Part 2: Practical Implementation

Task 2: AI-Driven IoT Concept

- Scenario: Design a smart agriculture simulation system using AI and IoT.
- Requirements:
    - List sensors needed (e.g., soil moisture, temperature).
    - Propose an AI model to predict crop yields.
    - Sketch a data flow diagram (AI processing sensor data).

Part A : Sensors & Hardware

Environmental sensors

- Soil moisture (volumetric water content) – sensor: capacitive probe (e.g., VH400).
- Soil temperature – digital probe (DS18B20).
- Air temperature & humidity – DHT22 / SHT31.
- Solar irradiance / PAR – light sensor (BH1750 or quantum PAR sensor).
- Rain gauge / precipitation sensor.

Soil & plant health

- Soil electrical conductivity (salinity/N status) – EC probe.
- pH sensor (periodic measurement or lab-derived).
- Multispectral / RGB camera (for NDVI/vegetation indices and phenology).
- Leaf wetness sensor (disease risk proxy).

Operational / infrastructure

- Wind speed & direction (anemometer) – for spray drift / evapotranspiration.
- Water flow / valve sensors for irrigation (flow meter).
- Weight scale (for small greenhouse pots) or yield sensor (for harvest conveyor simulation).

Communications & compute

- Microcontroller nodes (ESP32 / Arduino) for low-power sensing.
- Edge gateway (Raspberry Pi or Jetson Nano) with Wi-Fi / LoRaWAN / NB-IoT / LTE backhaul.
- Optional: Solar + battery for field deployment.

Part B : AI model proposal to predict crop yields

Practical hybrid architecture (cloud training, edge inference) that suits both simulation and real deployment.

Problem framing

- **Input:** Time series of sensor readings (soil moisture, temp, humidity, irradiance, EC), imagery-derived indices (NDVI, GNDVI), management actions (irrigation events, fertilizer doses), weather forecasts.
- **Output:** Per-plot yield estimate (kg/ha) at harvest time or short-term yield-change prediction.

Model options (recommended)

Primary (practical & accurate): Gradient-boosted trees on engineered features

- Model: XGBoost or LightGBM (works very well on tabular agronomic data).
- Why: fast training, handles missing data, interpretable (feature importance), small footprint for edge if needed (via Treelite or converting to ONNX/TFLite via surrogate).
- Inputs: aggregated/time-window features (weekly averages, cumulative GDD, days since last irrigation, max/min/mean sensor readings, rolling NDVI stats).

Advanced (spatio-temporal): Seq2One LSTM / TCN / Transformer

- Model: Temporal Convolutional Network (TCN) or LSTM with imagery embedding.
- Why: direct modeling of sequence dynamics; good when you have long time series and many temporal dependencies.
- Combine imagery: use a small CNN (e.g., MobileNetV2) to extract image features per date, then feed into TCN/LSTM along with tabular sensors.

Hybrid deployment pattern

- **Train** large model in cloud (uses TCN + CNN for best accuracy).
- **Export** distilled/lightweight model (XGBoost or small MLP) for edge inference (TFLite or ONNX). Distillation: train small model to mimic outputs of big model.

Feature engineering (examples)

- Cumulative Growing Degree Days (GDD) to date.
- Soil moisture anomaly = current − seasonal mean.
- Rolling mean & std (3/7/14 days) of NDVI.

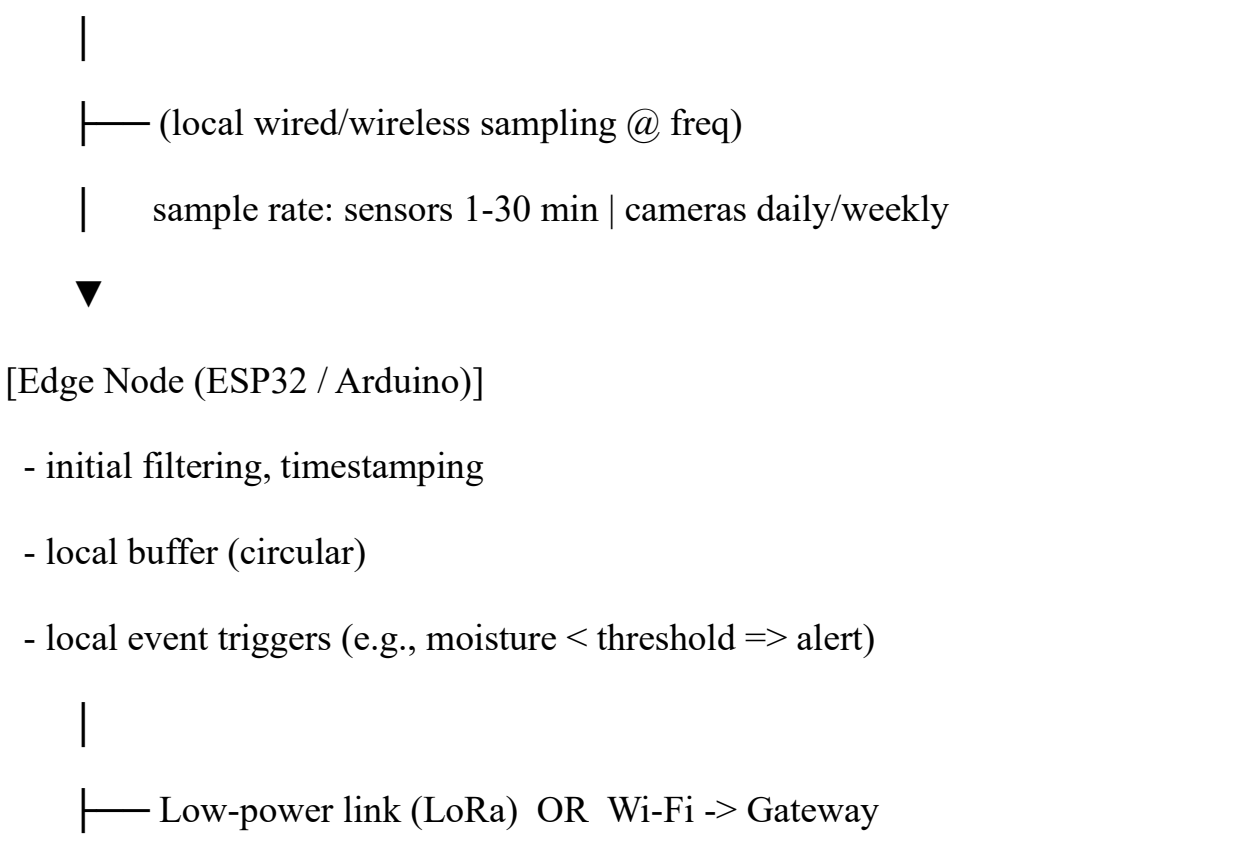
- Days since last fertilization/irrigation.
- Forecasted rainfall next 7 days (from API).
- Categorical: crop variety, planting density, plot ID.

Loss & evaluation metrics

- **Regression metrics:** MAE, RMSE, R^2.
- **Real-world:** percent error, calibration plots, confusion if using yield classes.
- **Cross-validation:** time-series split (walk-forward) not random K-fold.

Part C — Data flow diagram (text/sketch)

A simple diagram you can paste into docs or render in ASCII/markdown.



▼

[Edge Gateway (Raspberry Pi / Jetson / Pi + USB-cam)]

- aggregate sensor streams
- preprocess: rescale, fill-missing, compute rolling features
- local inference: run TFLite model (fast), produce per-plot estimate & alerts
- cache logs, send periodic batches

│

├── Secure uplink (MQTT / HTTPS / NB-IoT)

▼

[Cloud / Central Server]

- long-term storage (time-series DB: InfluxDB / Postgres)
- heavy training (XGBoost / TCN + CNN), retraining pipelines
- model registry, versioning
- opportunity for federated learning / aggregation

│

├── Web Dashboard / Mobile App

│ - visualize soil maps, predicted yields, alerts, irrigation recommendations

└── Farmer / Agronomist

(If you prefer a diagram image, I can generate a simple SVG/PNG layout.)

Part D — Implementation & simulation steps (high-level)

1. **Collect & simulate data**
    - For prototype, simulate sensor streams in Colab or local Python using historical datasets or simple physics (soil moisture = evapotranspiration model + irrigation events + noise).
    - For imagery, use small sample images or augment.
2. **Preprocess & label**
    - Build per-plot time-series CSVs.
    - Label with actual yields (if you have) or simulated yield function.
3. **Feature engineering**
    - Compute rolling statistics, cumulated GDD, NDVI summaries, event counts (irrigations).
4. **Train & validate**
    - Baseline: LightGBM/XGBoost on engineered features.
    - Advanced: LSTM/TCN + CNN if you have imagery & longer sequences.
5. **Export model**
    - Best model → cloud for production.
    - Distill or export lighter model for edge (convert to TFLite or ONNX).
6. **Deploy**
    - Install TFLite runtime on Raspberry Pi.

- Edge gateway reads sensors, runs inference, displays local actions (irrigation on/off).
- Periodic uploads of raw data/reports for retraining.

7. **Monitor**
    - Track drift, retrain monthly/seasonally.

Part E — Practical parameters & tips

- **Sampling rates:** soil moisture 15–60 min; temp/humidity 5–30 min; camera daily; EC/pH daily to weekly.
- **Data retention:** keep high-frequency raw for 2–4 weeks, aggregated weekly data long-term.
- **Security:** use TLS for cloud comms; sign firmware; rotate keys.
- **Latency needs:** local irrigation decisions => <1s (edge), yield prediction is offline (hours/days) so cloud OK.

Part F — Example minimal model pipeline (pseudo)

1. Read time-series sensors (pandas)
2. Compute features: rolling means, cumulative GDD, days since irrigation
3. Train LightGBM on train set, validate with time-series split
4. Export model: model.save_model('yield_lgb.txt')
5. Convert distilled model for edge inference:

- Use Treelite/ONNX or a tiny TensorFlow model -> TFLite

Part G — Deliverables you can hand in

- Design.md — sensors, sampling plan, deployment notes (use above).
- Model.ipynb — simulated data generation → feature engineering → lightgbm training → metrics (mae, rmse, r^2).
- Edge_deploy.py — simple script that loads tflite model and consumes a local csv stream (simulates raspberry pi).
- Diagram.png or ascii diagram (above).
- Short report (1–2 pages) explaining edge vs cloud roles, dataset needs, and evaluation plan.