A **Sales Forecasting System** can be a very powerful AI-driven product, especially when you have rich, structured data like **Region, Territory, Area, and Time-based (Year/Month)** attributes.

Let’s break this down into a **real-world AI product design** — including **architecture, model strategy, and best practices.**

**🧠 1. Goal Definition**

Your goal:

Predict future sales (next month/quarter/year) for each region/territory based on historical trends, seasonality, and possibly external factors.

**Common Forecasting Objectives:**

* **Total Sales Prediction** per month (overall or by region).
* **Territory Performance Forecasting** for planning resources.
* **Demand Forecasting** for product inventory optimization.
* **Growth/Decline Detection** (to identify underperforming areas).

**🧩 2. Data Design and Preparation**

Since you already have a **huge dataset**, this is your strongest asset.  
Make sure your data pipeline is **clean, well-structured, and versioned**.

**✅ Suggested Schema**

| **Date** | **Region** | **Territory** | **Area** | **Product** | **Sales** | **Quantity** | **Channel** | **...** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2023-01 | North | Dhaka | Mirpur | ProductA | 1,200,000 | 240 | Distributor | ... |

**✅ Preprocessing Steps**

1. **Handle Missing Data:** Fill missing months or zero sales.
2. **Feature Engineering:**
   * Extract: Year, Month, Quarter, DayOfWeek, etc.
   * Add time features like lag\_1, lag\_3, rolling\_mean\_3.
   * Include region-based features (population, outlets, etc. if available).
3. **Normalize / Scale:** For models like LSTM or NeuralProphet.
4. **Aggregate:**
   * For regional forecasting → group by Region/Month.
   * For finer granularity → Territory/Month.

**🤖 3. Model Selection (Forecasting Techniques)**

**🔹 Classical Models**

| **Model** | **When to Use** | **Description** |
| --- | --- | --- |
| **ARIMA / SARIMA** | Univariate time series | Works well with single-region forecasts. Handles seasonality. |
| **Prophet (Facebook)** | Quick and interpretable | Handles seasonality, holidays, trends well. |
| **XGBoost / LightGBM** | Tabular forecasting | Uses historical + categorical features for multi-region forecasts. |

**🔹 Deep Learning Models**

| **Model** | **Description** |
| --- | --- |
| **LSTM / GRU** | Captures long-term temporal dependencies. |
| **Temporal Fusion Transformer (TFT)** | SOTA for multi-time series forecasting. |
| **NeuralProphet** | Combines Prophet + Neural networks, easy to use. |

👉 Best practice:  
Start with **Prophet** or **LightGBM**, then move to **LSTM or TFT** for fine-tuned accuracy.

**🏗️ 4. System Architecture (AI Product View)**

Here’s a simple, scalable architecture:

┌─────────────────────┐

│ SQL / Data Lake │

└─────────┬───────────┘

│

Data Extraction (ETL)

│

┌─────────▼──────────┐

│ Data Preprocessor │ ← (Cleaning, Feature Engg.)

└─────────┬──────────┘

│

┌─────────▼──────────┐

│ ML / DL Model │ ← (Prophet, LSTM, TFT)

└─────────┬──────────┘

│

┌─────────▼──────────┐

│ Forecast Database │

└─────────┬──────────┘

│

┌─────────▼──────────┐

│ Dashboard (PowerBI │

│ / Streamlit / ASP) │

└────────────────────┘

**💡 5. Key Best Practices**

**⚙️ Data & Modeling**

* Maintain **separate time series per territory/region**.
* Split data by **time-based validation** (train on past → validate on future).
* Use **rolling forecast window** for real-world simulation.
* Apply **feature importance** (especially for XGBoost/LGBM) to understand key drivers.

**🧾 MLOps Practices**

* Use **DVC or MLflow** for experiment tracking.
* Schedule monthly retraining pipelines (Airflow, Prefect, or cron jobs).
* Deploy model as a **REST API** (FastAPI / Flask / .NET Web API).

**📊 Visualization**

* Build a forecasting dashboard with:
  + Actual vs Predicted
  + Region-wise forecast comparison
  + Confidence intervals
  + Trend & seasonality decomposition

**🚀 6. Example Project Plan**

| **Phase** | **Description** |
| --- | --- |
| Phase 1 | Data Cleaning + Exploratory Analysis |
| Phase 2 | Build baseline (Prophet / LGBM) |
| Phase 3 | Compare with LSTM / TFT |
| Phase 4 | Develop forecast API |
| Phase 5 | Build visualization dashboard |
| Phase 6 | Automate retraining & monitoring |

**🌍 Optional (Advanced)**

If you want to make it *enterprise-grade*:

* Add **external features**: marketing spend, holidays, weather, GDP, etc.
* Implement **anomaly detection** for unusual sales patterns.
* Use **Elasticsearch + Kibana** for fast data indexing and querying.

Let’s reframe your project architecture using **Flask + Prophet + LightGBM + SQL Server**.  
Below is a **complete professional setup** with best practices, structure, and example code.

**🧩 1. Updated Architecture (Flask-based)**

+--------------------+

| SQL Server DB |

+--------------------+

|

v

+--------------------+

| ETL & Feature Eng |

+--------------------+

|

v

+----------------------------+

| Flask API (Prophet + LGBM) |

+----------------------------+

|

v

+--------------------+

| Model Store (joblib)

+--------------------+

|

v

+--------------------+

| Dashboard (PowerBI, Streamlit)

+--------------------+

**🏗️ 2. Folder Structure**

sales\_forecast\_flask/

├─ app.py # Flask entry point

├─ config.py # Configs (DB, paths, env)

├─ requirements.txt

├─ /services

│ ├─ etl.py # SQL → Pandas + feature engineering

│ ├─ prophet\_train.py # Prophet training and forecast

│ ├─ tree\_train.py # LightGBM/XGBoost models

│ ├─ inference.py # Unified prediction logic

├─ /utils

│ ├─ db.py # SQLAlchemy connection

│ ├─ persistence.py # save/load models

│ ├─ metrics.py # RMSE, MAPE etc.

├─ /models # stored trained models

│ └─ prophet\_region1.pkl

│ └─ lgbm\_global.pkl

├─ /scripts

│ └─ retrain\_models.py

└─ /notebooks

└─ EDA.ipynb

**⚙️ 3. config.py**

import os

DB\_CONN = os.getenv(

"DB\_CONN",

"mssql+pyodbc://username:password@localhost/SalesDB?driver=ODBC+Driver+17+for+SQL+Server"

)

MODEL\_DIR = os.path.join(os.getcwd(), "models")

**🧱 4. Flask API (app.py)**

from flask import Flask, request, jsonify

from services.etl import load\_monthly\_sales, create\_lags

from services.prophet\_train import train\_prophet, forecast\_prophet

from services.tree\_train import train\_lgb

from utils.db import get\_engine

from utils.persistence import save\_model, load\_model

import os

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return jsonify({"message": "Sales Forecast API is running."})

@app.route('/train', methods=['POST'])

def train():

data = request.get\_json()

model\_type = data.get("model\_type", "lgbm")

region = data.get("region", None)

engine = get\_engine()

df = load\_monthly\_sales(engine)

df = create\_lags(df, ["region", "territory", "area", "product\_id"])

if model\_type == "prophet":

if not region:

return jsonify({"error": "region required for Prophet"}), 400

sub = df[df["region"] == region][["ds", "total\_sales"]].rename(columns={"total\_sales": "y"})

model = train\_prophet(sub)

path = save\_model(model, f"prophet\_{region}")

else:

features = ["month", "year", "lag\_1", "lag\_3", "rolling\_3"]

models = train\_lgb(df.dropna(), features)

path = save\_model(models[-1], "lgbm\_global")

return jsonify({"status": "trained", "model\_path": path})

@app.route('/predict', methods=['GET'])

def predict():

model\_name = request.args.get("model\_name", "lgbm\_global")

model\_path = os.path.join("models", f"{model\_name}\_v1.pkl")

try:

model = load\_model(model\_path)

except Exception as e:

return jsonify({"error": f"Model not found: {e}"}), 404

# In production, fetch last N months from SQL and generate forecast

preds = [12345, 14500, 13400] # placeholder

return jsonify({"model": model\_name, "predictions": preds})

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=8000, debug=True)

**🧮 5. ETL Example (services/etl.py)**

import pandas as pd

def load\_monthly\_sales(engine):

query = "SELECT yearmonth, region, territory, area, product\_id, total\_sales FROM sales\_agg\_monthly"

df = pd.read\_sql(query, engine)

df["ds"] = pd.to\_datetime(df["yearmonth"] + "-01")

df = df.sort\_values(["region", "territory", "area", "product\_id", "ds"])

df["month"] = df["ds"].dt.month

df["year"] = df["ds"].dt.year

return df

def create\_lags(df, group\_cols, target\_col="total\_sales", lags=[1,3]):

for lag in lags:

df[f"lag\_{lag}"] = df.groupby(group\_cols)[target\_col].shift(lag)

df["rolling\_3"] = df.groupby(group\_cols)[target\_col].shift(1).rolling(3).mean().reset\_index(level=group\_cols, drop=True)

return df

**📈 6. Prophet Trainer (services/prophet\_train.py)**

from prophet import Prophet

def train\_prophet(df):

model = Prophet(yearly\_seasonality=True)

model.fit(df)

return model

def forecast\_prophet(model, periods=6):

future = model.make\_future\_dataframe(periods=periods, freq='MS')

forecast = model.predict(future)

return forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']]

**🌳 7. LightGBM Trainer (services/tree\_train.py)**

import lightgbm as lgb

from sklearn.model\_selection import TimeSeriesSplit

from sklearn.metrics import mean\_squared\_error

def train\_lgb(df, features, target="total\_sales"):

X, y = df[features], df[target]

tscv = TimeSeriesSplit(n\_splits=3)

models = []

for train\_idx, val\_idx in tscv.split(X):

X\_train, X\_val = X.iloc[train\_idx], X.iloc[val\_idx]

y\_train, y\_val = y.iloc[train\_idx], y.iloc[val\_idx]

dtrain = lgb.Dataset(X\_train, y\_train)

dval = lgb.Dataset(X\_val, y\_val)

params = {"objective": "regression", "metric": "rmse"}

model = lgb.train(params, dtrain, valid\_sets=[dval], early\_stopping\_rounds=20)

preds = model.predict(X\_val)

print("Fold RMSE:", mean\_squared\_error(y\_val, preds, squared=False))

models.append(model)

return models

**💾 8. Model Persistence (utils/persistence.py)**

import joblib, os

def save\_model(model, name, version="v1"):

os.makedirs("models", exist\_ok=True)

path = os.path.join("models", f"{name}\_{version}.pkl")

joblib.dump(model, path)

return path

def load\_model(path):

return joblib.load(path)

**🗄️ 9. SQL Connection (utils/db.py)**

from sqlalchemy import create\_engine

from config import DB\_CONN

def get\_engine():

return create\_engine(DB\_CONN)

**🧰 10. Run & Test**

# Install dependencies

pip install -r requirements.txt

# Run Flask

python app.py

**Example API calls:**

**Train LightGBM:**

curl -X POST http://localhost:8000/train -H "Content-Type: application/json" -d '{"model\_type":"lgbm"}'

**Train Prophet:**

curl -X POST http://localhost:8000/train -H "Content-Type: application/json" -d '{"model\_type":"prophet","region":"North"}'

**Predict:**

curl "http://localhost:8000/predict?model\_name=lgbm\_global"

**✅ Best Practices**

* Use gunicorn or waitress for production deployment.
* Add background jobs (Celery or APScheduler) for periodic retraining.
* Store models and logs in Azure Blob, S3, or SQL VARBINARY for persistence.
* Integrate MLflow for experiment tracking.
* Build a simple Streamlit dashboard for region/territory-wise forecasts.