Step-by-Step Instruction Manual: Building Your Version 1.0–1.5 Core Forecasting Engine with LST HDF Data

This comprehensive manual gives you an **expert step-by-step guide** for building a robust forecasting system with *incremental upgrades, safe GitHub versioning, and region-specific data handling*. It reflects deep research and practical, modern best practices for ML engineering with large time series and HDF5 data.

1. Project Setup & Versioning

1.1. Initialize Your GitHub Repo

- `git init`
- Create a `.gitignore` containing `/data/`, `*.h5`, `*.ckpt`, `*.safetensors`, `.env`, etc.
- Write a short `README.md`.
- Add your starter code, commit:

```
git add .
git commit -m "Initial project scaffolding"
```

Tip:
Commit after each significant code addition or refactor. Push to GitHub after each local milestone. For large files (models/datasets), use Git LFS or DVC for data and model versioning (see). [1] [2]

2. LST Data Extraction & Region-Specific Processing

2.1. Explore the HDF Structure

Use `h5py` to inspect the contents:

```
import h5py
f = h5py.File('YOUR_LST_FILE.h5', 'r')
def printname(name): print(name)
f.visit(printname)
```

Identify datasets: likely keys include `/LST`, `/Latitude`, `/Longitude`, `/Time`, etc.

2.2. Extract Region-Specific Data

Suppose you want data for Karnataka. Define bounding box (lat/lon min/max) for your region.

```
import numpy as np

with h5py.File('YOUR_LST_FILE.h5', 'r') as f:
    lst = f['LST'][:]
    lats = f['Latitude'][:]
    lons = f['Longitude'][:]
    times = f['Time'][:]
    # Mask for region
    region_mask = (lats >= min_lat) & (lats <= max_lat) & \
                  (lons >= min_lon) & (lons <= max_lon)
    lst_region = lst[region_mask]
    times_region = times[region_mask]
```

Tip: *Save region-specific datasets to new HDF5 or as a pandas DataFrame for fast access later.*

2.3. Resample & Align Data

- Downsample to **hourly** or **daily** averages if needed (`pandas.DataFrame.resample('H')`).
- Fill or mask missing values.

3. Model Data Preparation

3.1. Prepare Data for Chronos/LSTM Models

- Structure as a pandas DataFrame: columns = [timestamp, LST value]
- For multivariate models, add further columns (solar irradiance, wind when available).

```
import pandas as pd
df = pd.DataFrame({'timestamp': times_region, 'LST': lst_region})
df.set_index('timestamp', inplace=True)
df = df.resample('H').mean().interpolate()
```

- **Split** into training/validation/test (e.g., train on `2015-2019`, validate on `2020`, test on `2021`).

3.2. Data Normalization

- Normalize features (e.g., MinMaxScaler or StandardScaler) for neural nets/LSTMs.

4. Model Training, Incremental Development & Future Proofing

4.1. Version 1.0: Baseline Model with LST Data Only

1. **Start with Chronos-T5 or LSTM:**
   Load model with LST as the sole feature.
2. **Train and evaluate** on region-specific historical data.

Can you start with LST only?
Yes—you can start training a model with just LST data. Afterward, you can upgrade to add other features or combine datasets by retraining or using transfer/incremental learning approaches (). [3] [4] [5]

4.2. Upgrading to Version 1.5: Adding More Data/Features

Best practice for adding new data/feature:

- **Option 1:** Merge old + new data, retrain the model from scratch for full performance.
- **Option 2:** Use *incremental/transfer learning*:
  - Load the trained model, freeze some layers or weights, and fine-tune on the expanded feature set ().[4] [5] [3]
  - Impute or fill missing features with mean/median/random or train models for each feature set and ensemble predictions.

5. Safe Git/GitHub Versioning at Every Step

RECOMMENDED GIT STRATEGY:

- **After each major step below, commit and push:**
  1. After HDF region extraction code works.
  2. After data cleaning and alignment.
  3. After initial model pipeline/train script ready.
  4. After baseline model training & evaluation.
  5. Before/after each new feature addition.
  6. Before refactoring code structure (e.g., modularizing functions/classes).
  7. After adding documentation/tests.

Sample commands:

```
git add src/data/region_extract.py
git commit -m "Add region-specific extraction for LST"
git push origin main
# Tag special milestones:
git tag -a v1.0 -m "Baseline LST-only model complete"
git push origin v1.0
```

```
# New feature/model work:
git checkout -b add_solar_feature
```

Consider using tags (`git tag -a v1.0 -m "..."`) before major shifts, and branches for adding features. DVC or Git LFS can version large model weights/artifacts.

6. Continual Improvement & Milestones

Version 1.0

- Region-specific LST processing, cleaning, and exploratory visualizations.
- Baseline ML model (Chronos or LSTM) using only LST for a single region.
- All code and artifacts pushed to GitHub.

Version 1.5

- Integrate additional MOSDAC fields (solar, wind, cloud).
- Merge with main data pipeline and retrain model.
- Use ensemble or transfer learning for incremental feature addition.
- All improvements committed to GitHub with tags/branches.

7. Key Best Practices

- **ALWAYS** backup code before major changes (commit, push, make a tag or branch).
- Use **atomic, descriptive commits**: “fix LST resampling for daylight hours”.
- On error or dead-end: roll back with `git checkout` to previous state.
- Document every new model training session, feature change, and experiment in README or with experiment logs.

8. References/Further Reading

- Incremental/transfer learning for time series[3]
- Machine Learning Versioning Best Practices[1]
- Multi-feature time series modeling[6]
- Pretrained model class-incremental learning[5]
- Working with HDF5 in Python[7]

Summary Table: Versioning Checkpoints

Step	Commit Tag/Message	Can Rollback?	What to Push
Initial Extraction	region-extract-v1	Yes	code, scripts
Data Cleaning	data-clean-LST-v1	Yes	new scripts, notes
Baseline Model	baseline-LST-model-v1	Yes	code, configs, logs
Add Feature	add-solar-feature-v1	Yes	new code, notes
Retraining/Upgrade	retrain-with-solar-v1	Yes	model code, weights
Major Refactor	refactor-pipeline-v1	Yes	all code, docs

By following these instructions, you can:

- Start with only LST for Version 1.0, and seamlessly add new features/data for Version 1.5 and beyond.
- Safely edit, revert, and enhance your codebase at every stage, never losing progress thanks to structured GitHub commits, tags, and branches.
- Train robust, scalable forecasting models and upgrade them as your data pipeline grows.

⁂

1. https://dagshub.com/blog/mastering-version-control-for-ml-models-best-practices-you-need-to-know/
2. https://lakefs.io/blog/model-versioning/
3. https://www.sciencedirect.com/science/article/abs/pii/S0045790621001592
4. https://community.deeplearning.ai/t/adding-new-feature-to-a-pretrained-model/206078
5. https://arxiv.org/abs/2503.07153
6. https://www.machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
7. https://pythonforthelab.com/blog/how-to-use-hdf5-files-in-python
8. https://docs.h5py.org/en/latest/high/dataset.html
9. https://towardsdatascience.com/having-hundreds-of-time-series-to-explore-and-not-knowing-where-to-start-5e7873fd2028/
10. https://arxiv.org/html/2403.07815v1