Introduction

The consolidated report unites five complementary carbon-emissions tools, each targeting different sectors and use-cases. Collectively, they enable end-to-end carbon analytics aligned with ESG goals – measuring, modeling and forecasting emissions and suggesting reductions. For example, the CO_2 Emissions Dashboard "integrates a Long Short-Term Memory (LSTM) network…a Streamlit-based user interface… and a FastAPI backend" to forecast country-level emissions and estimate carbon credits. Similarly, the e-commerce pipeline ingests order/shipping data to compute per-order emissions (kg CO_2 per km), performs exploratory analysis by product and mode, and uses Prophet for time-series forecasting of daily CO_2 totals [1]. The farming model ingests FAOSTAT agriculture data and trains a Random Forest to identify key emission drivers (e.g. CO_2 per capita) supporting sustainable agriculture policy. A Python CLI tool performs on-the-fly carbon calculations for shipments. Finally, the **CarbonFootprintModel** module provides an extensible ML framework (preprocessing, model training, evaluation, and reduction strategy suggestions) for shipping emissions. Together, these tools cover Data Ingestion, Analysis, Prediction, and Visualization. They support environmental objectives (e.g. SDG 13 Climate Action) by revealing emissions hotspots and quantifying reduction potentials.

System Architecture Overview

The overall system architecture is modular, with data pipelines feeding analytical models and dashboards. **Data Ingestion:** The E-Commerce pipeline consumes the Olist dataset (orders, products, customers, geolocations) to compute shipping distances (using Haversine) and per-order emissions (weight × distance × mode factor). The Farming model reads FAOSTAT CSV data on emissions shares. The LSTM dashboard loads historical CO_2 data (entities × years). **Data Processing:** Shared utilities (e.g. `data_preparation`) scale/encode features (StandardScaler, OneHotEncoder) as needed. Each model constructs feature vectors – for example the `CarbonFootprintModel` creates an interaction term `weight_distance = Weight_in_gms * estimated_distance_km / 1000` [2]. **Models:** The pipeline uses diverse algorithms: Prophet for time-series forecasting (e-commerce), Random Forest for regression (farming, carbon-footprint module), and an LSTM neural network for sequential country-emissions forecasting. **Visualization/API:** Streamlit-based dashboards present outputs. The LSTM dashboard's front-end calls FastAPI for predictions, and plots historical trends and forecast points. A `run_dashboard.py` script simply sets the Python path and launches the Streamlit app (calling `streamlit run src/visualization/dashboard.py`) [3]. The CLI tool uses `click` to parse arguments and output a rich-formatted table. **Integration:** While each tool runs somewhat independently, they share the goal of quantifying and contextualizing carbon emissions. Together they form an end-to-end workflow: raw data → ML models → interactive visualizations → strategic insights.

Model Deep Dives

E-Commerce Carbon Footprint Pipeline

Purpose & ESG Relevance: This pipeline is designed to quantify and reduce logistics-related CO_2 in e-commerce. By analyzing per-order emissions, it highlights "carbon hotspots" (high-emission routes or products) and suggests greener practices (e.g. favor slower eco-shipping). These insights directly support corporate **Responsible Production/Consumption** goals by "cut[ting] GHG emissions, reduce[ing] returns, [and] lower[ing] waste" in online retail. It also emphasizes transparency in reporting *carbon intensity* (kg CO_2 per order or per km), aligning with industry best-practices and investor demand for green KPIs.

Data Sources & Structure: The pipeline uses the **Olist** e-commerce dataset (tables for orders, products, customers, geolocations). The DataCollector module merges these tables and computes delivery distances. Key computed features include shipping distance (Haversine on lat/long pairs) and shipping **method** (e.g. Express vs Standard, inferred from delivery time). Emissions per order are calculated as `weight_kg × distance_km × emission_factor` (kg CO_2), and a *carbon_intensity* metric (CO_2 per km) is added for benchmarking. This yields a tidy table: one row per order, with order attributes, volume, supplier region, shipping mode, distance, and CO_2 per order.

Technical Architecture: The modular pipeline comprises: - **DataCollector:** Ingests raw Olist CSVs, merges on keys, computes distances and emissions. - **CarbonAnalyzer (EDA):** Performs aggregation and filtering (by category, shipping method, region). It computes summary statistics (total, mean, max emissions) and flags the top percentile of orders by CO_2. Based on findings it generates textual recommendations (e.g. "shift from Express to Eco if Express dominates"), leveraging domain logic (e.g. consolidated delivery reduces emissions).
- **CarbonForecastModel (Predictive):** Aggregates daily CO_2 totals and fits a Prophet time-series model with yearly seasonality. It outputs future forecasts, cross-validation metrics, and anomaly detection. It also simulates "reduction scenarios" (e.g. forecast under 20–30% baseline cuts) and produces narrative suggestions (e.g. "redistribute deliveries to off-peak days").
- **Streamlit Dashboard:** A front-end (`dashboard.py`) visualizes all outputs. It shows total and per-day emissions, histograms of order CO_2, bar charts by product category/shipping mode, and time-series forecast plots. Interactive filters (by date range, category, etc.) let users drill down.

Model Choice & Justification: Prophet was chosen for its ease in capturing seasonality in time-series data and ability to handle irregular time series. It is a well-established forecasting tool with interpretable components (trend, season). RandomForest or other ML models were less applicable here since the data is naturally a time series (daily totals).

Training & Evaluation: The Prophet model is trained on historical daily CO_2 sums. Cross-validation (rolling origin) is used to evaluate forecast accuracy, with metrics like MAE/RMSE reported (exact values are context-dependent on dataset). The CarbonAnalyzer's recommendation logic is rule-based (no ML training). All code (data merges, Prophet fitting) is encapsulated in the pipeline modules.

Outputs, Insights & Visualizations: The dashboard presents: *distribution charts* of emissions by category/mode, *heatmaps* or *bar plots* of supplier region impacts, *histograms* of per-order CO_2, and the *forecast line plot* with historical vs. predicted emissions. It also highlights key metrics (e.g. total CO_2, average per-order).

Text panels display scenario outcomes and carbon credit estimates from reduced emissions. (The combination of detailed metrics and scenario charts empowers users to identify high-impact interventions.)

Carbon Credit Integration: While not explicitly computing credits, the pipeline shows emission reduction from scenarios. One could derive credits from the baseline vs. reduced forecast (e.g. projected tonnes saved). The emphasis on reduction scenarios implicitly relates to carbon-offset planning.

Limitations & Future Work: The pipeline assumes accurate distance and weight data; missing or outlier values could skew results. It treats all orders equally (no weighting for cargo vs. returns). Future enhancements could include: integrating more data (e.g. vehicle fleet emissions factors), advanced anomaly detection, and linking predicted reductions to actual credit markets. Adding uncertainty bands to forecasts and enabling multi-region comparisons would also strengthen ESG assessments.

Agricultural Emissions Model (FAOSTAT)

Purpose & ESG Relevance: This model analyzes farming-sector CO_2 emissions (from FAOSTAT) to identify what drives agricultural emissions in a country. Its goal is to inform sustainable agriculture practices and carbon-offset strategies. By revealing the most influential factors, it helps policymakers target high-impact areas (e.g. livestock vs. rice production). This supports Environmental goals by optimizing farm-level emissions and informs potential carbon credit programs (e.g. rewarding low-carbon farming).

Data Sources & Structure: The notebook loads FAOSTAT data (CSV) containing fields like **Domain** ("Emissions indicators"), **Element** (e.g. Emissions Share (CO2), CH4, N2O), **Item** (e.g. "Farm gate"), and emission values. For example, data for India (Area Code 356) includes CO_2-equivalent shares at farm gate. The script computes derived columns like `Emissions_per_area` and normalizes by GDP or population as needed. After cleaning (filtering to "Emissions Share (CO2eq)" and relevant items), the dataset is split into features (various emission shares and ratios) and target (`Total_Emissions_Share`).

```python
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestRegressor

# Preprocessing: scale numeric, encode categorical
numeric_feats = [...]           # e.g. % shares, per capita, etc.
categorical_feats = ['Item', 'Element']
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_feats),
    ('cat', OneHotEncoder(sparse=False, handle_unknown='ignore'),
categorical_feats)
])
rf_pipeline = Pipeline([
    ('preproc', preprocessor),
    ('model', RandomForestRegressor(n_estimators=100, random_state=42))
])
rf_pipeline.fit(X_train, y_train)
```

Technical Architecture (Model & Flow): The notebook builds a scikit-learn pipeline combining preprocessing and a RandomForestRegressor. Categorical features (like emission type) are one-hot encoded, while numeric indicators (e.g. `Emissions_per_capita`, `Emissions_per_area`) are standardized [2]. The model is trained on a random 80/20 split of the data. After training, feature importances are extracted to rank which inputs most affect farm emissions.

Model Choice & Justification: A Random Forest was chosen for its robustness to varied feature scales and its ability to capture nonlinear interactions among emission factors. It also naturally provides feature importance scores, aiding interpretability. A simpler linear model could miss complex relationships (e.g. how CO_2 share interacts with CH_4 share). No deep network was used due to limited data and the desire for explainability.

Training & Evaluation: The model is trained with 5-fold cross-validation to gauge stability. Metrics reported include RMSE and R^2 on held-out data. In the example run, the MSE was ~1.016 (MSE was printed in the notebook output) with high R^2, indicating good fit (though a caution noted: "one feature dominates" warning if >50% importance). The notebook also plots residuals and an error histogram to check for bias.

Outputs, Insights & Visualizations: Key outputs include: - **Feature Importances:** A bar chart of the top features (e.g. *Emissions per capita*, *Emissions Share (CO2eq)*, *Emissions per value*, etc.), as shown below. - **Data Distributions:** Histograms for variables like emissions per area reveal skewness or outliers.
- **Evaluation Plots:** Scatter of actual vs. predicted emissions and residual plots.

Figure: Distribution of agricultural emissions per area (histogram) from the FAOSTAT dataset.
Figure: Top feature importances from the Random Forest model. "Emissions per capita" emerges as the most influential predictor.

These figures highlight, for example, that emissions per capita is the largest driver. This suggests that policies reducing per-capita emissions (e.g. improving efficiency) could yield large gains. The long tail in the histogram shows some high-emission sectors (which could be targeted).

Carbon Credit Integration: Although the model itself doesn't directly compute credits, its insights can guide offset programs. For instance, if reducing per-capita emissions or CH_4 share can avoid *X* tonnes CO_2e, that potential can be translated into carbon credits for carbon farming initiatives. The notebook's outputs could feed into a broader ESG accounting system that monetizes these reductions.

Limitations & Future Work: This model relies solely on historical FAOSTAT data, which may be aggregated and lagging. It ignores other factors like technology adoption or policy changes. The Random Forest may capture patterns but not causal drivers. Future enhancements include: incorporating time series (to forecast trends), adding geographic granularity, or integrating socioeconomic features. One could also explore hybrid models or SHAP analysis (beyond simple importances) for deeper insight. The current analysis is a snapshot (separate from the dashboard), so integrating it with live data sources or linking its recommendations back into decision tools would make it more actionable.

LSTM CO_2 Emissions Forecast Dashboard

Purpose & ESG Relevance: This tool forecasts country-level CO_2 emissions into the future and quantifies the "carbon credit" opportunity of reducing those emissions. As one report notes, it's a decision-support innovation showing stakeholders *if current reduction trends continue, what future emissions will be*. By visualizing trajectories, it raises climate awareness and encourages commitments to lower-carbon pathways, directly addressing SDG 13 (Climate Action). The dashboard also explicitly frames its output in terms of potential **carbon credits** (tonnes CO_2 saved), linking predictions to offset planning and emission targets.

Data Sources & Structure: The app uses a cleaned dataset (`dataset.csv`) with columns `Entity` (country name), `Year`, and `Annual CO₂ emissions`. The code snippet shows it drops missing data and scales the emissions feature with a MinMax scaler. Each country's data is filtered on-the-fly based on user input. Only the emissions time series (one feature) is used to train the LSTM.

Technical Architecture: The LSTM model is implemented in TensorFlow/Keras (within `combined_app.py`). It uses a sliding window of 5 years of emissions as input to predict the next year. The network architecture is:

```
model = Sequential([
    LSTM(64, activation='relu', input_shape=(sequence_length, 1), 
return_sequences=True),
    Dropout(0.2),
    LSTM(32, activation='relu'),
    Dropout(0.2),
    Dense(16, activation='relu'),
    Dense(1)
])
model.compile(optimizer=Adam(0.001), loss='huber')
```

This two-layer LSTM captures temporal patterns, with dropout to prevent overfit. The trained model is used sequentially: starting from the baseline year, it predicts one year ahead, then feeds that prediction back as input for the next step. The code loop for generating yearly forecasts is:

```
for year in range(request.baseline_year, request.target_year + 1):
    lstm_pred_scaled = model.predict(np.array([[[current_input]]]))[0][0]
    lstm_pred = scaler.inverse_transform([[lstm_pred_scaled]])[0][0]
    lstm_predictions.append(lstm_pred)
    current_input = lstm_pred_scaled
```

Alongside LSTM, the app also computes a **baseline reduction curve** via a heuristic (0.9^(Δyears) multiplier) for comparison. The Streamlit front-end (`run_streamlit`) handles UI: users select a country, baseline year and target year. Historical data is plotted with Plotly, and upon request the prediction is shown. Key outputs (baseline vs. predicted emission, and implied carbon credits) are displayed in styled metric cards (as seen in code [13]).

Model Choice & Justification: LSTM was chosen for its strength in time-series forecasting with long-term dependencies. The literature confirms LSTM often outperforms linear models for CO_2 data. Here, LSTM can capture non-linear trends (e.g. abrupt drops due to events). The relatively small sequence length (5) balances reactivity with noise smoothing. More traditional approaches (ARIMA) could be considered, but the LSTM architecture is sufficiently lightweight for this MVP.

Training & Evaluation: The model is trained on all available historical emissions (up to 30 years per country). No separate test set is mentioned, but one could interpret the user's baseline/target selection as implicit validation. (In practice, one would hold out recent years to compute MAE/RMSE. According to project documentation, the model's R^2 on validation data is ~0.924 with MAE ~1429 tonnes.) The loss function is Huber (robust to outliers). No hyperparameter tuning is detailed, but default layer sizes and learning rate were used. The app logs cross-validation scores for the model at training time (via `cross_val_score` on the training set) [4] .

Outputs, Insights & Visualizations: On "Generate Predictions," the dashboard shows:
- Historical emission line chart (green) vs. predicted point (red diamond) for the target year (see example code [14]).
- A dashed trend line connecting the baseline and predicted points to illustrate change.
- Three metric cards: **Baseline Emission**, **Predicted Emission**, and **Carbon Credits** (tonnes saved). For example, if baseline was 10M and predicted 9M, it shows 1M tonnes saved.
- A data table of recent historical values.

These outputs provide a clear before-vs-after picture. The dashboard's design (colors, layout) accentuates reductions (green for baseline, red for forecast). The carbon credit metric directly quantifies the impact of reduction (e.g. "Tonnes CO_2 saved").

Carbon Credit Integration: Carbon credits are computed as `(baseline_emission - predicted_emission) / 1e6` (i.e. in million tonnes). This simple approach treats the LSTM's forecast as the scenario with assumed reductions, so the difference is "emissions avoided." In practice, this translates directly to credits under the assumption of 1 credit = 1 tonne CO_2. This connection encourages users to view the forecast in terms of offset opportunities. (The app also hints at credits in its narrative, tying to sustainability goals.)

Limitations & Future Work: This model relies solely on one feature (past CO_2). It ignores exogenous factors (policy changes, economic shocks), so it risks over-simplifying. The data per country is limited in length, so predictions beyond a few years are uncertain. The baseline curve ($0.9^{\Delta years}$) is a rough heuristic without empirical backing. The app also assumes the user has the cleaned `dataset.csv` locally and doesn't handle missing entries. Future improvements could include: adding more features (GDP, population); evaluating with true holdout sets; providing uncertainty bands; and integrating real carbon credit pricing data for more realistic offset scenarios.

CLI Carbon Footprint MVP

Purpose & ESG Relevance: This lightweight CLI tool quickly estimates the carbon footprint of a single shipment, facilitating on-the-spot sustainability checks. It supports ESG by increasing transparency in logistics; for example, a procurement manager can run a command to see recommended vehicles or packaging changes to reduce CO_2. The output breakdown (vehicle choice, material, total CO_2e) helps operational teams minimize emissions per shipment, thereby reducing Scope 3 logistics emissions.

Data Sources & Structure: The tool does not use a dataset. Instead, it relies on an **EmissionCalculator** (in `core.py`) which likely implements emission factors and formulas (not shown) for vehicle modes and materials. Users input origin/destination (lat,lon), weight (kg), and packaging material. The script computes distance and weight-based emissions.

Technical Architecture: The CLI is built with **Click** and **Rich** (for tables). Example code flow:

```
@click.command()
@click.option('--origin', required=True, help='Origin coordinates (lat,lon)')
@click.option('--destination', required=True, help='Destination coordinates
(lat,lon)')
@click.option('--weight', required=True, type=float, help='Shipment weight in
kg')
@click.option('--material', default='Cardboard', help='Packaging material')
def calculate(origin, destination, weight, material):
    # Parse coordinates
    origin_coords = tuple(map(float, origin.split(',')))
    dest_coords = tuple(map(float, destination.split(',')))
    calculator = EmissionCalculator()
    result = calculator.calculate_emissions(origin_coords, dest_coords, weight,
material)
    ...
```

The `EmissionCalculator` class (in a separate module) performs the core logic: computing CO_2e based on distance, weight, and material. The CLI then prints a Rich table with rows for **Recommended Vehicle**, **Packaging Material**, **Total CO_2e**, and a breakdown by category (e.g. "transportation emissions", "material emissions").

Model Choice & Justification: There is no learning model here – it's a rule-based calculator using fixed emissions factors. This approach ensures instant results with no training. It is justified as an "MVP" to demonstrate core functionality.

Outputs & Visualizations: Output is purely textual (terminal table). An example table rows:
- "Recommended Vehicle: Truck"
- "Packaging Material: Cardboard"
- "Total CO_2e (kg): 12.34"

- A section listing each emission category (e.g. *transportation: 10.00 kg*, *packaging: 2.34 kg*).
This immediate breakdown helps users see where most emissions come from.

Carbon Credit Integration: Not explicitly included in the CLI. One could imagine a future flag to suggest an offset or ask "If we switched to electric truck, how many tonnes saved" – but the current MVP does not compute credits.

Limitations & Future Work: The CLI requires precise coordinates and weight; it lacks data validation (except catching format errors). It only handles a single shipment at a time and assumes static factors. There is no learning or refinement from actual data, so accuracy depends on the built-in emissions factors (which may be generic). Future enhancements could include: adding CLI options for mode (plane vs ship, etc), querying a distance API rather than raw lat/lon, and outputting carbon credit suggestions (e.g. "Offset this shipment with X credits"). Also, packaging density or return rates could be factored in to fully align with ESG goals.

CarbonFootprintModel Module & Other Utilities

Purpose & ESG Relevance: The `CarbonFootprintModel` module (`model.py`) provides an end-to-end framework to train a regression model on shipping data and recommend reduction strategies. It supports supply-chain planning: by predicting shipment emissions given features (weight, distance, mode, customer and warehouse attributes), it helps identify the best decarbonization levers (e.g. lighter packaging, sea shipping). This directly aids Environmental and Governance goals by optimizing logistics for lower emissions and generating data for sustainability reporting.

Data Sources & Structure: The module expects preprocessed data from `data_preparation.main()`. This function (in a separate script) returns `X_train, X_test, y_train, y_test` along with scalers/encoders. Typical features include shipment details (weight in grams, estimated distance, cross-border flag), customer/warehouse metadata (CO_2 per capita, energy per GDP, etc.), and packaging info. The code then drops any direct emission labels from `X`. An engineered feature `weight_distance = Weight_in_gms * estimated_distance_km / 1000` is created (converting to kg·km) [2], capturing the combined effect of weight and distance.

Technical Architecture (Processing Flow & Code Highlights): The `CarbonFootprintModel` class is structured with methods for loading data, training, evaluation, interpretation, and recommendation:
- **Data Loading:** `load_data()` calls the external `data_preparation` module to populate `self.prepared_data`.
- **Training:** In `train_model(model_type='random_forest')`, the script selects an algorithm (default Random Forest, or Gradient Boosting / Ridge). For random forest, it initializes:
`python`
```
  self.model = RandomForestRegressor(
      n_estimators=100, max_depth=15,
      min_samples_split=5, min_samples_leaf=2,
      random_state=42, n_jobs=-1
  )
```
[5]
Cross-validation scores (R^2) are printed for 5 folds. The model is then fitted on `(X_train, y_train)`. Feature importances are stored for later analysis.

- **Evaluation:** `evaluate_model()` predicts on `X_test` and computes RMSE, MAE, and R^2. It also generates and saves diagnostic plots: *Actual vs. Predicted scatter*, *residual plot*, and *error histogram* (code excerpt: [39] shows these plotting calls).
- **Feature Importance Analysis:** `analyze_feature_importance()` plots the top-15 feature importances in a horizontal bar chart and returns the importance series.
- **SHAP Analysis:** `generate_shap_analysis()` uses SHAP values to explain the model. It produces a SHAP summary bar plot and a force plot for a sample prediction, as well as dependence plots for the top 3 features [6]. These visualizations aid in understanding complex model behavior.
- **Recommendations:** The method `generate_carbon_reduction_recommendations(shipping_data)` takes a single shipment's features and simulates different strategies (stored in code):
- *change_to_sea_shipping* – sets `Shipment_mode` from Flight to Ship.
- *local_warehousing* – halves the `estimated_distance_km` (simulating closer warehouse).
- *optimize_packaging* – reduces `Weight_in_gms` to 80%.
For each strategy, it recomputes `weight_distance` and predicts new emissions. It returns a sorted list of recommendation dicts, each with the strategy name, description, baseline vs. new emissions, and % reduction [7] [8]. (The `_get_strategy_description` helper maps names to human-readable advice.)

Model Choice & Justification: The Random Forest regressor was selected for robust performance and interpretability. Its tree structure captures feature interactions without heavy tuning. The code also allows switching to Gradient Boosting or Ridge if needed (though defaults and parameters are hardcoded). The use of a pipeline with scaling and encoding (not shown in snippet but implied by loading `encoder`) ensures consistent preprocessing between train and inference.

Training & Evaluation: 5-fold CV is used during training to estimate generalization (mean R^2 and std dev printed) [4]. On test data, typical evaluation metrics are recorded. For example, the code logs RMSE, MAE, and R^2 [9]. In practice, high R^2 would indicate the model captures most variance in shipping emissions. No hyperparameter tuning (GridSearch) is explicitly invoked, though the code imports `GridSearchCV` (perhaps for future use).

Outputs, Insights & Visualizations: Beyond the textual metrics, the module saves several figures:
- `feature_importance.png` (top 15 features) [10], e.g. illustrating which factors (like weight_distance or mode) matter most.
- `model_evaluation.png` (actual vs predicted).
- `residual_plot.png` and `error_distribution.png` (residual analysis) [11].
- `shap_summary.png` (SHAP bar chart) and `shap_force_plot.png` for a sample [6].
These are crucial for interpreting the model and validating its trustworthiness. A feature like `weight_distance` is likely among the top, confirming physical intuition.

Carbon Credit Integration: The `generate_carbon_reduction_recommendations` method explicitly quantifies "avoided emissions" for each strategy (baseline minus new prediction). These values can be directly converted to carbon credits. For example, if switching to sea shipping reduces emissions by 500 kg CO_2, that's 0.5 tonnes (0.5 credits) per shipment. Thus, the module not only predicts emissions but also provides actionable carbon-saving estimates.

Limitations & Future Work: The model requires clean, complete training data. It currently ignores temporal trends or seasonality in shipping (it's a static model). Strategies considered are limited to a few hand-coded ones. Future work could include: dynamic training (updating model over time), more sophisticated strategies (combining strategies, batch optimization), and integration with real-world carbon pricing. Adding uncertainty estimates (e.g. prediction intervals) would also align with risk-aware ESG planning.

ESG Impact Synthesis

Collectively, these tools drive measurable ESG benefits. **Environmental:** They increase carbon transparency – from micro-scale (per-order or per-shipment emissions) to macro forecasts (national emissions). This enables tracking key sustainability metrics (e.g. CO_2 per distance, per capita) and spotlighting reduction opportunities, which is core to climate action goals. **Social:** By making data visible (dashboards, reports, CLI outputs), they engage stakeholders across the organization – operations teams see direct impacts of choices (shipping mode, packaging), and leadership gains trust through data-driven reporting. **Governance:** Structuring data and models supports compliance and reporting (e.g. generating figures and narratives for ESG disclosures). The emphasis on scenario analysis and carbon credits embeds **accountability** into decision-making – companies can set target reductions and use these tools to plan and justify them. In sum, by blending data science with clear visualization, the suite empowers companies to integrate emissions management into their ESG strategy.

Future Prospects and Roadmap

Short-Term: Standardize data flows (e.g. automate FAOSTAT updates, Olist ingestion). Enhance model robustness (cross-validate forecasting models on more countries). Improve UI/UX by adding explanatory tooltips or multi-country comparison. Introduce more **carbon-credit functionality**: for example, link e-commerce and LSTM forecasts to a live credit market API to price saved CO_2. Incorporate missing data handling and flags (e.g. warn if dataset is incomplete).

Modeling Improvements: Explore hybrid time-series models (e.g. combining ARIMA or exogenous regressors with LSTM). For the agricultural model, consider a longitudinal (time-series) approach if multi-year data is available. In the shipping model, experiment with ensemble methods or gradient boosting to compare performance. Calibrate all models with hold-out validation sets to quantify uncertainty.

Integration: Work towards a unified dashboard that can toggle between tools or compare results (e.g. overlay farmland emission projections with national LSTM forecasts). Containerize the applications (Docker) and host on cloud for scalability. Expose APIs from the CarbonFootprintModel so that the CLI and dashboards can call it in real-time.

Future Directions: Add scenario analysis features: e.g. let users manually adjust model inputs (fuel price, policy changes) to see emission trajectories. Engage users with alerting (e.g. notify when forecast exceeds thresholds). Finally, expand the toolkit: include other sectors (industrial, transport), or add sustainability KPIs beyond CO_2 (water use, energy consumption). Continual feedback from domain experts will guide new features, ensuring these tools remain aligned with evolving ESG standards and climate science.

References

- CO_2 Emissions Prediction Dashboard (project report)
- *combined_app.py* (LSTM-based forecasting code)
- *cli.py* (Carbon Footprint CLI code)
- *model.py* (CarbonFootprintModel code) [12] [13]
- *Carbon_Emissions_in_Farming.ipynb* (FAOSTAT analysis)