random seed for reproducibility.
    """
    random.seed(seed)
    unique_ids = np.unique(unit_ids)
    chosen_ids = random.sample(list(unique_ids), min(n_samples, len(unique_ids)))

```

```python
    # Collect true & pred for chosen engines
    true_sample, pred_sample, labels = [], [], []
    for uid in chosen_ids:
        mask = unit_ids == uid
        # last entry corresponds to the test RUL label
        true_sample.append(y_true[mask][-1])
        pred_sample.append(y_pred[mask][-1])
        labels.append(str(uid))

    x = np.arange(len(chosen_ids))
    width = 0.35

    plt.figure(figsize=(12, 6))
    plt.bar(x - width/2, true_sample, width, label="Actual")
    plt.bar(x + width/2, pred_sample, width, label="Predicted")
    plt.xticks(x, labels, rotation=45)
    plt.xlabel("Engine ID (sampled)")
    plt.ylabel("RUL")
    plt.title(f"Actual vs Predicted RUL (Sampled Engines) - {model_name}")
    plt.legend()
    plt.tight_layout()
    if savepath:
        plt.savefig(savepath, dpi=300, bbox_inches="tight")
    plt.show()
def plot_metric_comparison(metrics_list,
                           dataset_name: str = "FD001",
                           savepath: Optional[str] = None):
    """
    Grouped bar chart of RMSE/MAE across models.
    Accepts either:
      - list of dicts: [{"model":"LSTM","RMSE":15.2,"MAE":11.3}, ...]
      - DataFrame with columns: model/Model, RMSE, MAE
    """
    # Build DataFrame
    df = metrics_list.copy() if isinstance(metrics_list, pd.DataFrame) else pd.DataFrame(metrics_list)
    if df.empty:
        print("No metrics to plot.")
        return

    # Normalise column names
    if "model" not in df.columns and "Model" in df.columns:
        df = df.rename(columns={"Model": "model"})
```

```

    # Validate required columns
    required = {"model", "RMSE", "MAE"}
    missing = required - set(df.columns)
    if missing:
        raise ValueError(f"Missing columns for plotting: {missing}")

    # Keep only what we need, coerce to numeric
    df = df[["model", "RMSE", "MAE"]].copy()
    df[["RMSE", "MAE"]] = df[["RMSE", "MAE"]].astype(float)
    df = df.set_index("model")

    # Plot
    ax = df.plot(kind="bar", figsize=(10, 6))
    ax.set_title(f"RMSE / MAE Comparison — {dataset_name}")
    ax.set_ylabel("Error")
    ax.set_xlabel("")
    ax.grid(True, axis="y", alpha=0.3)
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.show()
```

Project Module and base set up

In [3]:

```python
# Project modules
import data_loader as dl
import pre_processing as pp
import evaluator as ev
import base_model as base
import lstm_model as lstm
import cnn_model as cnn
import cnn_lstm_model as cnnlstm
import plots

# ---- Paths ----
ROOT = Path.cwd()
CMAPS = ROOT / "CMaps"  # keep correct folder case
# ==== Minimal config you tweak next time ====
DATASET = "FD002"       # <— change this to FD002/FD003/FD004 later
SEQ_LEN = 30            # sliding window
MAX_RUL = 130           # RUL clipping
VAL_SPLIT = 0.30        # val split by unit

# Files derived from DATASET (so you edit one line only)
TRAIN_PATH = CMAPS / f"train_{DATASET}.txt"
TEST_PATH  = CMAPS / f"test_{DATASET}.txt"
RUL_PATH   = CMAPS / f"RUL_{DATASET}.txt"

# Artifacts folder for this dataset
ART_DIR = ROOT / f"{DATASET} data & artefacts"
ART_DIR.mkdir(exist_ok=True)

print(f"backend: torch | dataset: {DATASET}")
print("Train:", TRAIN_PATH.name, "| Test:", TEST_PATH.name, "| RUL:", RUL_PATH.name)
```

```
backend: torch | dataset: FD002
Train: train_FD002.txt | Test: test_FD002.txt | RUL: RUL_FD002.txt
```

Load & Preprocessing Data

In [4]:

```python
# --- Load FD001 ---
train_df = dl.load_raw_data(CMAPS / f"train_{DATASET}.txt")
test_df, rul_df = dl.load_test_data(
    CMAPS / f"test_{DATASET}.txt",
    CMAPS / f"RUL_{DATASET}.txt"
)

unit_ids = test_df["unit_number"].values

print("Loaded.")
print("  train_df:", train_df.shape, "  test_df:", test_df.shape, "  rul_df:", rul_df.shape)
assert train_df.shape[1] == 26 and test_df.shape[1] == 26

dl.inspect_data(train_df)
pp.summarise_engine_lifespans(train_df, dataset_name=DATASET)

# ----------------------------
# 1) TRAIN: make targets first (no leakage)
# ----------------------------
train_rul = pp.calculate_rul(train_df, max_rul=MAX_RUL)

# ----------------------------
# 2) Split by unit BEFORE deciding features/scaling
# ----------------------------
train_split, val_split = pp.split_by_unit(train_rul, test_size=0.2, random_state=42)

# ----------------------------
# 3) Decide flat sensors using TRAIN ONLY, then drop same cols from val/test
# ----------------------------
before_cols = list(train_split.columns)
train_split_clean = pp.drop_flat_sensors(train_split.copy())
after_cols  = list(train_split_clean.columns)
dropped_cols = [c for c in before_cols if c not in after_cols]

val_split_clean = val_split.drop(columns=[c for c in dropped_cols if c in val_split.columns]).reset_index(drop=True)
test_df_clean   = test_df.drop(columns=[c for c in dropped_cols if c in test_df.columns]).reset_index(drop=True)

# ----------------------------
# 4) TEST: build true RUL from RUL file, then clip like train
# ----------------------------
last_cycles = test_df_clean.groupby("unit_number")["time_in_cycles"].max()
```

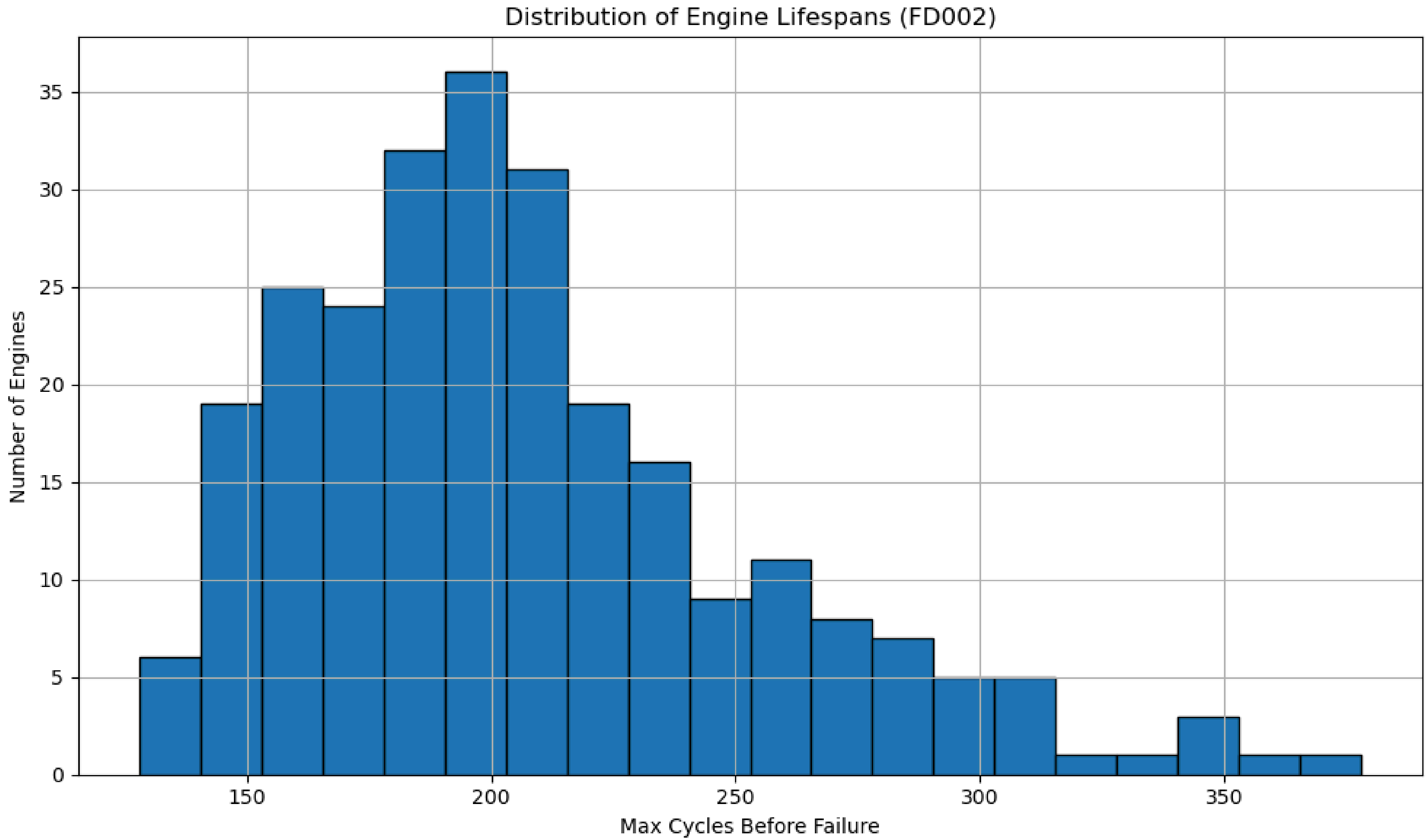
```python
rul_map = dict(zip(sorted(test_df_clean["unit_number"].unique()), rul_df["RUL"].values))
test_df_clean = test_df_clean.copy()
test_df_clean["RUL"] = test_df_clean.apply(
    lambda r: (last_cycles.loc[r["unit_number"]] - r["time_in_cycles"]) + rul_map[r["unit_number"]],
    axis=1
)
test_df_clean["RUL"] = np.minimum(test_df_clean["RUL"], MAX_RUL)

# ---------------------------
# 5) Scale sensors (per-condition for FD002/FD004, global for FD001/FD003)
# ---------------------------
from sklearn.preprocessing import StandardScaler

sensor_cols = [c for c in train_split_clean.columns if c.startswith("sensor_measurement")]
op_cols = [c for c in train_split_clean.columns
           if c.startswith("operational_setting") or c.startswith("op_setting")]

def _cond_keys(df, op_cols, ndigits=3):
    # Round op settings to make discrete condition keys robust to float jitter
    return list(map(tuple, df[op_cols].round(ndigits).to_numpy()))

def fit_per_condition_scalers(df, sensor_cols, op_cols, ndigits=3):
    df = df.copy()
    df["_cond_key"] = _cond_keys(df, op_cols, ndigits)
    scalers = {}
    for k, g in df.groupby("_cond_key", sort=False):
        scalers[k] = StandardScaler().fit(g[sensor_cols])
    return scalers

def transform_with_condition_scalers(df, sensor_cols, op_cols, scalers, fallback_scaler, ndigits=3):
    out = df.copy()
    out["_cond_key"] = _cond_keys(out, op_cols, ndigits)
    unseen_rows = 0
    for k, idx in out.groupby("_cond_key").groups.items():
        sc = scalers.get(k, fallback_scaler)
        if k not in scalers:
            unseen_rows += len(idx)
        out.loc[idx, sensor_cols] = sc.transform(out.loc[idx, sensor_cols])
    if unseen_rows:
        print(f"[WARN] {unseen_rows} rows used fallback scaler due to unseen condition(s).")
    return out.drop(columns=["_cond_key"])

```

```python
multi_condition = DATASET in {"FD002", "FD004"}

if multi_condition:
    # Fit: one scaler per operating condition on TRAIN ONLY
    cond_scalers   = fit_per_condition_scalers(train_split_clean, sensor_cols, op_cols, ndigits=3)
    global_scaler  = StandardScaler().fit(train_split_clean[sensor_cols])  # safe fallback
    print(f"Fitted {len(cond_scalers)} condition-specific scalers on training set.")

    # Apply the same condition scalers to train/val/test
    train_scaled = transform_with_condition_scalers(train_split_clean, sensor_cols, op_cols, cond_scalers, global_sc
    val_scaled   = transform_with_condition_scalers(val_split_clean,   sensor_cols, op_cols, cond_scalers, global_sc
    test_scaled  = transform_with_condition_scalers(test_df_clean,     sensor_cols, op_cols, cond_scalers, global_sc
else:
    # Single-condition datasets (FD001, FD003): one global scaler
    scaler = StandardScaler().fit(train_split_clean[sensor_cols])
    train_scaled = train_split_clean.copy()
    val_scaled   = val_split_clean.copy()
    test_scaled  = test_df_clean.copy()
    train_scaled[sensor_cols] = scaler.transform(train_scaled[sensor_cols])
    val_scaled[sensor_cols]   = scaler.transform(val_scaled[sensor_cols])
    test_scaled[sensor_cols]  = scaler.transform(test_scaled[sensor_cols])

# ---------------------------
# 6) Windowing with your helper
# ---------------------------
X_train, y_train = pp.generate_sliding_windows(train_scaled, seq_len=SEQ_LEN)
X_val,   y_val   = pp.generate_sliding_windows(val_scaled,   seq_len=SEQ_LEN)
X_test,  y_test  = pp.generate_sliding_windows(test_scaled,  seq_len=SEQ_LEN)

print("After preprocessing (FD001, no new pp funcs):")
print("  Train engines :", train_scaled['unit_number'].nunique())
print("  Val engines   :",   val_scaled['unit_number'].nunique())
print("  X_train shape :", X_train.shape, " y_train:", y_train.shape)
print("  X_val   shape :", X_val.shape,   " y_val  :", y_val.shape)
print("  X_test  shape :", X_test.shape,  " y_test :", y_test.shape)
print("  Dropped sensors:", dropped_cols)

# --- Save to use in model ---
out_npz = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
pp.save_preprocessed_data(X_train, y_train, X_val, y_val, X_test, y_test, filename=str(out_npz))
```

```
Loaded.
  train_df: (53759, 26)   test_df: (33991, 26)   rul_df: (259, 1)
Shape: (53759, 26)

Unique engines: 260

Missing values:
 0

Max cycles per engine:
count    260.000000
mean     206.765385
std       46.782198
min      128.000000
25%      174.000000
50%      199.000000
75%      230.250000
max      378.000000
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	34.9983	0.8400	100.0	449.44	555.32	1358.61	
1	1	2	41.9982	0.8408	100.0	445.00	549.90	1353.22	
2	1	3	24.9988	0.6218	60.0	462.54	537.31	1256.76	
3	1	4	42.0077	0.8416	100.0	445.00	549.51	1354.03	
4	1	5	25.0005	0.6203	60.0	462.54	537.07	1257.71	

5 rows × 26 columns


Distribution of Engine Lifespans (FD002)
Number of Engines
Max Cycles Before Failure
0
5
10
15
20
25
30
35
150
200
250
300
350

```
Dataset: FD002
Mean cycles to failure: 206.77
Standard deviation: 46.78
Minimum: 128
Maximum: 378
Fitted 177 condition-specific scalers on training set.
After preprocessing (FD001, no new pp funcs):
  Train engines : 208
  Val engines   : 52
  X_train shape : (37432, 30, 21)  y_train: (37432,)
  X_val   shape : (8787, 30, 21)  y_val  : (8787,)
  X_test  shape : (26505, 30, 21)  y_test : (26505,)
  Dropped sensors: []
Data saved to C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD002 data & artefacts\fd002_seq30.npz
```

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

Base Model

In [5]:
```
from base_model import train_linear_model

# 1) Load cached windows
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
X_train, y_train, X_val, y_val, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

# 2) Convert 3D windows -> 2D feature vectors for baseline
#    (stick to your existing helper; default strategy='last')
X_train_feat = pp.make_feature_vectors_from_windows(X_train, strategy='last')
X_val_feat   = pp.make_feature_vectors_from_windows(X_val,   strategy='last')

print("Feature shapes (base model):")
print("  X_train:", X_train_feat.shape, " y_train:", y_train.shape)
print("  X_val  :", X_val_feat.shape,   " y_val  :", y_val.shape)

# 3) Train simple Linear Regression
base_model = train_linear_model(X_train_feat, y_train)

# 4) Save the trained model
models_dir = ART_DIR / "models"
models_dir.mkdir(parents=True, exist_ok=True)

model_path = models_dir / f"base_linear_{DATASET.lower()}_seq{SEQ_LEN}_last.joblib"
joblib.dump(base_model, model_path)

print("Saved base model to:", model_path)
```

```
Feature shapes (base model):
  X_train: (37432, 21)  y_train: (37432,)
  X_val  : (8787, 21)  y_val  : (8787,)
Saved base model to: C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD002 data & artefacts\models\base_linear
_fd002_seq30_last.joblib
```

CNN Model

In [6]:

```
# --- Train / Save: CNN (FD001) ---

from cnn_model import build_cnn_model, train_cnn_model
import json

# 1) Load cached 3D windows
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
X_train, y_train, X_val, y_val, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

# 2) Build CNN
input_shape = (SEQ_LEN, X_train.shape[2])
cnn = build_cnn_model(input_shape)

# 3) Train (only pass what your module expects!)
cnn, history = train_cnn_model(cnn, X_train, y_train, X_val, y_val, epochs=30, batch_size=128)

# 4) Save model + meta
models_dir = ART_DIR / "models"
models_dir.mkdir(parents=True, exist_ok=True)
model_path = models_dir / f"cnn_{DATASET.lower()}_seq{SEQ_LEN}.keras"
cnn.save(model_path)

meta = {
    "model_type": "cnn",
    "dataset": DATASET,
    "seq_len": int(SEQ_LEN),
    "features": int(X_train.shape[2]),
    "epochs": 30,
    "batch_size": 128
}
with open(models_dir / f"cnn_{DATASET.lower()}_seq{SEQ_LEN}.meta.json", "w") as f:
    json.dump(meta, f, indent=2)

print("Saved CNN model to:", model_path)
```

```
Epoch 1/30
293/293 [==============================] - 3s 7ms/step - loss: 1302.1160 - val_loss: 574.4021
Epoch 2/30
293/293 [==============================] - 2s 7ms/step - loss: 594.6576 - val_loss: 463.9736
Epoch 3/30
293/293 [==============================] - 2s 6ms/step - loss: 493.7151 - val_loss: 466.8395
Epoch 4/30
293/293 [==============================] - 2s 6ms/step - loss: 421.1211 - val_loss: 459.9334
Epoch 5/30
293/293 [==============================] - 2s 6ms/step - loss: 355.0642 - val_loss: 431.8060
Epoch 6/30
293/293 [==============================] - 2s 6ms/step - loss: 309.4612 - val_loss: 430.1591
Epoch 7/30
293/293 [==============================] - 2s 6ms/step - loss: 280.3728 - val_loss: 407.6515
Epoch 8/30
293/293 [==============================] - 2s 7ms/step - loss: 257.5421 - val_loss: 417.8796
Epoch 9/30
293/293 [==============================] - 2s 6ms/step - loss: 240.1156 - val_loss: 431.1063
Epoch 10/30
293/293 [==============================] - 2s 6ms/step - loss: 226.5463 - val_loss: 434.2556
Epoch 11/30
293/293 [==============================] - 2s 6ms/step - loss: 209.3753 - val_loss: 444.0627
Epoch 12/30
293/293 [==============================] - 2s 6ms/step - loss: 200.7879 - val_loss: 525.7309
Epoch 13/30
293/293 [==============================] - 2s 6ms/step - loss: 191.7791 - val_loss: 452.9439
Epoch 14/30
293/293 [==============================] - 2s 6ms/step - loss: 179.0186 - val_loss: 457.2694
Epoch 15/30
293/293 [==============================] - 2s 6ms/step - loss: 177.1577 - val_loss: 449.0180
Epoch 16/30
293/293 [==============================] - 2s 6ms/step - loss: 172.5721 - val_loss: 471.7536
Epoch 17/30
285/293 [============================>.] - ETA: 0s - loss: 165.7126Restoring model weights from the end of the best epo
ch: 7.
293/293 [==============================] - 2s 6ms/step - loss: 166.1518 - val_loss: 465.6028
Epoch 17: early stopping
Saved CNN model to: C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD002 data & artefacts\models\cnn_fd002_se
q30.keras
```

LSTM MODEL

In [7]:

```
# --- LSTM model (train + save) ---

from lstm_model import build_lstm_model, train_lstm_model
import json

# 1) Load cached 3D windows
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
X_train, y_train, X_val, y_val, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

# 2) Build LSTM (input: [seq_len, n_features])
input_shape = (SEQ_LEN, X_train.shape[2])
lstm = build_lstm_model(input_shape)

# 3) Train (pass only what your module expects)
#    If your train_lstm_model returns (model, history), keep both; if it returns only model, handle that too.
result = train_lstm_model(lstm, X_train, y_train, X_val, y_val, epochs=50, batch_size=128)
if isinstance(result, tuple):
    lstm, lstm_history = result
else:
    lstm = result
    lstm_history = None

# 4) Save model + minimal metadata
models_dir = ART_DIR / "models"
models_dir.mkdir(parents=True, exist_ok=True)

model_path = models_dir / f"lstm_{DATASET.lower()}_seq{SEQ_LEN}.keras"
lstm.save(model_path)

meta = {
    "model_type": "lstm",
    "dataset": DATASET,
    "seq_len": int(SEQ_LEN),
    "features": int(X_train.shape[2]),
    "epochs": 50,
    "batch_size": 128
}
with open(models_dir / f"lstm_{DATASET.lower()}_seq{SEQ_LEN}.meta.json", "w") as f:
    json.dump(meta, f, indent=2)

```

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

```
Epoch 1/50
293/293 [==============================] - 10s 29ms/step - loss: 6867.7446 - val_loss: 5633.7949
Epoch 2/50
293/293 [==============================] - 7s 23ms/step - loss: 5464.1064 - val_loss: 4556.5864
Epoch 3/50
293/293 [==============================] - 7s 23ms/step - loss: 4447.6431 - val_loss: 3690.5579
Epoch 4/50
293/293 [==============================] - 7s 23ms/step - loss: 3622.3079 - val_loss: 2978.9644
Epoch 5/50
293/293 [==============================] - 7s 24ms/step - loss: 2942.3350 - val_loss: 2399.4915
Epoch 6/50
293/293 [==============================] - 7s 24ms/step - loss: 2382.5227 - val_loss: 1930.6628
Epoch 7/50
293/293 [==============================] - 7s 24ms/step - loss: 1923.9257 - val_loss: 1547.6835
Epoch 8/50
293/293 [==============================] - 7s 24ms/step - loss: 1550.2682 - val_loss: 1240.2026
Epoch 9/50
293/293 [==============================] - 7s 24ms/step - loss: 1249.9899 - val_loss: 1007.3118
Epoch 10/50
293/293 [==============================] - 7s 24ms/step - loss: 1010.3068 - val_loss: 827.0479
Epoch 11/50
293/293 [==============================] - 7s 24ms/step - loss: 819.1207 - val_loss: 673.8647
Epoch 12/50
293/293 [==============================] - 7s 24ms/step - loss: 672.0478 - val_loss: 584.8643
Epoch 13/50
293/293 [==============================] - 7s 25ms/step - loss: 557.7310 - val_loss: 482.5157
Epoch 14/50
293/293 [==============================] - 7s 24ms/step - loss: 468.0135 - val_loss: 436.0164
Epoch 15/50
293/293 [==============================] - 7s 25ms/step - loss: 399.2530 - val_loss: 402.4728
Epoch 16/50
293/293 [==============================] - 7s 25ms/step - loss: 346.8802 - val_loss: 379.9492
Epoch 17/50
293/293 [==============================] - 7s 25ms/step - loss: 302.6385 - val_loss: 353.8716
Epoch 18/50
293/293 [==============================] - 7s 25ms/step - loss: 265.1220 - val_loss: 369.9253
Epoch 19/50
293/293 [==============================] - 7s 25ms/step - loss: 237.8524 - val_loss: 369.2892
Epoch 20/50
293/293 [==============================] - 7s 25ms/step - loss: 215.6219 - val_loss: 327.9399
Epoch 21/50
```

```
293/293 [==============================] - 8s 26ms/step - loss: 192.1157 - val_loss: 369.8544
Epoch 22/50
293/293 [==============================] - 7s 25ms/step - loss: 178.1889 - val_loss: 325.7454
Epoch 23/50
293/293 [==============================] - 7s 25ms/step - loss: 165.9228 - val_loss: 372.2390
Epoch 24/50
293/293 [==============================] - 7s 25ms/step - loss: 154.8340 - val_loss: 375.3578
Epoch 25/50
293/293 [==============================] - 8s 26ms/step - loss: 147.8136 - val_loss: 402.7542
Epoch 26/50
293/293 [==============================] - 8s 26ms/step - loss: 138.8945 - val_loss: 368.7269
Epoch 27/50
293/293 [==============================] - 8s 26ms/step - loss: 133.9885 - val_loss: 374.7237
Epoch 28/50
293/293 [==============================] - 8s 26ms/step - loss: 126.7215 - val_loss: 410.9928
Epoch 29/50
293/293 [==============================] - 8s 27ms/step - loss: 123.9504 - val_loss: 386.5808
Epoch 30/50
293/293 [==============================] - 7s 26ms/step - loss: 118.0515 - val_loss: 374.7599
Epoch 31/50
293/293 [==============================] - 8s 26ms/step - loss: 116.0033 - val_loss: 426.8628
Epoch 32/50
293/293 [==============================] - 8s 26ms/step - loss: 111.4865 - val_loss: 417.8333
Saved LSTM model to: C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD002 data & artefacts\models\lstm_fd002_
seq30.keras
```

CNN_LSTM Model

In [8]:

```python
# --- CNN-LSTM model (train + save) ---

from cnn_lstm_model import build_cnn_lstm_model, train_cnn_lstm_model
import json

# 1) Load cached 3D windows
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
X_train, y_train, X_val, y_val, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

# 2) Build model (input: [seq_len, n_features])
input_shape = (SEQ_LEN, X_train.shape[2])
cnnlstm = build_cnn_lstm_model(input_shape)

# 3) Train (handle: returns History, returns Model, or returns (Model, History))
train_ret = train_cnn_lstm_model(
    cnnlstm, X_train, y_train, X_val, y_val,
    epochs=50, batch_size=128
)

# Normalize outputs
cnnlstm_history = None
model_to_save = cnnlstm  # fallback: the model we built

# Case A: (model, history)
if isinstance(train_ret, tuple) and len(train_ret) >= 1:
    if hasattr(train_ret[0], "save"):
        model_to_save = train_ret[0]
    if len(train_ret) >= 2 and hasattr(train_ret[1], "history"):
        cnnlstm_history = train_ret[1]

# Case B: just a model
elif hasattr(train_ret, "save"):
    model_to_save = train_ret

# Case C: just a History
elif hasattr(train_ret, "history"):
    cnnlstm_history = train_ret

# 4) Save model + minimal metadata
models_dir = ART_DIR / "models"
models_dir.mkdir(parents=True, exist_ok=True)
```

```python

model_path = models_dir / f"cnn_lstm_{DATASET.lower()}_seq{SEQ_LEN}.keras"
model_to_save.save(model_path)

meta = {
    "model_type": "cnn_lstm",
    "dataset": DATASET,
    "seq_len": int(SEQ_LEN),
    "features": int(X_train.shape[2]),
    "epochs": 50,          # keep in sync with your fit(...)
    "batch_size": 128      # keep in sync with your fit(...)
}
with open(models_dir / f"cnn_lstm_{DATASET.lower()}_seq{SEQ_LEN}.meta.json", "w") as f:
    json.dump(meta, f, indent=2)

# (Optional) persist training history if available
if cnnlstm_history:
    hist_path = models_dir / f"cnn_lstm_{DATASET.lower()}_seq{SEQ_LEN}.history.json"
    with open(hist_path, "w") as f:
        json.dump(cnnlstm_history.history, f, indent=2)


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

```
Epoch 1/50
293/293 [==============================] - 7s 18ms/step - loss: 2895.8706 - mae: 40.4937 - val_loss: 498.3223 - val_ma
e: 18.3105
Epoch 2/50
293/293 [==============================] - 5s 16ms/step - loss: 532.2922 - mae: 17.6908 - val_loss: 495.2248 - val_mae:
16.9075
Epoch 3/50
293/293 [==============================] - 5s 16ms/step - loss: 396.6046 - mae: 15.0917 - val_loss: 453.8861 - val_mae:
16.3660
Epoch 4/50
293/293 [==============================] - 5s 16ms/step - loss: 331.9761 - mae: 13.8263 - val_loss: 512.7544 - val_mae:
17.1792
Epoch 5/50
293/293 [==============================] - 5s 16ms/step - loss: 290.2196 - mae: 12.9465 - val_loss: 447.5881 - val_mae:
15.6119
Epoch 6/50
293/293 [==============================] - 5s 16ms/step - loss: 273.0599 - mae: 12.5368 - val_loss: 498.7149 - val_mae:
16.6896
Epoch 7/50
293/293 [==============================] - 5s 16ms/step - loss: 252.7054 - mae: 12.1254 - val_loss: 491.1961 - val_mae:
15.5206
Epoch 8/50
293/293 [==============================] - 5s 16ms/step - loss: 241.1153 - mae: 11.7866 - val_loss: 482.3051 - val_mae:
15.7484
Epoch 9/50
293/293 [==============================] - 5s 16ms/step - loss: 231.9801 - mae: 11.5771 - val_loss: 479.2918 - val_mae:
16.2059
Epoch 10/50
293/293 [==============================] - 5s 17ms/step - loss: 228.8225 - mae: 11.4944 - val_loss: 459.5554 - val_mae:
15.6177
Epoch 11/50
293/293 [==============================] - 5s 18ms/step - loss: 220.4808 - mae: 11.2964 - val_loss: 514.9454 - val_mae:
16.1148
Epoch 12/50
293/293 [==============================] - 5s 17ms/step - loss: 215.6837 - mae: 11.1305 - val_loss: 516.7359 - val_mae:
16.8580
Epoch 13/50
293/293 [==============================] - 5s 17ms/step - loss: 213.7494 - mae: 11.1008 - val_loss: 486.5841 - val_mae:
16.0323
Epoch 14/50
293/293 [==============================] - 5s 17ms/step - loss: 210.4710 - mae: 10.9711 - val_loss: 524.3619 - val_mae:
```

```
16.7954
Epoch 15/50
293/293 [==============================] - 5s 17ms/step - loss: 209.8484 - mae: 10.9738 - val_loss: 508.4923 - val_mae: 16.4439
Saved CNN-LSTM model to: C:\Users\mg020649\Documents\15 - Coding\Msc-Project-main\FD002 data & artefacts\models\cnn_lstm_fd002_seq30.keras
```

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

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

LOAD MODEL

In [9]:
```python
# LOAD baseline MODEL
model_paths = build_model_paths()  # will only load what exists
print("Will load:", model_paths)

models = load_models(model_paths)
print("Loaded models:", list(models.keys()))
```

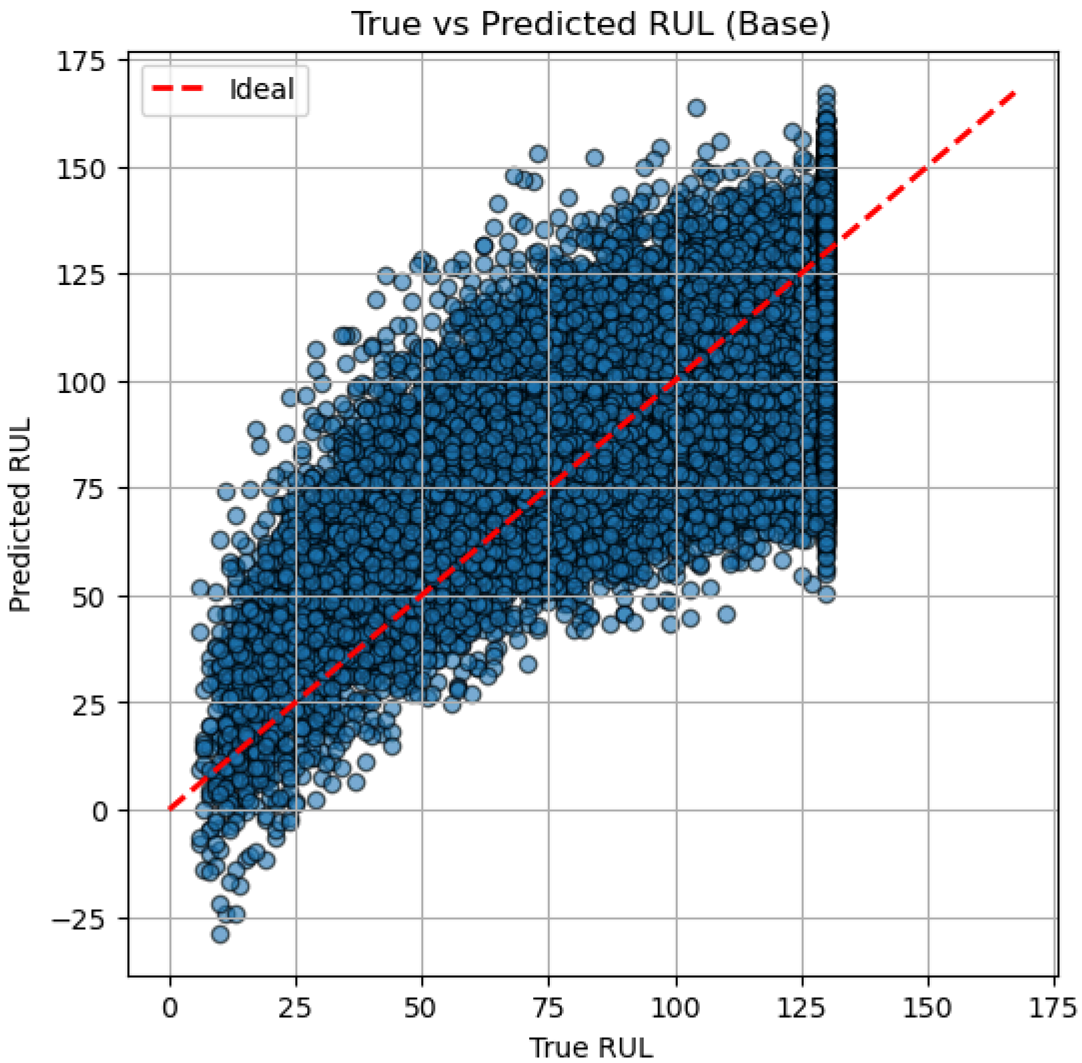
```
Will load: {'Base': 'C:\\Users\\mg020649\\Documents\\15 - Coding\\Msc-Project-main\\FD002 data & artefacts\\models\\base_linear_fd002_seq30_last.joblib', 'LSTM': 'C:\\Users\\mg020649\\Documents\\15 - Coding\\Msc-Project-main\\FD002 data & artefacts\\models\\lstm_fd002_seq30.keras', 'CNN': 'C:\\Users\\mg020649\\Documents\\15 - Coding\\Msc-Project-main\\FD002 data & artefacts\\models\\cnn_fd002_seq30.keras', 'CNN-LSTM': 'C:\\Users\\mg020649\\Documents\\15 - Coding\\Msc-Project-main\\FD002 data & artefacts\\models\\cnn_lstm_fd002_seq30.keras'}
Loaded models: ['Base', 'LSTM', 'CNN', 'CNN-LSTM']
```

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

Base build Analysis

In [10]:

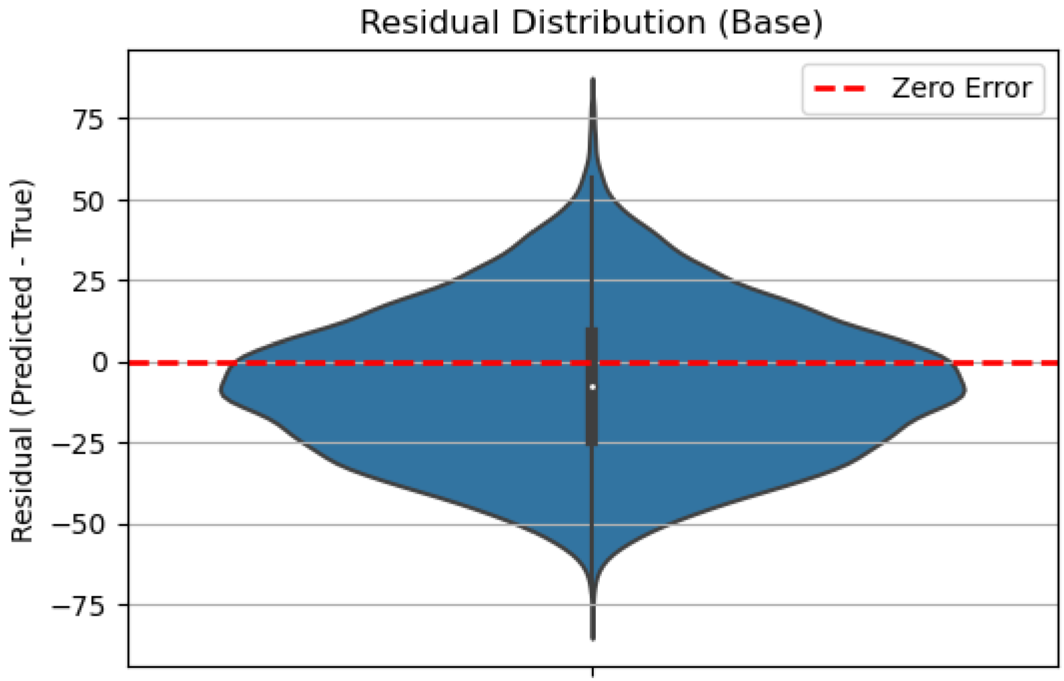
```
# plot_true_vs_pred

npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"

_, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

X_test_base = pp.make_feature_vectors_from_windows(X_test, strategy="last")

y_pred_base = models["Base"].predict(X_test_base).ravel()

y_true = y_test

plot_true_vs_pred(y_true, y_pred_base, model_name="Base")
```



True vs Predicted RUL (Base)
Ideal
Predicted RUL
True RUL
175
150
125
100
75
50
25
0
−25
0
25
50
75
100
125
150
175

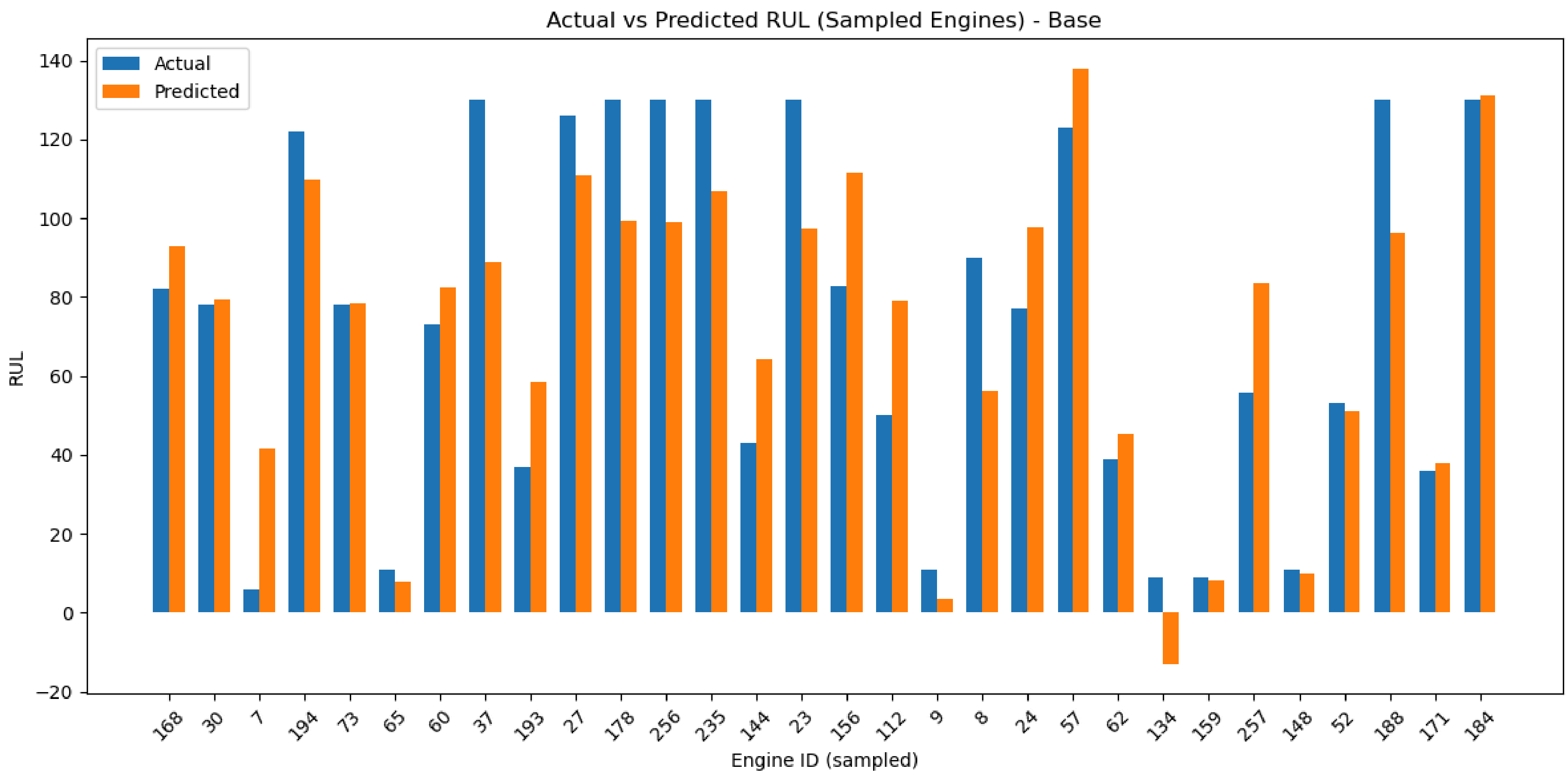
In [11]:

```
# Now plot residuals
plot_residuals(y_true, y_pred_base, model_name="Base", kind="violin")
```



In [12]:

```
# Build unit IDs aligned to the windowed test data (X_test, y_test)
ids = []
for uid, grp in test_scaled.groupby("unit_number"):
    grp = grp.sort_values("time_in_cycles")
    n = len(grp) - SEQ_LEN + 1
    if n > 0:
        ids.extend([uid] * n)
unit_ids_aligned = np.array(ids)

# Now plot
plot_per_engine_bars(y_true, y_pred_base, unit_ids_aligned, model_name="Base")

```



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

CNN Model Analysis

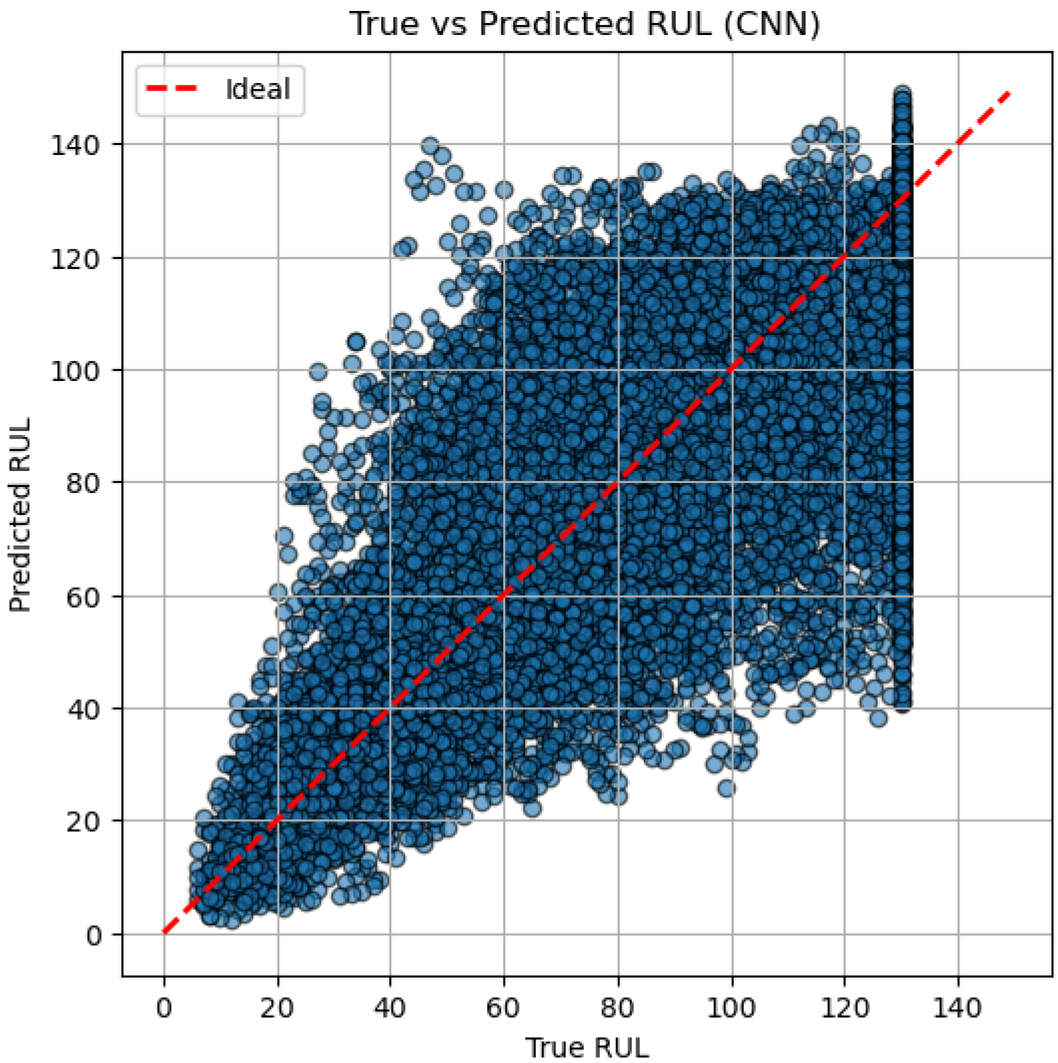
In [13]:

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

```
829/829 [==============================] - 1s 994us/step
```

In [14]:

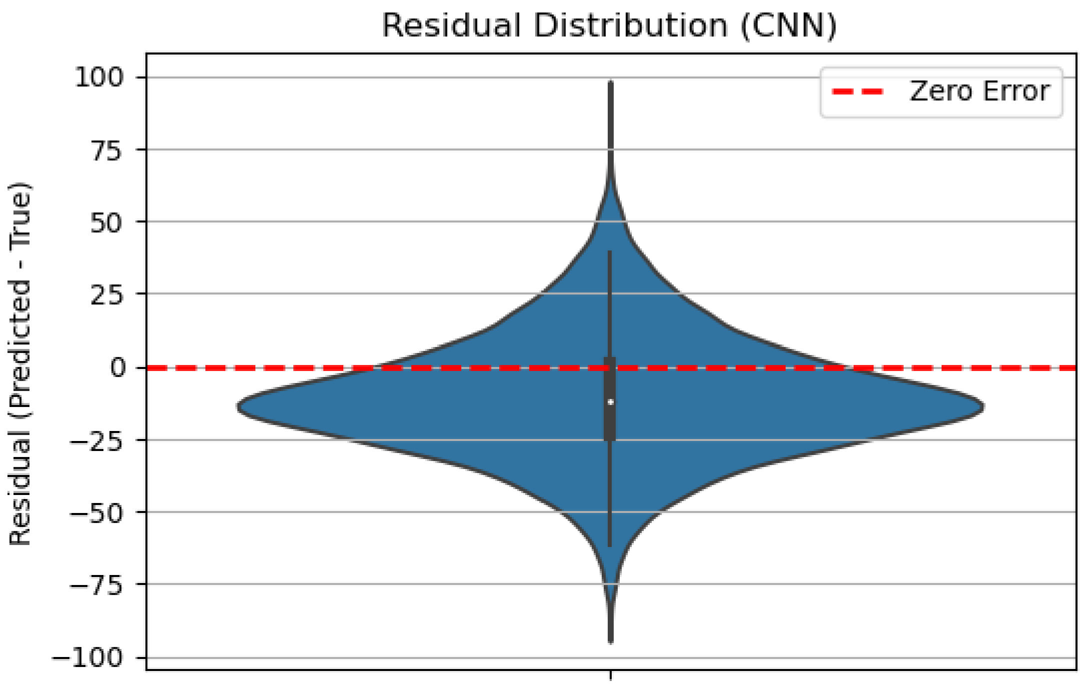
```
# plot_true_vs_pred

# can be rid, brought as don't want to run model build code again
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
_, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

y_true = y_test

plot_true_vs_pred(y_true, y_pred_cnn, model_name="CNN")
```



True vs Predicted RUL (CNN)
Ideal
Predicted RUL
True RUL
0
20
40
60
80
100
120
140

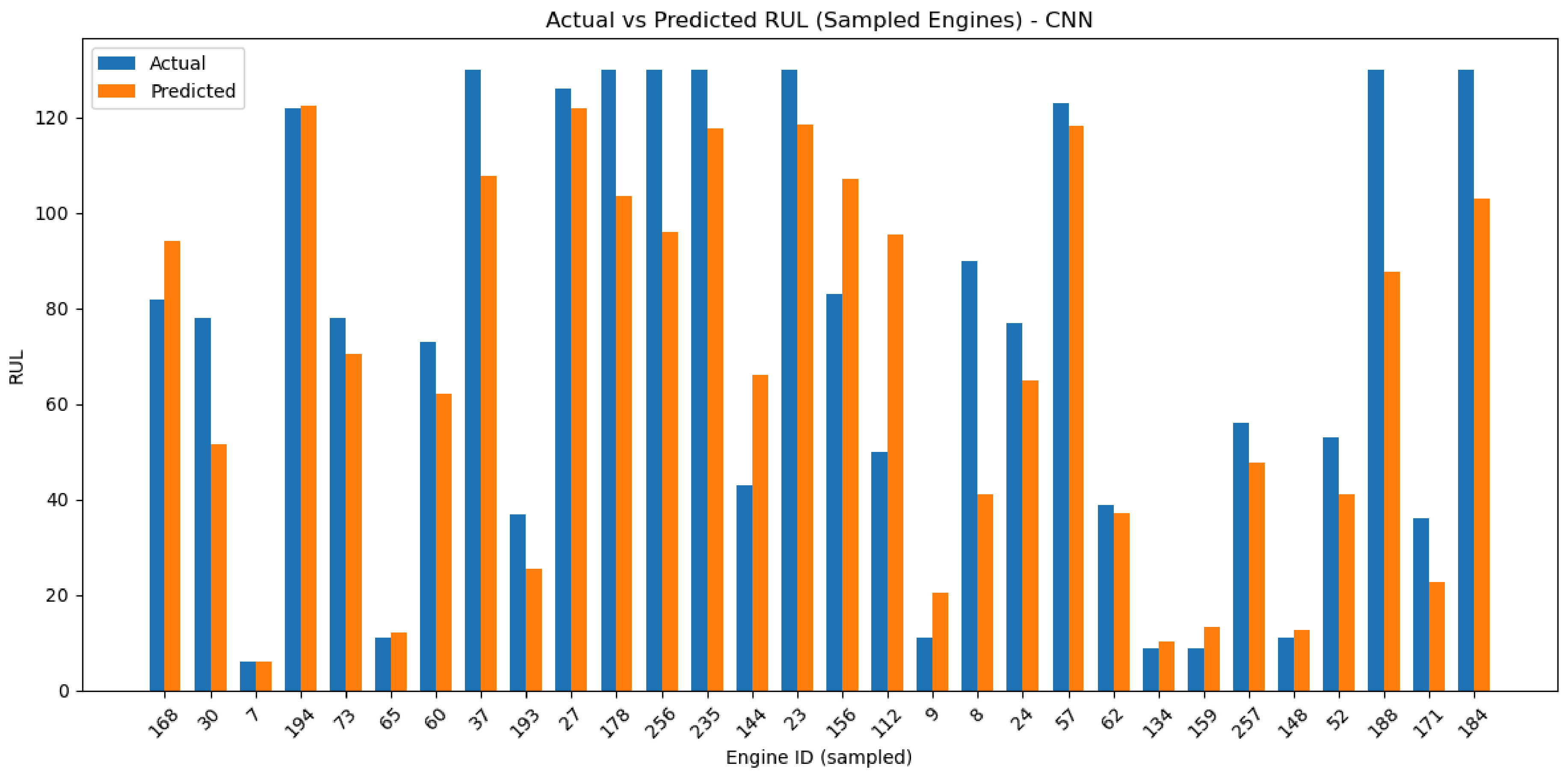
In [15]:

```
plot_residuals(y_true, y_pred_cnn, model_name="CNN", kind="violin")
```



In [16]:

```
# Build unit IDs aligned to the windowed test data (X_test, y_test)
ids = []
for uid, grp in test_scaled.groupby("unit_number"):
    grp = grp.sort_values("time_in_cycles")
    n = len(grp) - SEQ_LEN + 1
    if n > 0:
        ids.extend([uid] * n)
unit_ids_aligned = np.array(ids)

# Now plot
plot_per_engine_bars(y_true, y_pred_cnn, unit_ids_aligned, model_name="CNN")
```



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

LSTM ANALYSIS

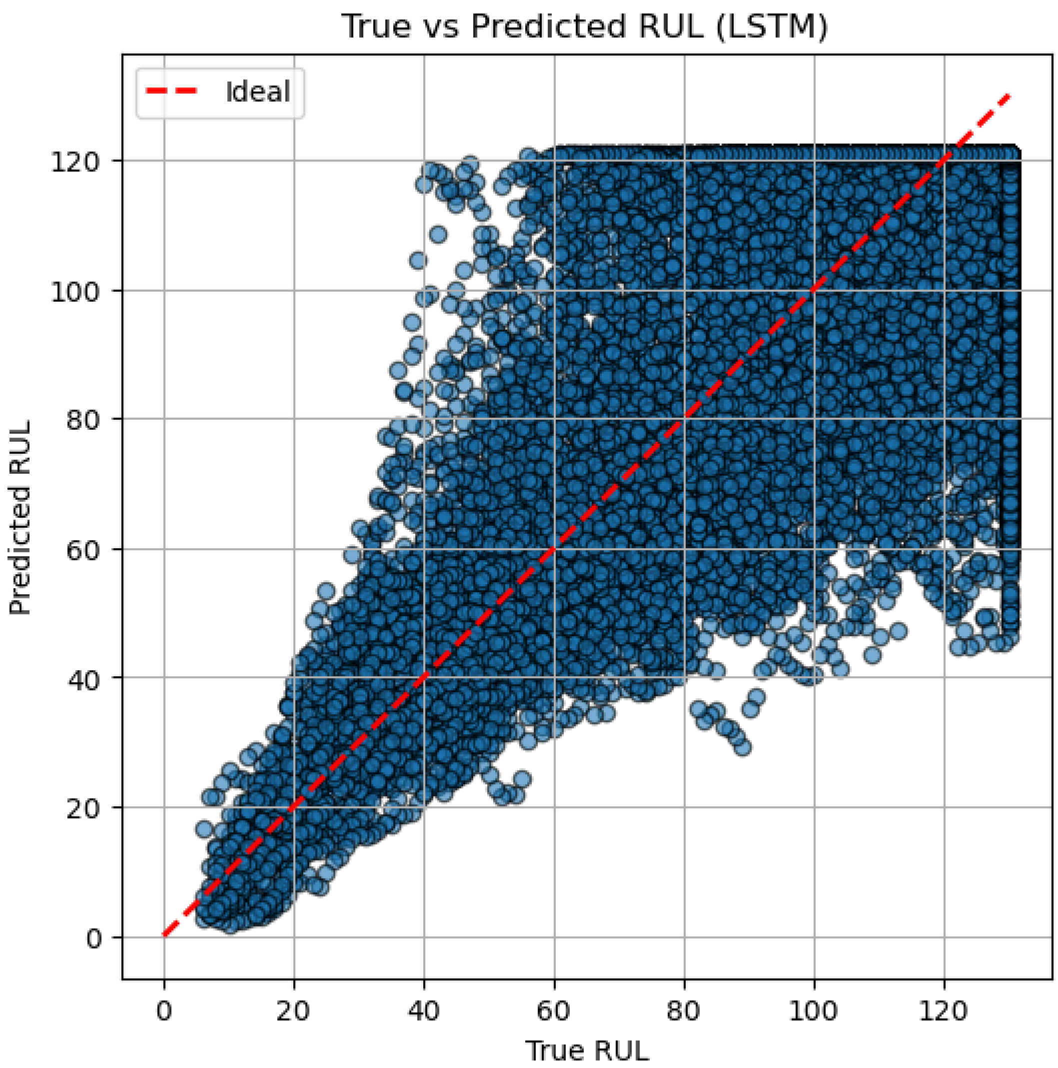
In [17]:

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

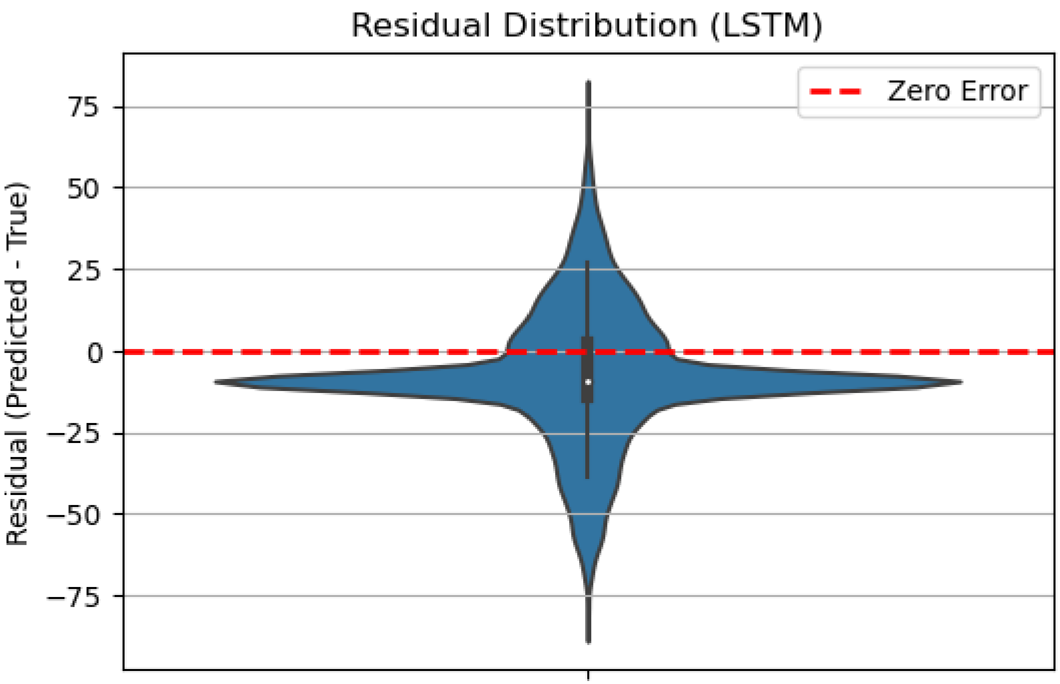
```
829/829 [==============================] - 4s 4ms/step
```

In [18]:

```
# plot_true_vs_pred

# can be rid, brought as don't want to run model build code again
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
_, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

y_true = y_test
plot_true_vs_pred(y_true, y_pred_lstm, model_name="LSTM")
```

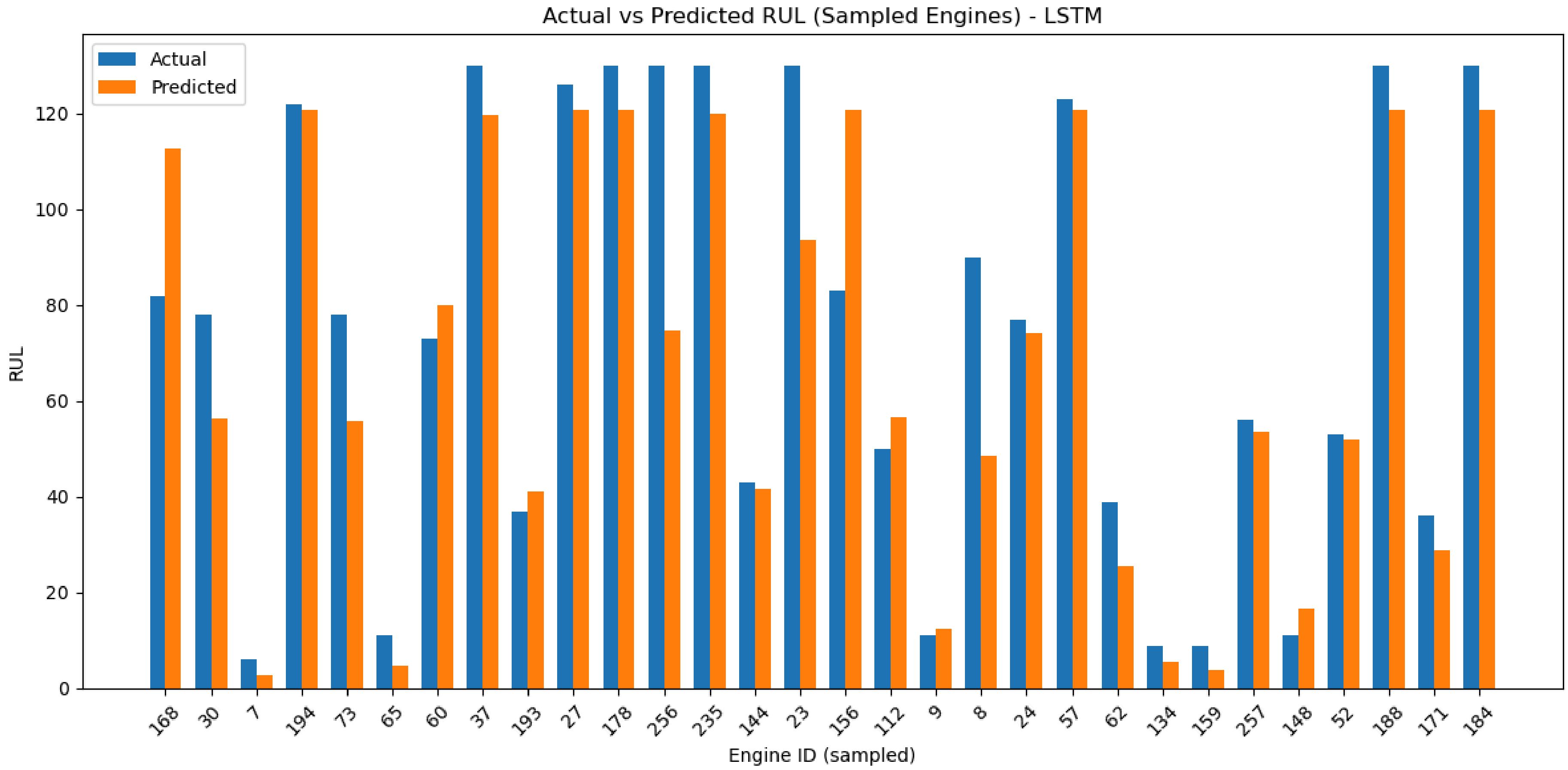


In [19]:
```
plot_residuals(y_true, y_pred_lstm, model_name="LSTM", kind="violin")
```



In [20]:

```
# Build unit IDs aligned to the windowed test data (X_test, y_test)
ids = []
for uid, grp in test_scaled.groupby("unit_number"):
    grp = grp.sort_values("time_in_cycles")
    n = len(grp) - SEQ_LEN + 1
    if n > 0:
        ids.extend([uid] * n)
unit_ids_aligned = np.array(ids)

# Now plot
plot_per_engine_bars(y_true, y_pred_lstm, unit_ids_aligned, model_name="LSTM")

```



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

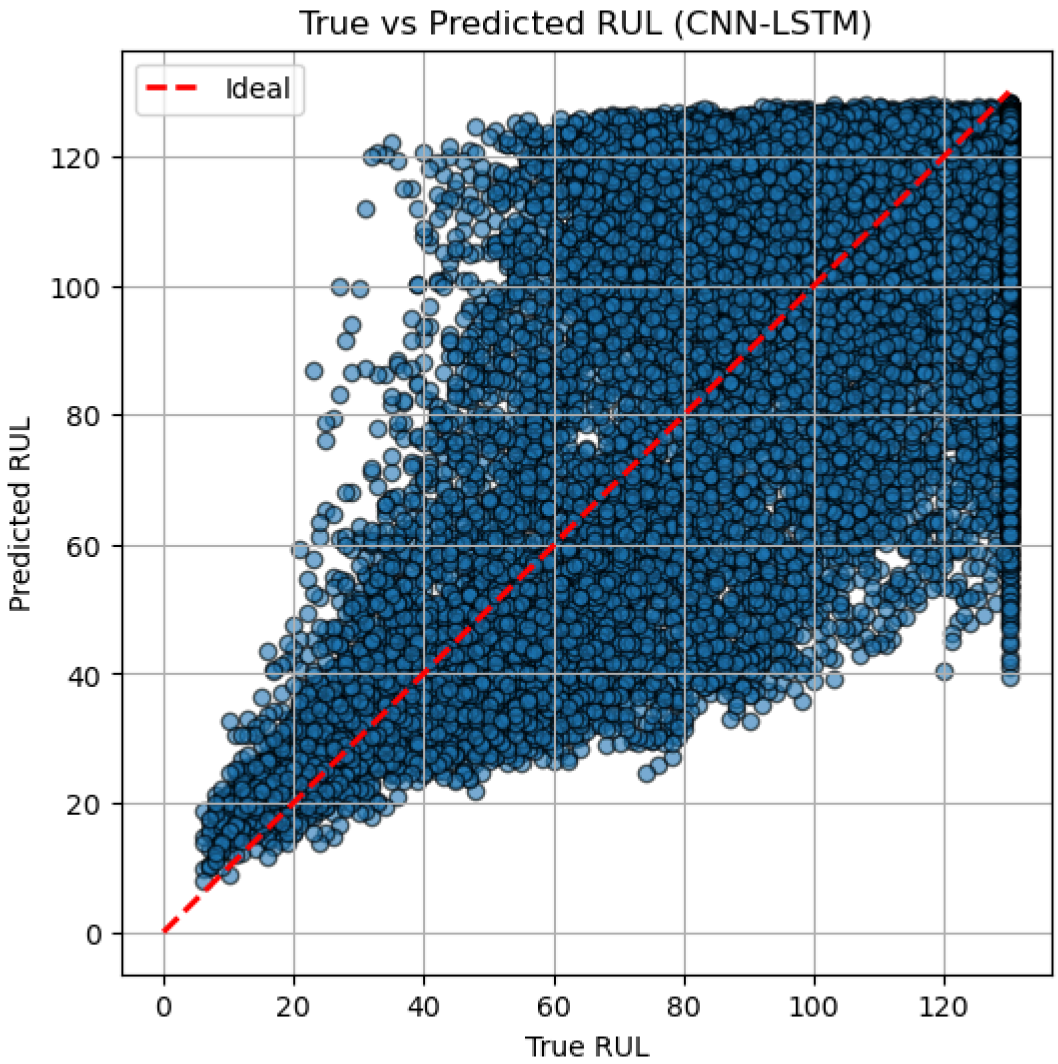
LSTM_CNN Build Analysis

In [21]:

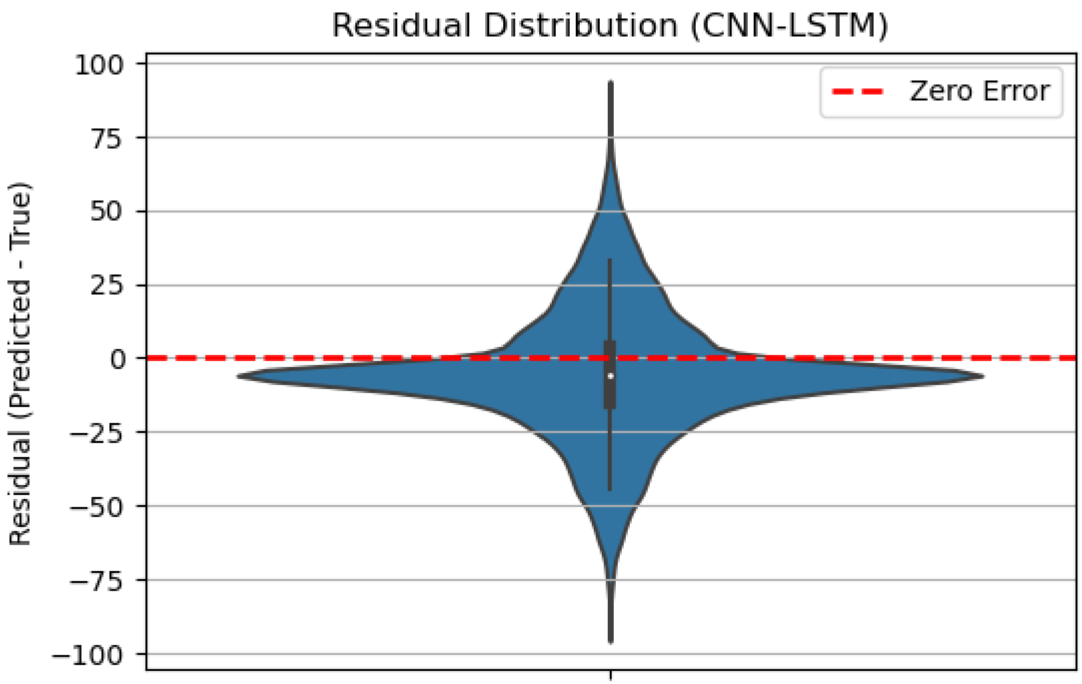
```
# CNN Prediction
from cnn_lstm_model import predict_cnn_lstm_model
y_pred_cnn_lstm = predict_cnn_lstm_model(models["CNN-LSTM"], X_test)
```

In [22]:

```
# plot_true_vs_pred

# can be rid, brought as don't want to run model build code again
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
_, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

y_true = y_test
plot_true_vs_pred(y_true, y_pred_cnn_lstm, model_name="CNN-LSTM")
```

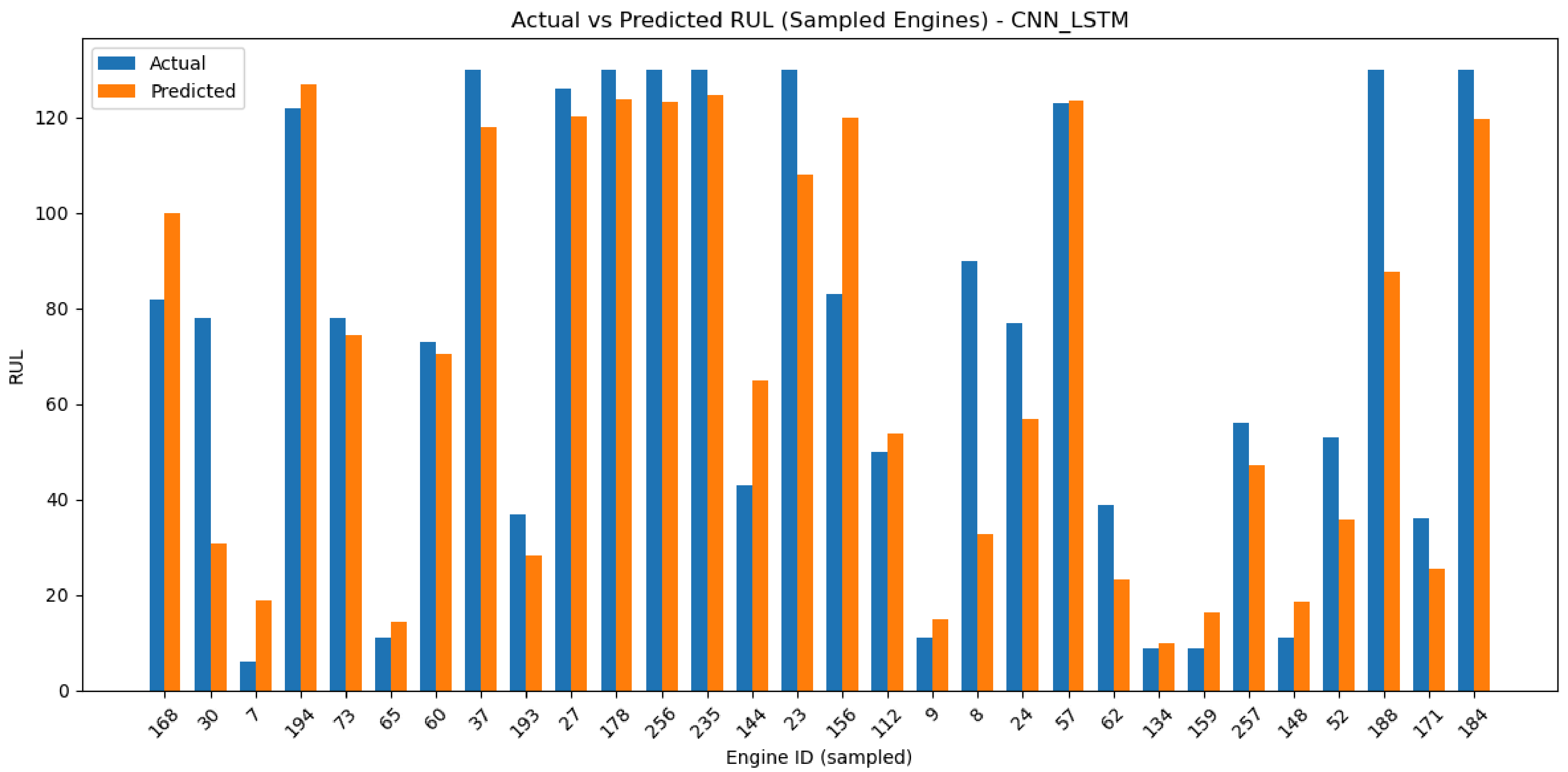


In [23]:
```
plot_residuals(y_true, y_pred_cnn_lstm, model_name="CNN-LSTM", kind="violin")
```



In [24]:

```
# Build unit IDs aligned to the windowed test data (X_test, y_test)
ids = []
for uid, grp in test_scaled.groupby("unit_number"):
    grp = grp.sort_values("time_in_cycles")
    n = len(grp) - SEQ_LEN + 1
    if n > 0:
        ids.extend([uid] * n)
unit_ids_aligned = np.array(ids)

# Now plot
plot_per_engine_bars(y_true, y_pred_cnn_lstm, unit_ids_aligned, model_name="CNN_LSTM")
```



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

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

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

base model

In [25]:

```python
from evaluator import evaluate_model

# load preprocessed test set
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
_, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

# convert if using base model (needs 2D vectors)
X_test_feat = pp.make_feature_vectors_from_windows(X_test, strategy="last")

# load trained base model
model_path = ART_DIR / "models" / f"base_linear_{DATASET.lower()}_seq{SEQ_LEN}_last.joblib"
base_model = joblib.load(model_path)

# predict & evaluate
y_base_pred = base_model.predict(X_test_feat)
evaluate_model(y_test, y_base_pred, model_name="Base Linear Model")
```

```
Base Linear Model Evaluation:
  RMSE: 24.2948
  MAE : 19.7025
```

Out[25]:

```
{'model': 'Base Linear Model', 'RMSE': 24.294798, 'MAE': 19.702452}
```

CNN model

In [26]:

```python
# --- Predict & evaluate: CNN on FD001 test ---

from evaluator import evaluate_model

# 1) Load cached test windows
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
_, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

# 2) Load saved CNN
cnn_path = ART_DIR / "models" / f"cnn_{DATASET.lower()}_seq{SEQ_LEN}.keras"
cnn_model = load_model(cnn_path)

# 3) Predict (CNN expects 3D windows)
y_cnn_pred = cnn_model.predict(X_test, verbose=0).squeeze()

# 4) Evaluate
evaluate_model(y_test, y_cnn_pred, model_name="CNN")
```

```
CNN Evaluation:
  RMSE: 24.2322
  MAE : 19.5151
```

Out[26]:

```
{'model': 'CNN', 'RMSE': 24.232237, 'MAE': 19.515072}
```

LSTM model

In [27]:
```python
# --- Predict & evaluate: LSTM on FD001 test ---

from evaluator import evaluate_model

# 1) Load cached test windows
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
_, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

# 2) Load saved LSTM
lstm_path = ART_DIR / "models" / f"lstm_{DATASET.lower()}_seq{SEQ_LEN}.keras"
lstm_model = load_model(lstm_path)

# 3) Predict (LSTM expects 3D input)
y_lstm_pred = lstm_model.predict(X_test, verbose=0).squeeze()

# 4) Evaluate
evaluate_model(y_test, y_lstm_pred, model_name="LSTM")
```

```
LSTM Evaluation:
  RMSE: 21.8969
  MAE : 16.7716
```

Out[27]:
```
{'model': 'LSTM', 'RMSE': 21.89685, 'MAE': 16.771591}
```

CNN_LSTM model

In [28]:

```python
# --- Predict & evaluate: LSTM on FD001 test ---

from evaluator import evaluate_model

# 1) Load cached test windows
npz_path = ART_DIR / f"{DATASET.lower()}_seq{SEQ_LEN}.npz"
_, _, _, _, X_test, y_test = pp.load_preprocessed_data(str(npz_path))

# 2) Load saved LSTM
lstm_path = ART_DIR / "models" / f"cnn_lstm_{DATASET.lower()}_seq{SEQ_LEN}.keras"
lstm_model = load_model(lstm_path)

# 3) Predict (LSTM expects 3D input)
y_cnn_lstm_pred = lstm_model.predict(X_test, verbose=0).squeeze()

# 4) Evaluate
evaluate_model(y_test, y_cnn_lstm_pred, model_name="CNN_LSTM")
```

```
CNN_LSTM Evaluation:
  RMSE: 22.6800
  MAE : 16.8688
```

Out[28]:

```
{'model': 'CNN_LSTM', 'RMSE': 22.68002, 'MAE': 16.868832}
```

In [29]:

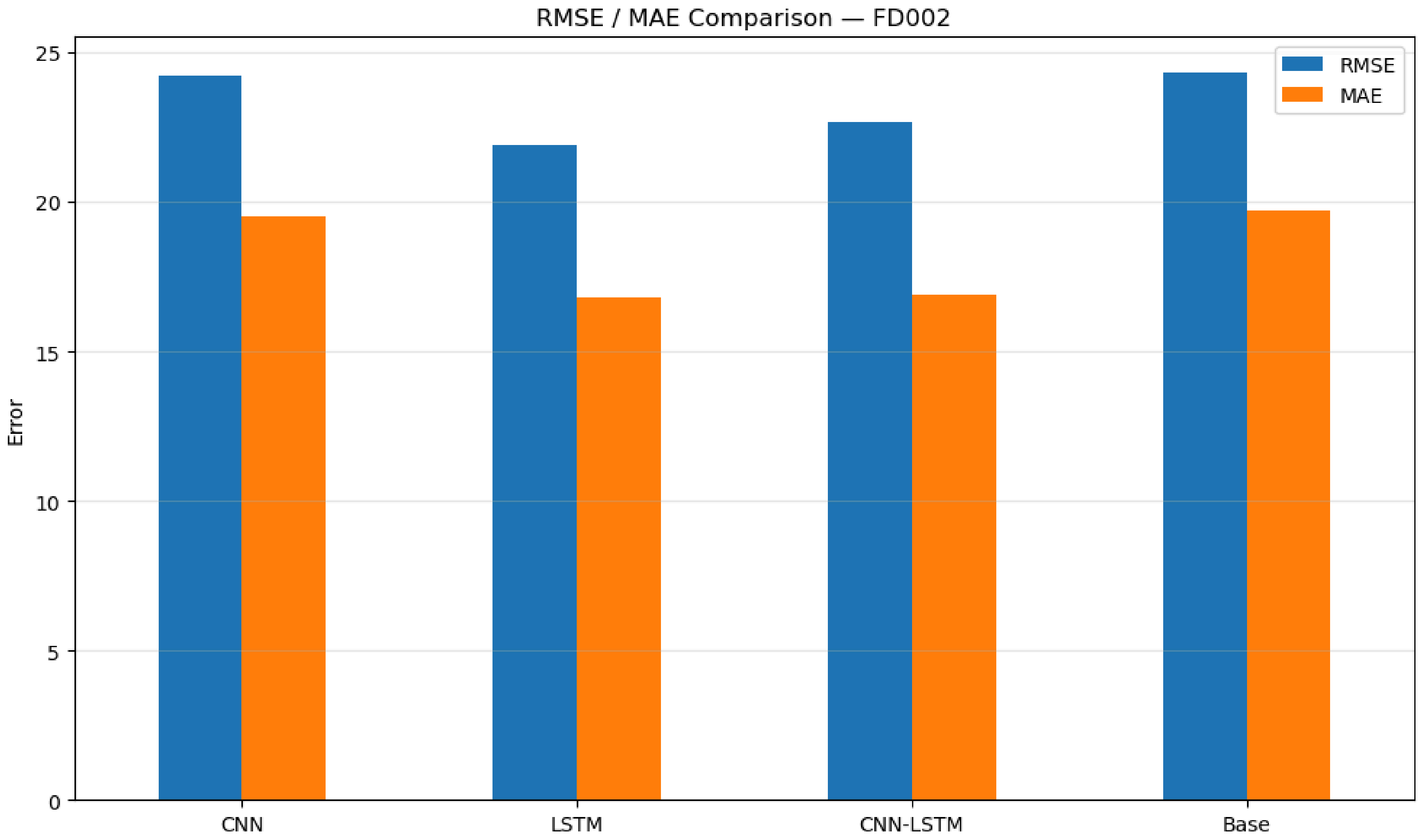
```python
from evaluator import evaluate_model

# build tables

res_cnn     = evaluate_model(y_test, y_pred_cnn,     model_name="CNN")
res_lstm    = evaluate_model(y_test, y_lstm_pred,    model_name="LSTM")
res_cnnlstm = evaluate_model(y_test, y_cnn_lstm_pred, model_name="CNN-LSTM")
res_base    = evaluate_model(y_test, y_base_pred,    model_name="Base")

metrics = [res_cnn, res_lstm, res_cnnlstm, res_base]
metrics
```

```
CNN Evaluation:
  RMSE: 24.2322
  MAE : 19.5151
LSTM Evaluation:
  RMSE: 21.8969
  MAE : 16.7716
CNN-LSTM Evaluation:
  RMSE: 22.6800
  MAE : 16.8688
Base Evaluation:
  RMSE: 24.2948
  MAE : 19.7025
```

Out[29]:

```
[{'model': 'CNN', 'RMSE': 24.232237, 'MAE': 19.515072},
 {'model': 'LSTM', 'RMSE': 21.89685, 'MAE': 16.771591},
 {'model': 'CNN-LSTM', 'RMSE': 22.68002, 'MAE': 16.868832},
 {'model': 'Base', 'RMSE': 24.294798, 'MAE': 19.702452}]
```

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



In [31]:
```python
metrics_df = pd.DataFrame([res_cnn, res_lstm, res_cnnlstm, res_base])[
    ["model", "RMSE", "MAE"]
].rename(columns={"model":"Model"})

metrics_df = metrics_df.sort_values("RMSE").reset_index(drop=True).round(2)
display(metrics_df)
```

	Model	RMSE	MAE
0	LSTM	21.900000	16.770000
1	CNN-LSTM	22.680000	16.870001
2	CNN	24.230000	19.520000
3	Base	24.290001	19.700001

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