If you want, I can also generate:

- A **project architecture diagram (DL vs FL)**
- A **flowchart** for your report
- A **viva-ready explanation** in 10 lines
- A **folder structure section** for your documentation

Just tell me **what you need next**!

- A project architecture diagram (DL vs FL)
- A flowchart for your report
- A viva-ready explanation in 10 lines
- A folder structure section for your documentation

give all

Project architecture diagram (DL vs FL)

```sql
                                     +--------------------+
                                     |    HAM10000 RAW    |
                                     | images + metadata  |
                                     +---------+----------+
                                               |
                            (ham_sort.py — sort images into class folders)
                                               |
                             +-----------------v-----------------+
                             |         ham10000/sorted/          |
                             |  class subfolders (akiec, bcc…)   |
                             +-----------------+-----------------+
                                               |
                   +---------------------------+---------------------------+
                   |                                                       |
         (Centralized DL)                                           (Federated FL)
                   |                                                       |
  +----------------v---------------+                      +----------------v----------------+
  | trainModel.py (CNN + SPP)      |                      | split_data.py -> data/clients/  |
  | - builds CNN+SPP               |                      | - client_1/, client_2/, ...     |
  | - trains on ham10000/sorted    |                      +----------------+-----------------+
  | - saves models/skin_lesion_model.h5                               |
  +----------------+---------------+                                   |
                   |                                                   |
                   |                                                   |
            models/skin_lesion_model.h5                 federated_simulation.py (server)
                   |                                                   |
                   |                                           uses resnet50_fl.py
                   |                                       (model architecture for clients)
           app.py (Flask) <----------------------------------+------------------------+
           - loads centralized DL model                       |                        |
           - loads federated_global_model.keras               |                        |
           - provides UI (upload, model selector)             |                        |
           - blockchain login (MetaMask)                      |                        |
                                                             |                        |
                                               Each client: local training on client_i/   |
                                               -> returns weights -> server FedAvg -> global
model
                                                             |
                                                     models/federated_global_model.keras
                                                             |
                                                        app.py (can serve predictions)
```

Flowchart for your report (step-by-step)

1. **Data preparation**
   Run `python3 ham_sort.py` to organize HAM10000 into `ham10000/sorted/`.
2. **Create FL clients**
   Run `python3 split_data.py --source ham10000/sorted --out ./data --clients 3` to create `data/client_1`, `data/client_2`, `data/client_3`.
3. **Train centralized model (optional / baseline)**
   `python3 trainModel.py` → produces `models/skin_lesion_model.h5`.
4. **Run federated simulation**
   `python3 federated_simulation.py --data_dir data --rounds 6 --local_epochs 2` → produces `models/federated_global_model.keras`.
5. **(Optional) Fine-tune global model centrally**
   Load `federated_global_model.keras`, unfreeze top layers, and fine-tune on `ham10000/sorted`.
6. **Start web app**
   `python3 app.py` (or `python3 web/app.py` if moved) — UI provides MetaMask login and prediction page.
7. **Demo flow**
   - Login via MetaMask → upload image → choose model (Centralized / Federated) → Analyze → view result + confidence.
8. **Evaluation & plots**
   Show global accuracy per round, per-client accuracy curves, and loss curves (saved by `federated_simulation.py`).
9. **Discussion & interpretation**
   Compare centralized vs federated model accuracy, discuss privacy benefits and trade-offs.
10. **Conclusion**
    Emphasize privacy-preserving training, outcome, and possible next steps (more rounds, fine-tuning, deploying).

Viva-ready explanation (10 concise lines — use exactly in viva)

1. We implemented a skin-lesion classifier and demonstrated Federated Learning with Federated Averaging.
2. The HAM10000 dataset is first organized into class folders and optionally split into client datasets.
3. The centralized baseline is trained with a CNN + Spatial Pyramid Pooling and saved as `skin_lesion_model.h5`.
4. For FL, each client uses the same ResNet50-based architecture (`resnet50_fl.py`) and trains only on local data.
5. After every round, the server collects client weights and performs weighted averaging (FedAvg) to update the global model.

6. Data never leaves a client — only model weights are shared, preserving privacy for each hospital's images.
7. We monitor per-client training curves and global validation accuracy per round to assess convergence.
8. The final FL global model is saved as **`federated_global_model.keras`** and can be served via Flask.
9. In the Flask app the user can choose Centralized or Federated model for prediction to compare outputs live.
10. This setup demonstrates privacy-preserving collaborative learning while achieving competitive performance with central training.

Folder structure section for your documentation

```python
skin_cancer_detection/          # project root
├─ app.py                       # Flask app (login + prediction UI)
├─ trainModel.py                # Centralized DL training (CNN + SPP)
├─ federated_simulation.py      # FL simulation (clients, FedAvg, plots)
├─ resnet50_fl.py               # FL ResNet50 architecture (used by federated_simulation.py)
├─ ham_sort.py                  # Sort HAM10000 metadata -> class folders
├─ split_data.py                # Split sorted data into client_i folders for FL
├─ models/                      # Saved models
│   ├─ skin_lesion_model.h5     # Centralized model (baseline)
│   └─ federated_global_model.keras  # Federated global model
├─ data/                        # FL client datasets (created by split_data.py)
│   ├─ client_1/
│   ├─ client_2/
│   └─ client_3/
├─ ham10000/                    # Original dataset (sorted version kept here)
│   └─ sorted/                  # class subfolders used for centralized training
├─ templates/                    # HTML templates for Flask UI
├─ uploads/                      # Uploaded images (created at runtime)
├─ samples/                      # Small demo images (optional)
├─ requirements.txt              # Python dependencies
└─ README.md                     # Project description & how to run
```

Notes:

- Keep **`ham10000/`** outside Git or store a small sample; large datasets should not be committed.
- **`data/`** is generated by **`split_data.py`** and is required for FL simulation.
- **`uploads/`** is runtime — ensure it exists and is writable by the Flask process.

If you want, I can:

- export the architecture diagram as an image file (PNG) ready to include in your report, or
- make a prettier flowchart (Mermaid or Visio-style) you can paste into your slides — tell me which format you prefer.