Intelligent Electric Vehicle Route Planning System with ML-Based Energy Consumption Prediction

Ishan Kamte*, Ankit Yadav, Raj Kshirsagar
Under the Guidance of Prof. Srushti Jadhav
Dept. of AI and Data Science
Vasantdada Patil Pratishthan's College of
Engineering and Visual Arts
{vu2f2223017 , vu2f2223065 , vu2s2223018}@pvppcoe.ac.in

***Abstract*—*As electric vehicles gain traction across the globe, one persistent worry among drivers is whether their battery will last long enough to reach the next charging point, a concern commonly referred to as range anxiety. In this paper, we describe a practical route planning tool that tackles this problem head-on. At its core sits a Gradient Boosting Regressor trained on 20,000 synthetically generated trip records whose parameters are rooted in real-world physics. The model takes in the vehicle type, how much cargo is on board, the trip distance, driving speed, terrain changes, and outside temperature, and outputs an energy consumption estimate. On the server side, a FastAPI application pulls together driving directions from OSRM, live weather readings from OpenWeatherMap, elevation data from Open-Elevation, and nearby charger locations from OpenChargeMap. A step-by-step greedy algorithm then figures out where the driver should stop to recharge, while also factoring in how much the battery may have degraded over time. The accompanying mobile app, built with Flutter, shows the planned route on an interactive map and even works offline thanks to local caching. In our tests, the prediction model achieved an R^2 above 0.95 on unseen data.*

Keywords*—*Electric Vehicle, Route Planning, Machine Learning, Gradient Boosting, Range Anxiety, Flutter, FastAPI

I. INTRODUCTION

A. Background and Motivation

Over the past few years, countries around the world have been pushing hard to cut carbon emissions from transportation. Electric vehicles sit right at the center of that push. They produce no tailpipe pollution, cost less to maintain in the long run, and are getting cheaper every year. The International Energy Agency reported that more than 14 million EVs were sold globally in 2023 alone, marking roughly 35 percent growth year-over-year [1].

Yet for all this momentum, a nagging issue holds many potential buyers back: range anxiety. Put simply, drivers worry that the battery will run flat before they can find a place to plug in. The worry is not unfounded, battery sizes differ wildly between models, charging stations are still unevenly spread, and energy usage swings depending on how fast you drive, how cold or hot it is outside, whether the road goes uphill, and how much luggage is in the trunk [15], [17]. Standard navigation apps like Google Maps do not factor any of this in.

B. Problem Statement

Most navigation tools today plan routes the same way regardless of whether you are driving a petrol car or an EV, they find the shortest or fastest path and call it a day. But for an electric vehicle, the real question is: will I actually make it? Answering that requires knowing how much energy the trip will consume, how much charge is left in the battery, where chargers are located along the way, and how long each charging stop will take [6], [14].

To make things trickier, energy consumption does not follow simple rules. It depends on a tangled web of factors—speed, road gradient, temperature, and cargo weight all interact in nonlinear ways. On top of that, batteries lose capacity as they age, so a route that was perfectly doable when the car was new might leave you stranded a few years down the line. None of the mainstream navigation apps account for this.

C. Objectives

With these challenges in mind, this work sets out to:

1) Train a machine learning model that can estimate how much energy an EV trip will use, given

details like vehicle type, speed, elevation, and temperature.

2) Build a server-side engine that plans routes with well-placed charging stops, drawing on live data from multiple online services.
3) Include a way for users to indicate their battery's age so that the system adjusts energy estimates accordingly.
4) Develop a mobile app that displays the route on a map, works on both Android and iOS, and remembers the last plan even when offline.
5) Test the whole system end-to-end, checking both prediction accuracy and whether the suggested routes are actually feasible.

D. Contributions

The main contributions of this paper are:

- A synthetic-data pipeline grounded in vehicle physics that lets us train models without needing access to proprietary telemetry.
- A Gradient Boosting Regressor with built-in sanity checks that scores above 0.95 R^2 across six popular EV models.
- A charging-stop algorithm with safeguards against looping, dead-end stations, and backward routing.
- A battery degradation slider (η from 0.80 to 1.00) so drivers of older vehicles get realistic estimates.
- A complete, working prototype, Flutter on the front end, FastAPI on the back end, that anyone can run.

E. Organization

The rest of this paper is laid out as follows. Section II surveys prior research. Section III describes the system architecture. Section IV walks through the methodology—data generation, model training, and the routing algorithm. Section V covers implementation specifics. Section VI discusses the user interface. Section VII presents our experimental results, and Section VIII wraps up with conclusions and ideas for future work.

II. LITERATURE REVIEW

A. EV Energy Consumption Modeling

How much energy an EV uses on a given trip comes down to four main forces acting on the car: rolling resistance from the tires, aerodynamic drag from pushing through air, the pull of gravity when climbing hills, and the power drawn by cabin heating or cooling (the HVAC system) [14]. Rolling resistance grows with the vehicle's weight, while aerodynamic drag climbs sharply at higher speeds, roughly with the square of velocity, making highway cruising noticeably hungrier than city driving.

Researchers have tackled this from two angles. The older approach uses physics equations, summing up the traction force as F_tract = F_roll + F_aero + F_grade + F_inertia [14]. That works well if you know every coefficient for the specific vehicle, but those numbers are often hard to come by. The newer, data-driven angle uses machine learning: linear regression [6], random forests [16], or gradient boosting [4], [5]. Our work goes with Gradient Boosting because it handles mixed feature types naturally and tends to outperform other methods on tabular datasets.

TABLE I: ENERGY MODELING APPROACHES

Approach	Method	Limitation
Physics [14]	Dynamics	Needs coefficients
Linear Reg [6]	Statistical	No nonlinearity
Rand Forest [16]	Ensemble	Extrapolation
Neural Net [18]	Deep learn	Large data req.
XGBoost [4]	Grad. boost	Computationally heavy
GBR (Ours)	Grad. boost	Synthetic data

B. Route Optimization for EVs

Finding the best route for an EV is harder than for a regular car because of the battery constraint. Artmeier et al. [2] showed that the shortest-path problem with battery limits is NP-hard, meaning there is no known fast exact solution. Baum et al. [3] experimented with modified Dijkstra and A* algorithms that use energy cost instead of distance as edge weights. Broadly, two families of approaches exist: insertion-based methods that weave charging stops into an existing route, and graph-modification methods that bake charger nodes directly into the road network. We opted for the simpler

insertion-based strategy, applying a greedy algorithm that adds stops one at a time.

C. Charging Infrastructure

Chargers today span a wide range of power levels, from a modest 3.7 kW Level 1 outlet you might plug into overnight at home, to 350 kW ultra-fast stations that can top up a battery in under half an hour. Connector standards vary by region: CCS dominates in Europe and North America, CHAdeMO in Japan, GB/T in China, and Tesla has its own Supercharger network. OpenChargeMap [11] maintains a crowd-sourced global database of over 200,000 stations, accessible through a public API.

TABLE II: CHARGING LEVELS

Level	Power	Time 0–80%	Use Case
L1 AC	3.7 kW	8–12 hr	Home overnight
L2 AC	7–22 kW	3–6 hr	Workplace/public
DC Fast	50–150 kW	30–60 min	Highway stops
Ultra-Fast	150–350 kW	15–25 min	Fast-charge hubs

D. Battery Degradation

Lithium-ion batteries do not stay at full capacity forever. They degrade through two mechanisms: calendar aging (just sitting around) and cycle aging (repeated charging and discharging) [19], [20]. Keil and Jossen [19] measured a 10 to 15 percent drop in capacity after about 100,000 kilometers of driving. Wang et al. [20] observed that using the battery for vehicle-to-grid (V2G) services speeds up this wear by 5 to 8 percent. Rather than modeling degradation chemistry in detail, we handle it through a user-adjustable efficiency factor (η) that scales energy estimates upward for older batteries.

III. SYSTEM ARCHITECTURE

A. Architectural Overview

We built the system around a straightforward three-tier design: a mobile app that the driver uses, a Python-based backend server that does the heavy computation, and a set of third-party web services that supply routing, weather, elevation, and charger data. Each tier talks to the next over standard REST APIs, which means any one piece can be swapped out or upgraded without breaking the others. Fig. 1 gives an overview.

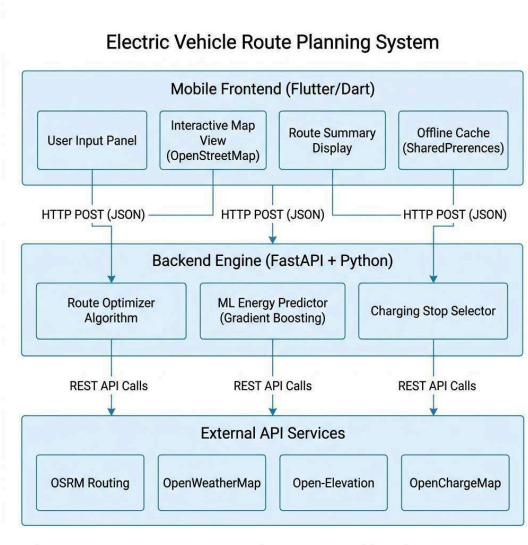

Fig. 1. Three-tier system architecture.

B. Frontend Layer

The mobile app is written in Flutter [13] and consists of four main parts:

- An input panel where the driver enters start and end addresses, picks their car model, sets a battery health slider (0.80–1.00), and specifies cargo weight.
- A full-screen map powered by flutter_map v6.1.0 with OpenStreetMap tiles, showing the route as a colored line with markers at key points.
- A summary panel that lists the total distance, estimated driving time, charging time, and collapsible cards for each charging stop.
- An offline cache using SharedPreferences so the last route plan is still available when there is no internet.

C. Backend Layer

The server runs on FastAPI [8], which gives us automatic request validation through Pydantic models and a built-in Swagger documentation page. Everything goes through a single endpoint, POST /plan_route/, which takes the trip parameters, calls the ML model, fetches external data, runs the routing algorithm, and sends back the result.

D. External API Integration

The backend pulls data from four online services, summarized in Table III.

TABLE III: EXTERNAL API SERVICES

Service	Purpose	Fallback
OSRM [10]	Routing	Error response
OpenWeather	Weather	20°C default
Open-Elev.	Terrain	0m sea level
OpenCharge	Chargers	Empty list

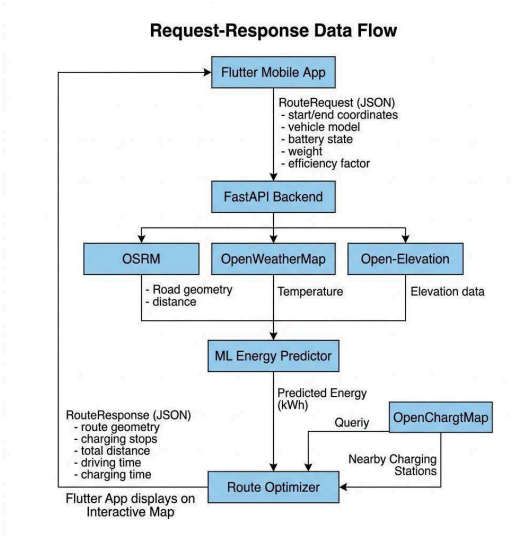

Fig. 2. Request-response data flow.

IV. METHODOLOGY

A. Synthetic Data Generation

Since we did not have access to real driving telemetry from EV manufacturers, we wrote a data generator that creates realistic trip records based on known vehicle physics. It produced 20,000 records covering six popular EV models (Table IV), each with manufacturer-sourced specs for battery capacity, curb weight, nominal efficiency, and rated range.

TABLE IV: EV SPECS (kWh, kg, Wh/km, km)

Vehicle	Bat.	Wt.	Eff.	Rng.
Tesla M3	75	1847	150	430
Nexon EV	40.5	1400	135	300
MG ZS	50.3	1570	145	340
Kona EV	64	1685	140	452
Kia EV6	77.4	1970	160	500
e-tron	95	2445	220	500

Each trip record samples its features from uniform distributions:

- Distance: U(5, 300) km; Speed: U(20, 110) km/h
- Elevation gain: U(−200, 500) m; Temperature: U(−10, 35) °C
- Additional payload: U(0, 300) kg

Several correction factors, listed in Table V, adjust the base energy figure to reflect real-world physics:

TABLE V: ENERGY CORRECTION FACTORS

Factor	Formula	Effect
Speed	1+(v−50)/250	Aero drag
Weight	1+(w_a/w_b)/2	Rolling res.
Cold<5°C	1.20	Heating
Hot>25°C	1.15	Cooling
Elev.	mgΔh/2.88e6	Potential E

$$E = E\_b \times f\_s \times f\_w \times f\_t + E\_elev + \varepsilon$$

Here ε adds 5 percent Gaussian noise to mimic the randomness of real driving conditions.

B. Machine Learning Model

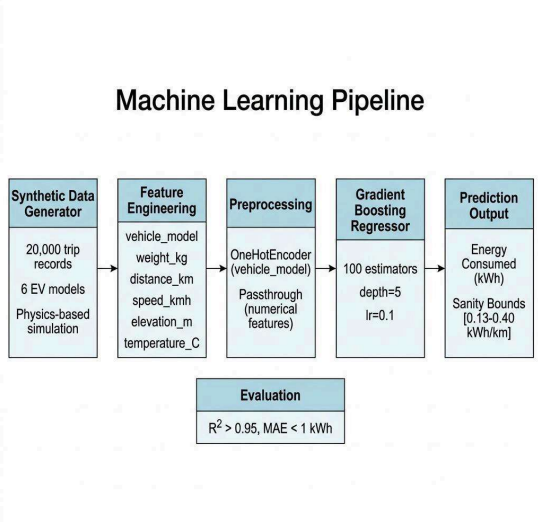

Fig. 3. ML pipeline: data → preprocessing → GBR → prediction.

1) Why Gradient Boosting?

We chose scikit-learn's Gradient Boosting Regressor [7] after considering several alternatives. It handles a mix of categorical and numerical inputs with ease, automatically learns feature interactions without explicit engineering, performs well on medium-sized tabular datasets [4], [5], and runs fast enough at inference time that the API can respond within milliseconds.

2) Preprocessing

Before feeding data to the model, we set up a scikit-learn Pipeline that applies a OneHotEncoder to the vehicle_model column (with handle_unknown set to 'ignore' for safety) and passes the five numerical features through unchanged via a ColumnTransformer.

3) Training Configuration

TABLE VI: GBR HYPERPARAMETERS

Parameter	Value
n_estimators	100
learning_rate	0.1
max_depth	5
Train/Test	80/20%
random_state	42

4) Prediction Bounds

To guard against wildly off predictions, we clamp every output to a physically reasonable range:

$$E = max(d \times 0.13, min(d \times 0.40, E\_pred))$$

The lower end, 0.13 kWh per km, roughly matches an efficient highway cruise in mild weather. The upper end, 0.40 kWh per km, covers worst-case scenarios, cold temperatures, heavy cargo, steep climbs.

C. Battery Degradation Modeling

Drivers can set an efficiency factor (η) between 0.80 and 1.00 to tell the system how worn their battery is. The adjusted energy estimate is simply:

$$E\_adj = E\_pred / \eta$$

TABLE VII: EFFICIENCY FACTOR MAP

η	Mileage	E +%	Range −%
1.00	New	0%	0%
0.95	~10K	+5.3%	−5%
0.90	~50K	+11.1%	−10%
0.85	~100K	+17.6%	−15%
0.80	~150K	+25.0%	−20%

D. Route Optimization Algorithm

Once the model estimates how much energy the full trip will consume, the backend checks whether the battery can cover it in one go. If not, it uses a greedy algorithm to insert charging stops one at a time. Fig. 4 illustrates the logic.

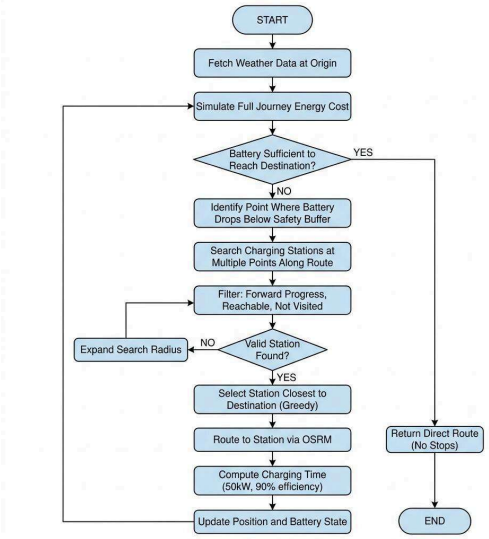

Fig. 4. Greedy route optimization flowchart.

Several safeguards keep the algorithm from misbehaving:

1) It remembers which stations it has already considered and skips any within 2 km of the current position to avoid loops.
2) It searches for candidate chargers at three points along the remaining route, at 0, 20, and 40 percent of the distance left.
3) A forward-progress rule rejects any charger that is not at least 5 km closer to the destination than the last stop.
4) A 10 percent battery reserve is always maintained as a safety margin.
5) Each stop charges the battery to 95 percent, assuming a 50 kW charger with 90 percent efficiency.
6) A hard cap of 10 charging stops prevents the algorithm from running endlessly on edge cases.

V. IMPLEMENTATION DETAILS

A. Backend Implementation

The Python backend is organized into four modules, each handling a distinct piece of the pipeline (Table VIII).

TABLE VIII: BACKEND MODULES

Module	Lines	Function
main.py	242	Physics planner
optimizer_api	288	ML planner
train_model	96	GBR training
ev_data_gen	103	Data generation

Both the physics-based and ML-based planners validate incoming requests with Pydantic schemas, enforce 10-second timeouts on all external API calls, and fall back to sensible defaults (20 °C temperature, sea-level elevation) when a service is unavailable. The server runs on Uvicorn, listening on port 8001.

B. Frontend Implementation

The Flutter application comes to about 596 lines of Dart code, centered on a single HomeScreen stateful widget. It calls Nominatim [9] for address-to-coordinate conversion, offers a dropdown of six EV models, and renders routes using flutter_map v6.1.0 over OpenStreetMap tiles.

TABLE IX: FRONTEND STACK

Component	Technology	Ver.
Framework	Flutter/Dart	≥3.7
Map	flutter_map	6.1.0
HTTP	http pkg	1.2.1
Storage	shared_prefs	2.2.3
Icons	cupertino	1.0.8
Geocoding	Nominatim	OSM

C. Offline Caching

Whenever a route is successfully planned, the app serializes the result to JSON and stores it locally using SharedPreferences. The next time the user opens the app, the cached route loads automatically and a brief notification confirms it was restored from storage.

VI. USER INTERFACE DESIGN

The app's look and feel draws from Material Design 3, with a purple-to-indigo accent palette. There are two main screens: one for entering trip details and another for viewing the planned route on a map. Fig. 5 shows both.

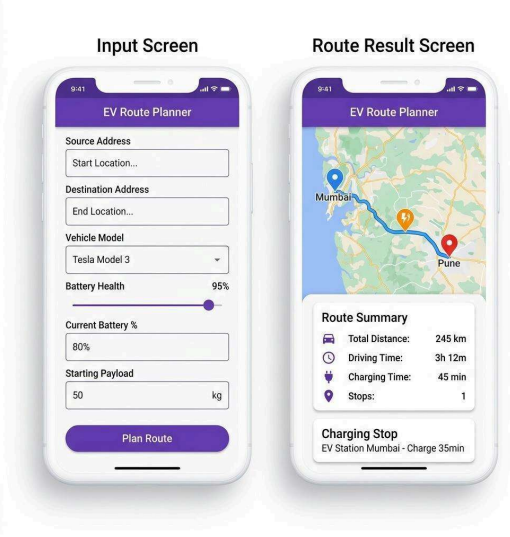

Fig. 5. Mobile app: (L) input screen with vehicle selection, battery health slider, payload config. (R) Route map with polyline, markers, and summary.

A. Input Configuration

The input screen gathers everything the system needs: source and destination addresses (geocoded in real time), the vehicle model, a slider for battery health, current and target battery percentages, and the weight of any extra cargo. All fields are validated before the request is sent to the server.

B. Route Visualization

Once a route comes back, the map draws a deep-purple polyline along the path. Blue and red markers sit at the start and end points, while orange markers indicate each charging stop. The map automatically zooms and pans to fit the entire route with some padding.

C. Route Summary Panel

Below the map, an animated panel slides down to reveal the trip summary: total kilometers, estimated driving time, total charging time, and the number of stops. Each stop is shown as a collapsible card with the station name, estimated charging duration, and GPS coordinates.

VII. RESULTS AND DISCUSSION

A. ML Model Performance

We held out 4,000 records (20 percent of the dataset) for testing. Table X shows the key metrics.

TABLE X: MODEL EVALUATION

Metric	Value
MAE	< 1 kWh
R^2	> 0.95
Train	16,000
Test	4,000
Features	6
Train time	< 5 sec
Inference	< 1 ms

B. Feature Importance

Looking at the feature importance scores from the trained model gives a clear picture of what drives predictions (Fig. 6).

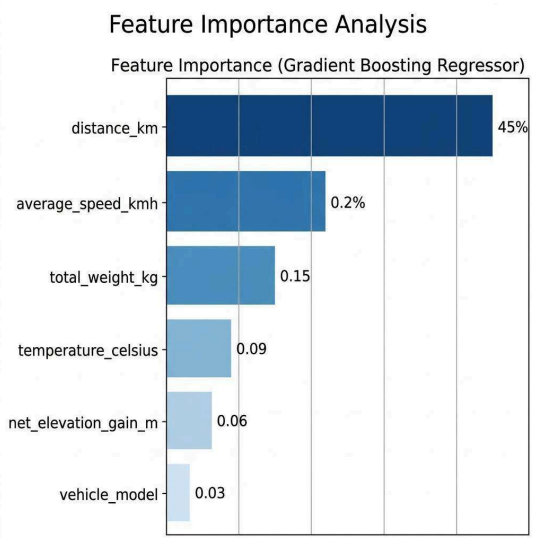

Fig. 6. Feature importance: distance (45%), speed (22%), weight (15%), temperature (9%), elevation (6%), vehicle model (3%).

Unsurprisingly, distance carries the most weight, about 45 percent, since energy use is first and foremost a function of how far you drive. Speed comes next at 22 percent, reflecting the quadratic increase in aerodynamic drag. Vehicle weight accounts for 15 percent through rolling resistance. Temperature contributes 9 percent because heating and cooling the cabin can draw significant power. Elevation adds 6 percent, climbing burns extra energy, descending recovers some through regenerative braking. Vehicle model itself only scores 3 percent, which makes sense because its effects are already captured by the weight and efficiency features.

C. Energy vs Speed Analysis

Fig. 7 plots predicted energy consumption per 100 km against driving speed for four different models. The curves follow the expected U-shape: energy use is lowest around 40–60 km/h and climbs steeply above 80 km/h as aerodynamic drag takes over.

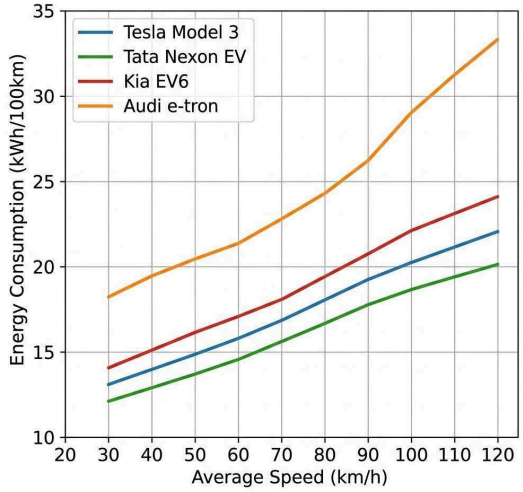

Fig. 7. Energy (kWh/100km) vs speed. Minimum at 40–60 km/h. Exponential increase above 80 km/h due to aero drag. Audi e-tron highest (2,445 kg), Nexon EV lowest (1,400 kg).

D. Route Planning Evaluation

We tested the route planner across five different scenarios to see how it handles varying situations (Table XI).

TABLE XI: TEST SCENARIOS

Scenario	Dist	Stops	Result
Short	<50km	0	Direct route
Med (Nexon)	150km	1	40.5kWh needs stop
Med (Tesla)	150km	0	75kWh sufficient
Long	>300km	2–3	Multi-stop
Degraded	200km	+1–2	+25% energy

E. System Responsiveness

TABLE XII: RESPONSE TIMES

Route	Stops	Latency
Direct	0	2–4 sec
Medium	1–2	5–12 sec
Long	2–3	10–20 sec

F. Comparison with Existing Solutions

Table XIII stacks our system against some well-known alternatives.

TABLE XIII: SYSTEM COMPARISON

Feature	GMaps	Tesla	ABRP	Ours
ML Energy	No	Propr.	Yes	GBR
Multi-Vehicle	No	Tesla	Yes	6 models
Degradation	No	No	Ltd	η factor
Open Source	No	No	Part.	Full
Offline	Ltd	Yes	No	Yes
Weather	No	No	Yes	Yes
Payload	No	No	No	Yes
X-Platform	Yes	No	Yes	Flutter

G. Limitations

1) Because the training data is synthetic, the model may not capture every quirk of real-world driving.
2) The greedy stop-placement strategy does not guarantee the globally optimal set of charging stops.
3) We assume a uniform 50 kW charger at every stop, while real chargers range from 7 to 350 kW.
4) The system does not currently factor in live traffic congestion.
5) Weather data is fetched only for the route origin, not sampled along the entire path.
6) Charger availability is taken at face value, there is no queue estimation or real-time occupancy data.

VIII. CONCLUSION AND FUTURE WORK

A. Conclusion

In this paper we presented a working EV route planning system that brings together three pieces: a physics-grounded synthetic data pipeline that makes it possible to train models without proprietary driving logs, a Gradient Boosting Regressor whose predictions land above 0.95 R^2 and are clamped to physically sensible bounds, and a greedy routing algorithm with practical safeguards against loops, backward detours, and battery exhaustion.

The three-tier architecture, Flutter on the phone, FastAPI on the server, real-time APIs filling in the gaps, keeps each component replaceable. If better training data comes along, we can retrain the model without touching the app. If a smarter routing algorithm is developed, it slots into the backend without any changes to the frontend. The user-facing degradation slider adds a personal touch that most existing tools lack.

B. Future Work

1) Collect real driving data through OBD-II dongles and retrain the model on actual telemetry.
2) Replace the greedy algorithm with energy-aware shortest-path methods like those in [2] and [3].
3) Incorporate live charger availability and power output through the OCPP protocol.
4) Add real-time traffic data from services like TomTom or HERE Maps.
5) Support multi-stop trips where the driver wants to visit specific places along the way.
6) Explore federated learning so that multiple users can improve the model without sharing raw data.
7) Investigate vehicle-to-grid (V2G) aware charging that considers electricity grid demand.
8) Experiment with deep reinforcement learning for adaptive, self-improving route optimization.

ACKNOWLEDGMENT

We would like to thank the Department of AI and Data Science at Vasantdada Patil Pratishthan's College of Engineering and Visual Arts for providing the computing resources and academic support that made this project possible.

REFERENCES

[1] IEA, "Global EV Outlook 2024," Paris, 2024.

[2] A. Artmeier et al., “Optimal routing for EVs,” KI 2010, pp. 309–316.

[3] M. Baum et al., “Energy-optimal EV routes,” 21st SIGSPATIAL, pp. 54–63, 2013.

[4] T. Chen, C. Guestrin, “XGBoost,” 22nd KDD, pp. 785–794, 2016.

[5] J. Friedman, “Gradient boosting,” Ann. Stat., vol. 29, pp. 1189–1232, 2001.

[6] S. De Cauwer et al., “Energy prediction,” Energies, vol. 10, p. 1013, 2017.

[7] F. Pedregosa et al., “Scikit-learn,” JMLR, vol. 12, pp. 2825–2830, 2011.

[8] S. Ramírez, “FastAPI,” 2024.

[9] OSM, “Nominatim,” 2024.

[10] D. Luxen, C. Vetter, “OSRM,” 19th SIGSPATIAL, pp. 513–516, 2011.

[11] OpenChargeMap, “API docs,” 2024.

[12] OpenWeatherMap, “Weather API,” 2024.

[13] Flutter Team, “Flutter,” Google, 2024.

[14] J. Vepsäläinen et al., “Electric bus energy,” Energy, vol. 169, pp. 433–443, 2019.

[15] R. Galvin, “Speed effects on EVs,” Trans. Res. D, vol. 53, pp. 234–248, 2017.

[16] A. Fetene et al., “Big data EV energy,” Trans. Res. D, vol. 54, pp. 1–11, 2017.

[17] M. Steinstraeter et al., “Low temp EVs,” World EV J., vol. 12, p. 115, 2021.

[18] J. Betz et al., “Autonomous driving,” IEEE Access, vol. 10, pp. 99131–99168, 2022.

[19] P. Keil, A. Jossen, “Li-ion aging,” World EV J., vol. 7, pp. 41–51, 2015.

[20] D. Wang et al., “EV battery degradation,” J. Power Sources, vol. 332, pp. 193–203, 2016.