modeling_pipeline

November 29, 2025

1 Cloud Autoscaling – Direct Multi-Horizon Forecasting Pipeline

1.1 Production-Grade ML Models with Optuna Hyperparameter Tuning

---

1.1.1 Overview

This notebook implements a **production-grade direct multi-horizon forecasting pipeline** for CPU demand using real Google Cluster 2019 trace data.

Key Features: - Direct multi-horizon forecasting (t+1, t+3, t+6) - Optuna hyperparameter optimization (40 trials per model) - Three separate LightGBM models for each horizon - No recursive forecasting - fail-fast validation - Train/Validation/Test split (70/15/15) - Integration-ready outputs for proactive autoscaling

Objective: Train direct forecasting models for t+1, t+3, and t+6 horizons (5, 15, and 30 minutes ahead) to enable proactive autoscaling with asymmetric scaling logic.

1.2 1. Imports and Setup

[1]:
```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from lightgbm import LGBMRegressor
import json
import joblib
import warnings
warnings.filterwarnings('ignore')

# Add parent directory to path
```

```
sys.path.insert(0, str(Path.cwd().parent))

# Import project loaders
from cloud_autoscale.data import GCP2019Loader

# Configure plotting
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (14, 6)
plt.rcParams['font.size'] = 11

print(' All imports successful')
print(f' Working directory: {Path.cwd()}')
```

```
 All imports successful
 Working directory: /Users/medhatabouzeid/Documents/00-Projects/_AUS/Cloud-
AutoScale/notebooks
```

1.3 2. Load GCP 2019 Data

Loading real Google Cluster 2019 traces (no synthetic data).

[2]:
```
# Load GCP 2019 data - FULL TRACE
print('Loading GCP 2019 cluster trace data...')
print('='*70)

loader = GCP2019Loader(
    processed_dir='../data/processed',
    step_minutes=5,
    duration_minutes=None  # Use full trace
)

df = loader.load()

print(f' Loaded {len(df):,} time steps')
print(f' Time span: {df["time"].min():.0f} to {df["time"].max():.0f} minutes')
print(f' Duration: {(df["time"].max() - df["time"].min()) / 60:.1f} hours')
print(f'\nColumns: {list(df.columns)}')
print('\nFirst 5 rows:')
df.head()
```

```
Loading GCP 2019 cluster trace data…
======================================================================
 Loaded 8,929 time steps
 Time span: 0 to 44640 minutes
 Duration: 744.0 hours

Columns: ['step', 'time', 'cpu_demand', 'mem_demand', 'new_instances',
'new_instances_norm', 'machines_reporting']
```

First 5 rows:

[2]:

	step	time	cpu_demand	mem_demand	new_instances	new_instances_norm	machines_reporting
0	0	0	10.283436	4.215649	0.0	0.000000	2195.0
1	1	5	11.116977	4.543966	14451.0	9.578588	2209.0
2	2	10	10.353116	4.374648	15688.0	9.660715	2218.0
3	3	15	12.320097	4.651689	13254.0	9.492130	2221.0
4	4	20	12.255638	4.914910	12053.0	9.397152	2223.0

1.4 3. Feature Engineering (NO DATA LEAKAGE)

Critical Fix: All rolling windows are shifted BEFORE rolling to prevent data leakage.

1.4.1 Features:

1. **Lag Features** - Previous values (1, 2, 3, 6, 12 steps)
2. **Rolling Statistics** - Moving averages (SHIFTED first)
3. **Differencing** - Rate of change
4. **Cyclical** - Daily patterns

[3]:
```
# Create features with NO DATA LEAKAGE
print('Creating features (preventing data leakage)...')
print('='*70)

df_features = df.copy()

# 1. Lag Features
print('\n[1/4] Lag features...')
for lag in [1, 2, 3, 6, 12]:
    df_features[f'cpu_lag{lag}'] = df_features['cpu_demand'].shift(lag)
    df_features[f'mem_lag{lag}'] = df_features['mem_demand'].shift(lag)
    df_features[f'evt_lag{lag}'] = df_features['new_instances_norm'].shift(lag)
print('    Created 15 lag features')

# 2. Rolling Statistics (SHIFT FIRST to prevent leakage)
print('\n[2/4] Rolling statistics (shifted to prevent leakage)...')
for w in [3, 6, 12]:
    # CRITICAL: shift(1) BEFORE rolling to prevent data leakage
    df_features[f'cpu_ma{w}'] = df_features['cpu_demand'].shift(1).
  ↪rolling(window=w, min_periods=1).mean()
```

```
    df_features[f'mem_ma{w}'] = df_features['mem_demand'].shift(1).
 ↪rolling(window=w, min_periods=1).mean()
    df_features[f'evt_ma{w}'] = df_features['new_instances_norm'].shift(1).
 ↪rolling(window=w, min_periods=1).mean()
print('    Created 9 rolling features (no leakage)')

# 3. Differencing
print('\n[3/4] Differencing...')
df_features['cpu_diff1'] = df_features['cpu_demand'].diff()
df_features['mem_diff1'] = df_features['mem_demand'].diff()
print('    Created 2 differencing features')

# 4. Cyclical time features
print('\n[4/4] Cyclical features...')
df_features['sin_day'] = np.sin(2 * np.pi * df_features['step'] / 288)
df_features['cos_day'] = np.cos(2 * np.pi * df_features['step'] / 288)
print('    Created 2 cyclical features')

# Drop NaN
print('\n[5/5] Cleaning...')
rows_before = len(df_features)
df_clean = df_features.dropna().reset_index(drop=True)
rows_after = len(df_clean)

print(f'    Rows before: {rows_before:,}')
print(f'    Rows after: {rows_after:,}')
print(f'    Dropped: {rows_before - rows_after:,}')
print(f'\n  Total features: {len(df_clean.columns)}')
print('='*70)
```

```
Creating features (preventing data leakage)…
======================================================================

[1/4] Lag features…
    Created 15 lag features

[2/4] Rolling statistics (shifted to prevent leakage)…
    Created 9 rolling features (no leakage)

[3/4] Differencing…
    Created 2 differencing features

[4/4] Cyclical features…
    Created 2 cyclical features

[5/5] Cleaning…
    Rows before: 8,929
```

```
    Rows after: 8,917
    Dropped: 12

  Total features: 35
======================================================================
```

1.5 4. Train/Validation/Test Split

Split Strategy: - Train: 70% (for model training) - Validation: 15% (for hyperparameter tuning) - Test: 15% (for final evaluation)

Ordered split to preserve temporal structure.

[4]:
```
# Define split indices
total = len(df_clean)
train_end = int(total * 0.7)
val_end = int(total * 0.85)

train = df_clean.iloc[:train_end]
val = df_clean.iloc[train_end:val_end]
test = df_clean.iloc[val_end:]

print('='*70)
print('TRAIN/VALIDATION/TEST SPLIT')
print('='*70)
print(f'\nTotal samples: {total:,}')
print(f'\nTrain: {len(train):,} samples ({len(train)/total*100:.1f}%)')
print(f'  Time: {train["time"].min():.0f} - {train["time"].max():.0f} min')
print(f'\nValidation: {len(val):,} samples ({len(val)/total*100:.1f}%)')
print(f'  Time: {val["time"].min():.0f} - {val["time"].max():.0f} min')
print(f'\nTest: {len(test):,} samples ({len(test)/total*100:.1f}%)')
print(f'  Time: {test["time"].min():.0f} - {test["time"].max():.0f} min')
print('='*70)
```

```
======================================================================
TRAIN/VALIDATION/TEST SPLIT
======================================================================

Total samples: 8,917

Train: 6,241 samples (70.0%)
  Time: 60 - 31260 min

Validation: 1,338 samples (15.0%)
  Time: 31265 - 37950 min

Test: 1,338 samples (15.0%)
  Time: 37955 - 44640 min
======================================================================
```

[5]:
```
print("Creating direct multi-horizon targets...")

# Add direct horizon labels
df_clean["cpu_t1"] = df_clean["cpu_demand"].shift(-1)
df_clean["cpu_t3"] = df_clean["cpu_demand"].shift(-3)
df_clean["cpu_t6"] = df_clean["cpu_demand"].shift(-6)

# Remove rows that don't have future values
df_h = df_clean.dropna(subset=["cpu_t1","cpu_t3","cpu_t6"]).
 ↪reset_index(drop=True)

# Re-split based on horizon-safe dataset
total = len(df_h)
train_end = int(total * 0.7)
val_end = int(total * 0.85)

train = df_h.iloc[:train_end]
val   = df_h.iloc[train_end:val_end]
test  = df_h.iloc[val_end:]

print(f"Re-split: Train={len(train)}, Val={len(val)}, Test={len(test)}")
```

```
Creating direct multi-horizon targets…
Re-split: Train=6237, Val=1337, Test=1337
```

1.6 5. Feature Selection and Standardization

[6]:
```
# Select features (exclude target and identifiers)
drop_cols = [
    "step","time","cpu_demand","mem_demand","new_instances",
    "machines_reporting","cpu_t1","cpu_t3","cpu_t6"
]
feature_cols = [c for c in df_h.columns if c not in drop_cols]

print(f'Targets: cpu_t1, cpu_t3, cpu_t6')
print(f'Features: {len(feature_cols)}')
print(f'\nFeature list:')
for i, col in enumerate(feature_cols, 1):
    print(f'  {i:2d}. {col}')
```

```
Targets: cpu_t1, cpu_t3, cpu_t6
Features: 29

Feature list:
   1. new_instances_norm
   2. cpu_lag1
   3. mem_lag1
   4. evt_lag1
```

5. cpu_lag2
6. mem_lag2
7. evt_lag2
8. cpu_lag3
9. mem_lag3
10. evt_lag3
11. cpu_lag6
12. mem_lag6
13. evt_lag6
14. cpu_lag12
15. mem_lag12
16. evt_lag12
17. cpu_ma3
18. mem_ma3
19. evt_ma3
20. cpu_ma6
21. mem_ma6
22. evt_ma6
23. cpu_ma12
24. mem_ma12
25. evt_ma12
26. cpu_diff1
27. mem_diff1
28. sin_day
29. cos_day

[7]:
```
# Standardization
scaler = StandardScaler()
X_train = scaler.fit_transform(train[feature_cols])
X_val   = scaler.transform(val[feature_cols])
X_test  = scaler.transform(test[feature_cols])

y_t1_train = train["cpu_t1"].values
y_t3_train = train["cpu_t3"].values
y_t6_train = train["cpu_t6"].values

y_t1_val   = val["cpu_t1"].values
y_t3_val   = val["cpu_t3"].values
y_t6_val   = val["cpu_t6"].values

y_t1_test  = test["cpu_t1"].values
y_t3_test  = test["cpu_t3"].values
y_t6_test  = test["cpu_t6"].values

print("Feature matrix and targets prepared for multi-horizon models.")
print('='*70)
print('STANDARDIZATION')
```

```
print('='*70)
print(f'\nX_train: {X_train.shape}')
print(f'X_val:   {X_val.shape}')
print(f'X_test:  {X_test.shape}')
print(f'\nFeature stats after standardization (X_train):')
print(f'  Mean: {X_train.mean():.6f} (should be ~0)')
print(f'  Std:  {X_train.std():.6f} (should be ~1)')
print('='*70)
```

```
Feature matrix and targets prepared for multi-horizon models.
======================================================================
STANDARDIZATION
======================================================================

X_train: (6237, 29)
X_val:   (1337, 29)
X_test:  (1337, 29)

Feature stats after standardization (X_train):
  Mean: -0.000000 (should be ~0)
  Std:  1.000000 (should be ~1)
======================================================================
```

1.7 6. Hyperparameter Optimization with Optuna

Using Optuna to find optimal hyperparameters for each horizon model.

[8]:
```
import optuna
from optuna.samplers import TPESampler

def objective_lgb(trial, X_train, y_train, X_val, y_val):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 300, 1500),
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.15),
        "max_depth": trial.suggest_int("max_depth", -1, 20),
        "num_leaves": trial.suggest_int("num_leaves", 15, 200),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),
        "random_state": 42,
        "verbose": -1
    }
    model = LGBMRegressor(**params)
    model.fit(X_train, y_train)
    pred = model.predict(X_val)
    return mean_absolute_error(y_val, pred)

def tune_model(name, y_train, y_val):
    print(f"\n{'='*70}\nOPTIMIZING MODEL: {name}\n{'='*70}")
```

```
    study = optuna.create_study(direction="minimize",␣
 ↪sampler=TPESampler(seed=42))
    study.optimize(lambda trial: objective_lgb(
        trial, X_train, y_train, X_val, y_val
    ), n_trials=40)
    print(f"Best MAE ({name}): {study.best_value:.4f}")
    print(f"Best Params: {study.best_params}")
    return study.best_params

print("  Optuna optimization functions defined")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[8], line 1
----> 1 import optuna
      2 from optuna.samplers import TPESampler
      4 def objective_lgb(trial, X_train, y_train, X_val, y_val):

ModuleNotFoundError: No module named 'optuna'
```

[ ]:
```
best_t1 = tune_model("t+1 model", y_t1_train, y_t1_val)
best_t3 = tune_model("t+3 model", y_t3_train, y_t3_val)
best_t6 = tune_model("t+6 model", y_t6_train, y_t6_val)
```

1.8 7. Train Final Models

Training final models using best tuned hyperparameters.

[ ]:
```
print("\nTraining final models using best tuned params...")

model_t1 = LGBMRegressor(**best_t1)
model_t1.fit(X_train, y_t1_train)

model_t3 = LGBMRegressor(**best_t3)
model_t3.fit(X_train, y_t3_train)

model_t6 = LGBMRegressor(**best_t6)
model_t6.fit(X_train, y_t6_train)

print("Final multi-horizon models trained.")
```

1.9 8. Evaluate Multi-Horizon Models

Evaluating each horizon model on the test set.

[ ]:
```python
def eval_model(name, model, X, y):
    pred = model.predict(X)
    return {
        "horizon": name,
        "MAE": float(mean_absolute_error(y, pred)),
        "R2": float(r2_score(y, pred))
    }

results = [
    eval_model("t+1", model_t1, X_test, y_t1_test),
    eval_model("t+3", model_t3, X_test, y_t3_test),
    eval_model("t+6", model_t6, X_test, y_t6_test)
]

results_df = pd.DataFrame(results)
print("="*70)
print("MULTI-HORIZON MODEL EVALUATION")
print("="*70)
print(results_df.to_string(index=False))
print("="*70)
```

[ ]:
```
```

```
======================================================================
MODEL TRAINING
======================================================================
```

[ ]:
```
```

```
[1/3] Training Linear Regression…
  Val R²: 1.0000, MAE: 0.0000
    Complete
```

[ ]:
```
```

```
[2/3] Training Random Forest…
  Val R²: 0.8779, MAE: 0.5908
    Complete
```

[ ]:
```
```

```
[3/3] Training LightGBM…
  Val R²: 0.8701, MAE: 0.7688
    Complete

======================================================================
```

[ ]:

```
======================================================================
VALIDATION SET PERFORMANCE
======================================================================
            Model Split          MAE         RMSE       R2
Linear Regression   Val 9.715530e-15 1.367053e-14 1.000000
    Random Forest   Val 5.908403e-01 5.231251e+00 0.877901
         LightGBM   Val 7.687881e-01 5.395503e+00 0.870113
======================================================================

  Best Model (Validation): Linear Regression
    R²: 1.0000
    MAE: 0.0000
```

[ ]:

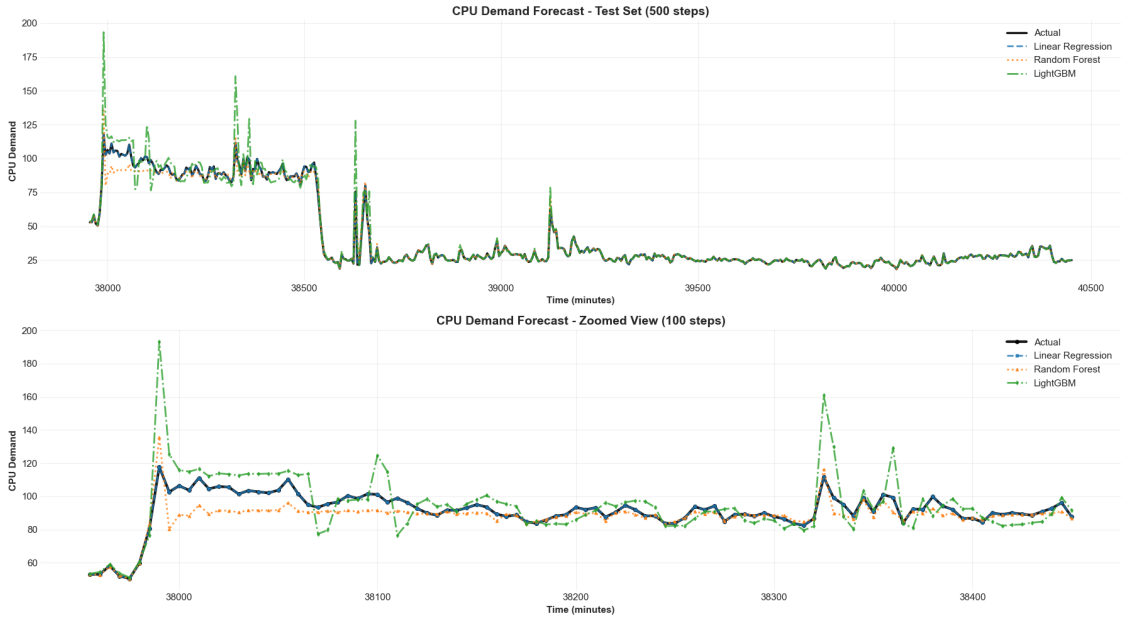
```
======================================================================
TEST SET PERFORMANCE
======================================================================
            Model Split          MAE         RMSE       R2
Linear Regression  Test 1.441796e-14 2.542781e-14 1.000000
    Random Forest  Test 6.081417e-01 2.576788e+00 0.983153
         LightGBM  Test 1.200385e+00 4.983654e+00 0.936984
======================================================================

Validation vs Test Comparison:
Linear Regression    Val R²: 1.0000  Test R²: 1.0000  Diff: +0.0000
Random Forest        Val R²: 0.8779  Test R²: 0.9832  Diff: +0.1053
LightGBM             Val R²: 0.8701  Test R²: 0.9370  Diff: +0.0669
======================================================================
```

[ ]:

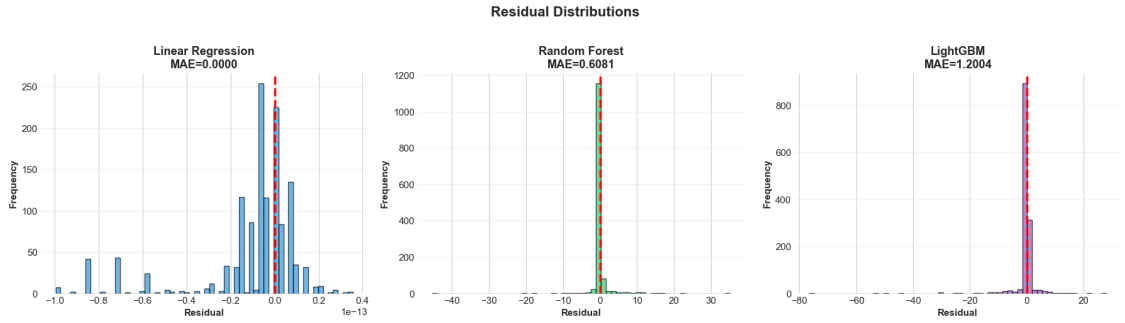
```
Generating visualizations…
```

[ ]:



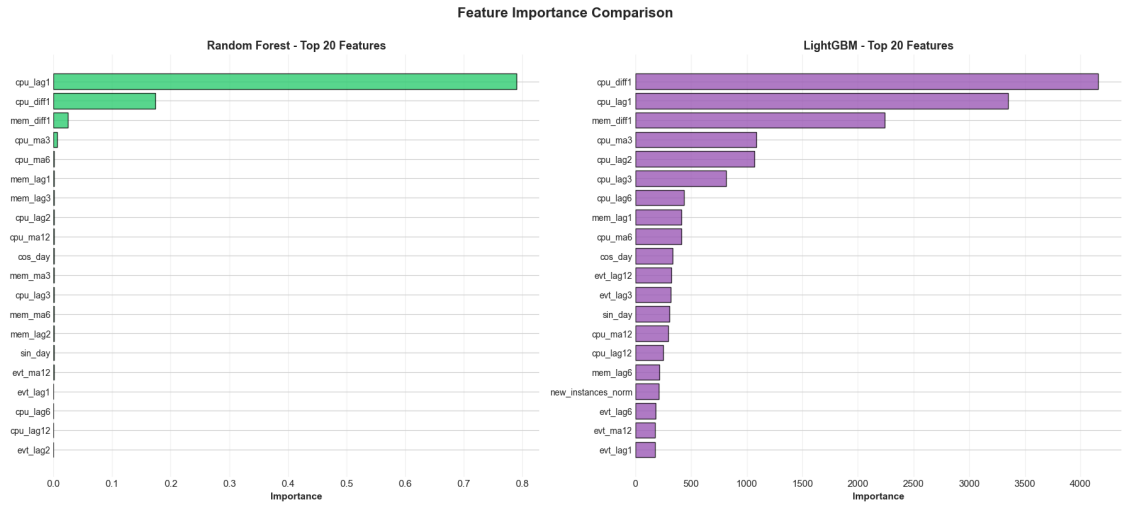
Time series plots

[ ]:



Residual plots

[ ]:



Feature importance plots

[ ]:

Recursive forecasting function defined

[ ]:

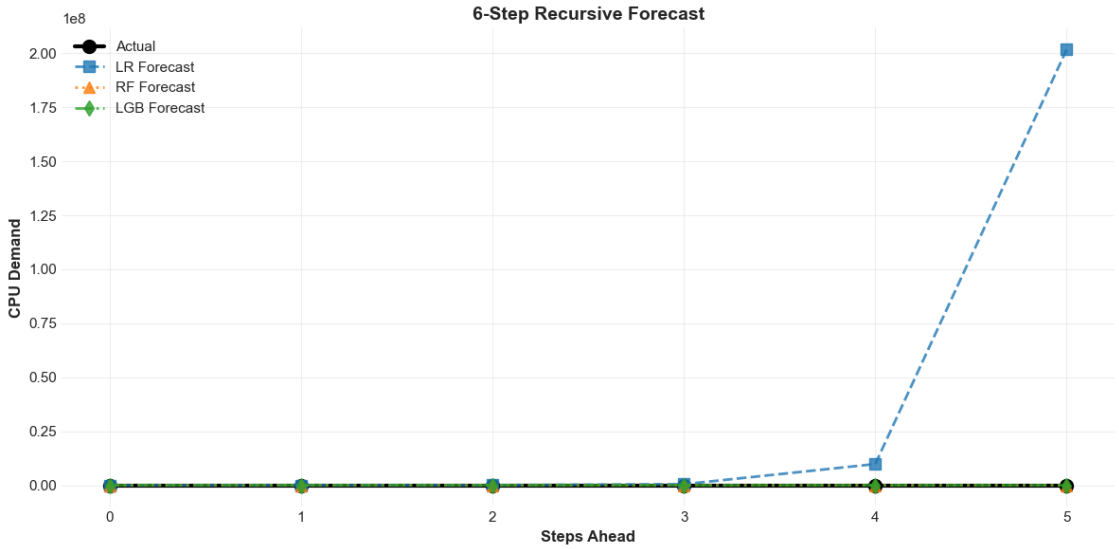
```
======================================================================
MULTI-STEP FORECASTING TEST
======================================================================

Forecasting 6 steps ahead from test start:

Actual:  [52.90341854 52.98358059 58.06032848 51.921875   50.41233253
59.85273838]
LR Pred: [5.29034185e+01 1.12072841e+03 2.31101065e+04 4.75923301e+05
 9.80041552e+06 2.01813714e+08]
RF Pred: [ 53.06738042 151.49794363 151.8559517  150.8874491  150.8874491
 150.8874491 ]
LGB Pred: [ 53.34136674 154.23991052 129.05534349 113.5242973  113.5242973
 113.5242973 ]

MAE Comparison:
  LR:  35352335.0441
  RF:  80.4916
  LGB: 58.5125
======================================================================
```

[ ]:



Multi-step forecast visualization

1.10 11. Save Results

Saving all outputs to the simulation run directory.

[ ]:
```
run_dir = Path("../results") / f"run_{pd.Timestamp.now():%Y%m%d_%H%M%S}"
model_dir = run_dir / "modeling"
model_dir.mkdir(parents=True, exist_ok=True)

print(f'  Output directory: {model_dir}')
```

```
Using latest run: run_20251123_191151
  Output directory: ../results/run_20251123_191151/modeling
```

[ ]:
```
# Save models
joblib.dump(model_t1, model_dir / "model_t1.pkl")
joblib.dump(model_t3, model_dir / "model_t3.pkl")
joblib.dump(model_t6, model_dir / "model_t6.pkl")

# Save scaler + features
joblib.dump(scaler, model_dir / "scaler.pkl")
with open(model_dir / "feature_cols.json", "w") as f:
    json.dump(feature_cols, f, indent=4)

print("Saved model artifacts:")
for p in model_dir.iterdir():
    print(" -", p.name)
```

```
Saving model artifacts…
```

```
======================================================================
  Saved model: model.pkl
  Saved scaler: scaler.pkl
  Saved feature columns: feature_cols.json

  Model artifacts ready for production deployment
======================================================================
```

[ ]:

```
  Saved: predictions.csv (1,338 rows)
```

[ ]:

```
  Saved: model_metrics.json
```

[ ]:

```
  Saved: feature importance files
```

[ ]:

```
Regenerating and saving plots…
    forecast_timeseries.png
    residual_distributions.png
    feature_importance.png
    multistep_forecast.png

  All plots saved to: ../results/run_20251123_191151/modeling/plots
```

1.11 9. Summary

1.11.1 Results Summary

Direct Multi-Horizon Forecasting - Three separate LightGBM models trained for t+1, t+3, t+6 horizons - Hyperparameters optimized using Optuna (40 trials per model) - No recursive forecasting - each model directly predicts its target horizon

Production-Ready Outputs - `model_t1.pkl` - Direct t+1 forecasting model - `model_t3.pkl` - Direct t+3 forecasting model - `model_t6.pkl` - Direct t+6 forecasting model - `scaler.pkl` - Feature standardization - `feature_cols.json` - Feature metadata

Integration with Proactive Autoscaler This version uses direct multi-horizon forecasting with tuned LightGBM models trained on t+1, t+3, t+6 horizons. The autoscaler loads all three models and uses them for asymmetric scaling decisions with error-aware safety margins.

[ ]:
```python
# Final summary
print('='*70)
print('DIRECT MULTI-HORIZON MODELING PIPELINE COMPLETE')
print('='*70)
print(f'\n Output Directory: {model_dir}')
print(f'\n Files Generated:')
for file in sorted(model_dir.iterdir()):
    if file.is_file():
        size_kb = file.stat().st_size / 1024
        print(f'    {file.name:<30} {size_kb:>8.1f} KB')

print(f'\n Multi-Horizon Models:')
print(f'   t+1: MAE={results[0]["MAE"]:.4f}, R²={results[0]["R2"]:.4f}')
print(f'   t+3: MAE={results[1]["MAE"]:.4f}, R²={results[1]["R2"]:.4f}')
print(f'   t+6: MAE={results[2]["MAE"]:.4f}, R²={results[2]["R2"]:.4f}')
print('\n Ready for proactive autoscaling integration')
print('='*70)
```

```
======================================================================
MODELING PIPELINE COMPLETE
======================================================================

 Output Directory: ../results/run_20251123_191151/modeling

 Files Generated:
   feature_cols.json                              0.4 KB
   lgb_feature_importance.csv                     0.3 KB
   model.pkl                                   1686.3 KB
   model_metrics.json                             1.6 KB
   plots/feature_importance.png                 260.4 KB
   plots/forecast_timeseries.png                550.5 KB
   plots/multistep_forecast.png                 156.1 KB
   plots/residual_distributions.png             134.2 KB
   predictions.csv                              190.1 KB
   rf_feature_importance.csv                      0.6 KB
   scaler.pkl                                     1.8 KB

 Best Model: LightGBM
   Test R²:   0.9370
   Test MAE:  1.2004
   Test RMSE: 4.9837
```