WPD INNOVATION
Transforming the electricity network

Spatially Enabled Asset Management (SEAM)

D05 Model Design Document

WESTERN POWER DISTRIBUTION
Serving the Midlands, South West and Wales

Version Control

Issue	Date
1.0	29/06/2021

Publication Control

Name	Role
SEAM Project Team	Author
Jenny Woodruff	Reviewer
Jenny Woodruff	Approver

Contact Details

Email

wpdinnovation@westernpower.co.uk

Postal

Innovation Team
Western Power Distribution
Pegasus Business Park
Herald Way
Castle Donnington
Derbyshire DE74 2TU

Disclaimer

Neither WPD, nor any person acting on its behalf, makes any warranty, express or implied, with respect to the use of any information, method or process disclosed in this document or that such use may not infringe the rights of any third party or assumes any liabilities with respect to the use of, or for damage resulting in any way from the use of, any information, apparatus, method or process disclosed in the document.

Contents

1. Context and purpose of this document

The purpose of the Model Design document is to provide a detailed representation of the Spatially Enabled Asset Management (SEAM) model designs. This builds upon the information provided in D01 – The SEAM Specification Document and D02, the Model Definition Document produced at an early stage of the project as shown in Figure 1: SEAM Project Deliverables below. These documents captured the common errors that could be found within WPD's data and sought to focus on those that were "harder to fix", generating groups of use cases. From these groups of use cases it became apparent that two different types of model would be required to cover the range of potential data errors. Model 1 involves the creation of a traditional graph model and testing the ability of the network created to satisfy the load requirements. Model 2 uses a spatial graph model to infer incorrect or missing customer attributes without requiring a full and correct connectivity model as a pre-requisite.

This document follows completion of the proof-of-concept (PoC) model and User Interface (UI) development activities and describes the models and interface as they were following the minor updates that occurred during the User Acceptance Testing phase.

The design document covers the following topics:

- Description and scope of the models
- High-level model designs
- Model logic and data flows (for each step the inputs, outputs, assumptions, processes, and techniques / calculations)
- Code structure
- User Interface design

The design refers to the final PoC models and UI delivered as part of D04 AI Model and User Interface.

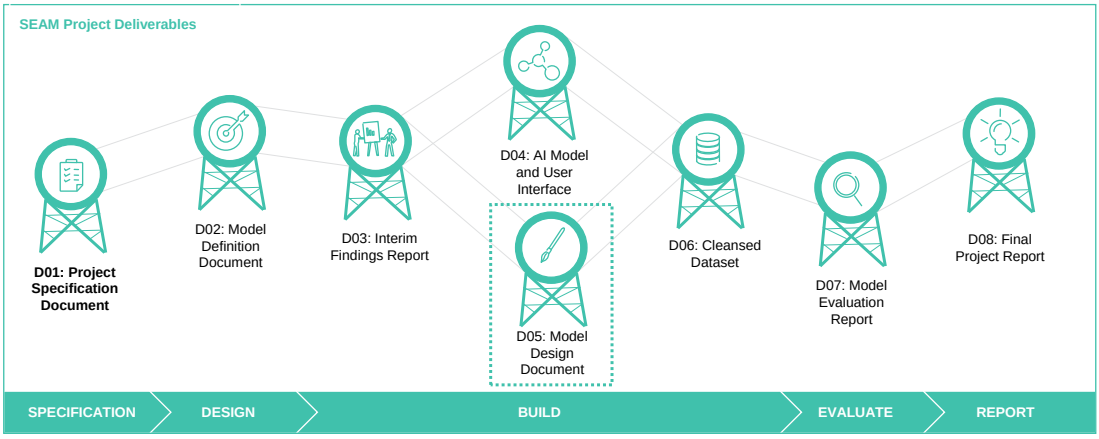

Figure 1: SEAM Project Deliverables

2. Model 1

2.1. Description of the model

The purpose of the connectivity model is to use the connectivity of assets and customers to perform a simplistic transportation model to verify that the existing assets (and their attributes, i.e. capacity in terms of kW) is configured such than it can supply demand at peak times in the winter for customers connected to the circuit.

Creating a graph for each unique circuit_id (As recorded in Electric Office), wires, cables, customers, substations are connected to the model, where line assets (wires and cables) are edges and point assets / customers are nodes. Demand is calculated at the customer nodes (either using estimated annual consumption and Elexon profile class or half-hourly meter readings if available) and a network flow problem is configured to determine an optimal strategy for routing power through the network. A Max Flow algorithm is used as an alternative to running power flow analysis as this achieves similar results but without the complexity, longer processing times and licensing requirements associated with power flow analysis tools. Where the network is unable to supply the peak demand this could be the result of an error in the connectivity model resulting in the network appearing to extend further than it's real bounds e.g. in the case of a missing open point. Alternatively an asset with incorrect data about the asset type, size or material would result in an incorrect capacity value which could also result in an apparent overload if the capacity were underestimated. Incorrect customer load data might also result in apparent overload and lastly there is the possibility that the network is correctly represented but genuinely overloaded, which is also worthy of investigation and follow up activity by network planners.

The electrical connectivity of the linear assets in the circuits (cables and wires) and customers is not provided at LV circuit level, as it is at MV and HV level and steps are taken to approximate this connectivity:

- Connectivity is implied in this modelling when nodes that are connected to 1 edge are at exact locations
- For disconnected 'isolated graphs', when nodes that are connected to 1 edge are the most closely clustered compared to other nodes that are connected to 1 edge
- Customers are connected to the closest node that are connected to 1 edge
- Substation is located at the closest node to the centroid of the Electric Office geometry

2.2. Scope of the model

The intention of this model is to use the implied connectivity of assets and the assumption that customers are supplied real power from substations through the connected assets such that any discrepancies in the data (either connectivity, specification of assets, customer connectivity) can be identified through:

- Power not being transported to the customer
- Headroom (i.e. capacity less flow) on wires and cables is not sufficient according to design / operational safety factors

As the data relating to the connectivity of the assets are not exact with explicit relationships between the assets, some additional tests and exception reports will be generated which highlight:

- Nodes where connections / geometries should exist to fully connect isolated disconnected circuits
- Nodes where customers and the closest nodes that are connected to 1 edge
- Nodes where substation can be located on the graph of wire and cable edges

Due to the merging of disparate datasets from different systems, exception reports where no matches are found in the matching process are also generated

- For matching Customers with MPANs included in the P222 quarterly data exchange which provides EAC data to Electric Office circuit_id A tool to read and process the P222 csv files from each data aggregator to create a compiled database per licence area was developed as part of the Falcon project, so these datasources refer

to Falcon because of the use of that tool rather than reflecting the trial area of Milton Keynes and surrounding suburbs that was used for the Falcon project.

- For matching Half Hourly customers to Electric Office circuit_id

2.3. Model logic and data flow

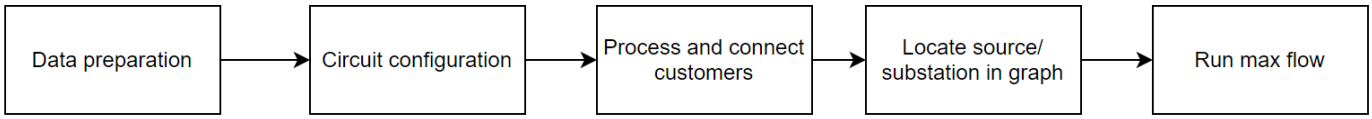

Figure 2: High level view of the processes required for running the model

2.3.1. Data preparation

The data preparation stage prepares the datasets and is comprised of a number of merges between data sets, filters, and also calculation of the capacity, with the first stage of capacity backfilling applied, i.e. estimating the capacity where there is not sufficient asset data to confirm the capacity. Datasets which have been processed are stored in pandas dataframes, serialised stored as pickle files in the directory '../temp' and de-serialised and processed.

Function	Description	Techniques / calculations	Data in
ingest_merge_customer_uprn	merges three datasets to create a dataset which links locational data to customer identifier (MPAN, UPRN) to a substation and feeder	CROWN customer MPAN dataset is left joined to UPRN database on 'UPRN' This is then left joined to CROWN customer dataset on 'MPAN CORE'	**UPRN database:** osopenuprn_202101.csv *UPRN to geospatial reference (i.e. X-coordinate, Y-coordinate)* **CROWN customer dataset:** Q_SEAMconnectivitydata.xlsx **CROWN customer MPAN dataset:** Q_SEAMMPANsrequiringmatching UPRNdata_21012021.xlsx
query_access_demand	finds the unique set of MPANs in the CROWN customer data set and searches the SQLite database for matching MPANs to return rows which match	Create a connection to the SQLite databases in the data directory and returns dataframe of matching MPAN rows	**Falcon estimated annual consumption:** FalconEac - SWEB Nov 2020.SQLite **Half hourly customer consumption:** HH SWEB 2017 Q4.SQLite HH SWEB 2018 Q1.SQLite HH SWEB 2018 Q2.SQLite HH SWEB 2018 Q3.SQLite

merge_customer_demand_hh merge_customer_demand_eac	Merges datasets to create a dataset which links customer to demand profile (half hourly) or estimated annual consumption	customer-mpan-uprn-gdf is inner joined to combined EAC tables on MPAN CORE customer-mpan-uprn-gdf is inner joined to combined Half hourly tables on MPAN CORE	
process_eo	Filters electric office datasets by substation based on assets which are within the bounding area specified as an input	Reads in all layers in the fname specified, clips each table by a polygon made by user define xmin, xmax, ymin, ymax. The unique set of substations within the tables cable, wire, substation_pm and substation_gm are then used to filter these tables in order to preserve complete circuits where possible, i.e. if a circuit is partially over the box region	**Electric Office GIS dataset:** SEAM 2021-02-26.sqlite
process_directives	Process digitised csv files version of WPD policy documents which contain the current ratings for wires and cables of specified cross-sectional area, material / type and number of cores / wires.	Concatenate all directive files relating to wire and cable; group by on number of cores, size, material and phase and calculate max / min for each category. This is required as there are some duplicated values (due to other factors recorded in the policy document but not reflected in the data, such as types of casing on the cable, e.g. XLPE, oil impregnated paper etc.)	**Tabulated WPD Directives:** cable ratings - SD8B_4_part1.csv cable ratings - SD8B_4_part2.csv cable ratings - SD8B_3_part3.csv extract_from_wire_ratings_SD8A_3.csv
find_eo_spec_elements	Size, material and number of cores data is embedded within the 'specification_description' column within cables and 'specification_description_1' within wires and the individual elements are extracted in this stage	Material is extracted by finding a unique set of materials in the directive files and finding longest matching string within the column Size and wires / conductor is extracted by using regex match pattern If size is less than 1, it is assumed that this is a in^2 metric and converted to mm^2 by multiplying by 645.16	
directive_conformity_size_material	Checks for material and size extracted from the specification_description column against the directive files.	Checks for material and size conformity in directives. Adjustments are made if the size is within 0.05 of size which exists in the directive. This is output to the column 'size_adjusted'	
directive_conformity_size_material_wire_num	Checks for material, size and number of cores extracted from the specification_description column against the directive files.	Checks for material, size and number of cores conformity in directives. If material, size are conforming but number of cores is not and adjustment is made to output column 'w_adjusted' / 'c_adjusted' to the number in directives with least absolute difference.	
Merge_directive	Merge 'normalised' specification_directive with directives		
find_cable_wire_capacity	Calculate wire / cable capacity for each asset in Electric Office	Calculate maximum power capacity in kW using the nominal voltage and rated current found in directives and matched using the specification_description column Capacity = (nominal_voltage_pp* amps) / 1000	

merge_cust_eac_eo merge_cust_hh_eo	Merge customer data with electric office data	Synthetic circuit id is created from CROWN columns 'DIST SUBSTATION' and 'LV FEEDER' as customers are not assigned circuit_ids Substation and LV feeder is extracted from circuit_id by first finding a full LV feeder code, or if this is not found the last two digits are used to match against the LV feeder set. The two datasets are then joined on the synthetic circuit id	
backfill_missing_capacity	Backfill missing capacity for wires and cables according to network_type, usage, bool_wpd_running_3_phase and type: at circuit_id level if existent or at area level if not.	Merge the Electric Office wire / cable datasets with: Circuit_id level aggregation: Aggregate electric office wire / cable datasets using network_type, network_type, usage, bool_wpd_running_3_phase, type, circuit_id and aggregate using a function of the user's choice (min, max, mean) Area level aggregation: Aggregate electric office wire / cable datasets using network_type, network_type, usage, bool_wpd_running_3_phase, type and aggregate using a function of the user's choice (min, max, mean) Merge columns, taking circuit_id level aggregation where possible to produce columns 'capacity_backfill' and 'capacity_backfill_type' NB: 'unmetered service' line asset types are treated as 'service' and backfilled accordingly	

Table 1: Data preparation (model 1)

2.3.1. Circuit configuration

In this section, the circuits are configured from the GIS data, with connections added to connect some microdisconnects which appear in this data. The Electric Office data is processed using the python library NetworkX, for studying graphs and networks. Once configured, the individual graph objects are serialised and stored in directories for further analysis.

Function	Description	Techniques / calculations
Build_circuits	This is the main function which loops through individual circuit_id circuits and drives the procedures for creating circuit graphs	For each circuit with circuit_id: Linestrings which include curvature are planarized to preserve where curvature is important for connectivity Planarized linestrings are then made into a network graph object, where lines are planarized cables and nodes are connections between these lines The nodes with degree two (where all attributes, apart from the geometry, are the same) are removed and replaced with a line connecting to its other node. (this is to clean the graph to

		preserve only where edge attributes change and / or there are junctions connecting many lines) If the number of isolated graphs is greater than one, Kmean clustering is used to cluster nodes with degree 1 into (number of nodes with degree 1 - number of isolated connected graphs + 1) clusters. This method assumes that any disconnected nodes are closer than other nodes which should remain disconnected. This method does not connect disconnects where the disconnect occurs at vertex to line, line to line, or where the disconnect is caused by exceptions where the distance between disconnected nodes is greater than the next set of disconnected nodes which are actually disconnected. If this connection method creates a single output connected graph, the set of nodes with degree 1 and the centre of the cluster is saved. The new edges are then from the cluster centre to each of the nodes with degree 1. This ensures that there is no ambiguity as to which nodes are connected for connections between 2+ nodes. New edges are labelled with {"gen_type": "sim_disconnects"} Separate graphs are saved for each circuit_id in the \temp\circuit_graph directory

Table 2: Circuit configuration (model 1)

2.3.2. Connecting customers

In this section, the graphs configured in the previous section are used to connect customers by adding edges from the customer to the nearest node with degree 1 in the graph. Within this section, the settlement date and period are found for the peak demand for each circuit to calculate the demand per customer at this time.

Function	Description	Techniques / calculations
find_peak_settlement_period	This function aggregates energy demand from half-hourly and estimated annual consumption customers by circuit_id and finds the max half-hour time period. The individual customer peak is then calculated per customer using the peak date and settlement period.	Standardise date format columns from P222 file provided EAC data as processed by the Falcon tool and half hourly datasets and filter “Settlement Date” column to 2019 winter months. *Find peak demand per circuit_id* Merge estimated annual consumption dataset with Elexon coefficient dataset[1] using a left join on the fields “ProfileClassId” and “Profile Class ID”. Multiplying the estimated annual consumption figure by the Elexon coefficients (from data file *Default_Period_Profile_Class_Coefficient_308.csv*) accounts for the mean demand element. If only the mean demand element (i.e. Elexon coefficient for the settlement period multiplied by the estimated annual consumption number) is accounted for then there is 50% chance that the demand would exceed this value in a given situation. This reflects the fact that the original use for Elexon profiles is to calculate values for settlement. These calculations typically take place at Grid Supply Points where there are many thousands of customers included and the profiles reflect the full impact of diversification between customers. The

[1] https://www.elexon.co.uk/operations-settlement/profiling/

		formulation within this model which acts to correct for the lower level of diversity between customers for a single LV feeder is: Design demand = Mean demand + 1.28 x standard deviation This design demand corresponds to a 90% probability of meeting the demand within the design voltage regulation – an acceptable level of risk (ACE 105). The standard deviation element is integrated in this model as an input variable as a percentage of the mean demand. i.e. if the user chooses 0.1 this is 10% of the mean demand and this is used as the standard deviation. This demand is given in kWh for the half hour; the corresponding power in kW is calculated by Design demand / 0.5 i.e. Design demand * 2. Thus if the customer demand in a half hour period is 2kWh then this is the equivalent of an average power of 4kW being drawn for the half hour period. The half hourly dataset and estimated annual are combined and a group by is applied on the columns “Settlement Date” and “Settlement Period ID” with a sum aggregation on the “hour_hour_demand” column for each circuit_id. The output dataframe is then: circuit_id, circuit_peak_Settlement Date, circuit_peak_Settlement Period ID, circuit_peak_half_hour_demand. *Find peak demand per customer given the peak settlement date and id* This output is then joined back onto the customer consumption and the peak demand per customer is calculated and saved.
Connect_customers	This function connects customers to their corresponding circuit_id graph network	A k-d tree (space-partitioning data structure for organizing points in a k-dimensional space) of all nodes in the circuit graph with degree = 1 (i.e. nodes with only one edge connected to) is used as the search space against customer’s x and y coordinates for the nearest node. The new edge is then added with the label {"gen_type": "sim_cust_circuit"} and the new graph is saved.

Table 3: Connecting customers (model 1)

2.3.3. Locate substation

In this section, the substation is located on the graph, given a threshold (default set to 10m).

Function	Description	Techniques / calculations
locate_sub	This function connects locates the node closest to the substation in the circuit, where the distance is below a threshold (set to 10m)	For each circuit_id, all substations queried against a k-d tree of all nodes in the circuit. If the distance is less than threshold (10m), the substation is added to the circuit via attributes in the node.

Table 4: Locate substation (model 1)

2.3.4. Run max flow

Within this section, a simple graph-structure based backfilling of capacity is applied for specifically synthesized edges which connect the customers to the network as well as edges which connect disconnects. Pre-processing, which is required to simplify the circulation with demand problem to a maximum flow problem, of the graphs is also conducted at this stage. The maximum flow analysis is then conducted and parsing / post-processing of results is also carried out within this script.

Function	Description	Techniques / calculations
run_max_flow_circuits	This function runs the procedures which: - backfills capacity in any generated edges (i.e. gentype = sim_cust_circuit and gentype = sim_disconnects), - allocates supply amount from the substation location, - adds theoretical edges to the circuit graphs (in order to run maximum flow on a circulation with demands problem), - runs max flow and parses the output.	Capacity backfilling for generated edges is done on a neighbourhood aggregation basis as these edges are added as extensions / connections of existing cables. The user input max, mean or min sets how the aggregation is applied. Power supply, which is -ve of the total demand in the circuit is allocated at substation. Where there are more than one substation per circuit, the supply is split evenly across all substations. In order to reduce the circulation with demands problem (i.e. finding feasible flow that satisfies capacity constraints and demand constraints) to a maximum flow problem, which has very efficient and fast algorithms, additional source and sink nodes, with edges of capacity equal to demand / supply connecting to the substation(s) and customers, respectively.[2] See below for a diagram. The networkX's implementation of the maximum flow algorithm is used to solve the maximum flow problem. The results are then parsed and saved as a new directed graph; i.e. there is a feasible output flow attribute for each edge. Post processing is then applied to remove theoretical edges and convert the theoretical node edge attributes back to node attributes.
make_report	Converts the maximum flow output graph to tabular format and compile post analysis headroom reports.	The graphs as output from run_max_flow_circuits are then converted back into a GeoDataframe and the headroom is calculated on each cable / wire as well as the relevant nodes (i.e. customers and substation) where for edges: Head_room = capacity – flow Head_room_pc = (capacity – flow) / capacity For nodes: Demand_not_met = capacity – flow For the output max_flow_reporting report, the GeoDataframe is queried to provide:

[2] https://www.cs.umd.edu/class/fall2017/cmsc451-0101/Lects/lect17-flow-circ.pdf

		n_headroom_abs – number of wires / cables with headroom less than threshold as input by the user n_headroom_pc – number of wires / cables with percentage headroom less than percentage threshold as input by the user n_cust – number of customers with demand not met > 0.1

Table 5: Run max flow (model 1)

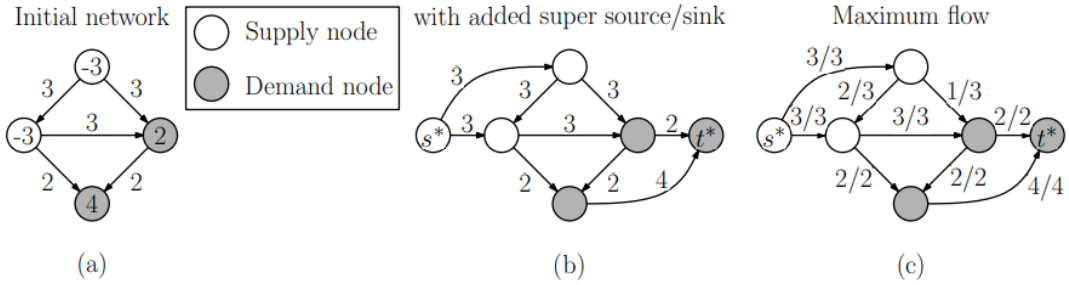

Figure 3: Reducing the circulation problem to network flow [1]

2.4. Code structure

The overall model is executed from main.py which runs a number of modules in sequence:

- file_check.py
- data_prep.py
- circuit_config.py
- customer_connectivity.py
- locate_substation.py
- run_max_flow.py

Below diagrams show the functions and modules and the procedure in which they are executed.

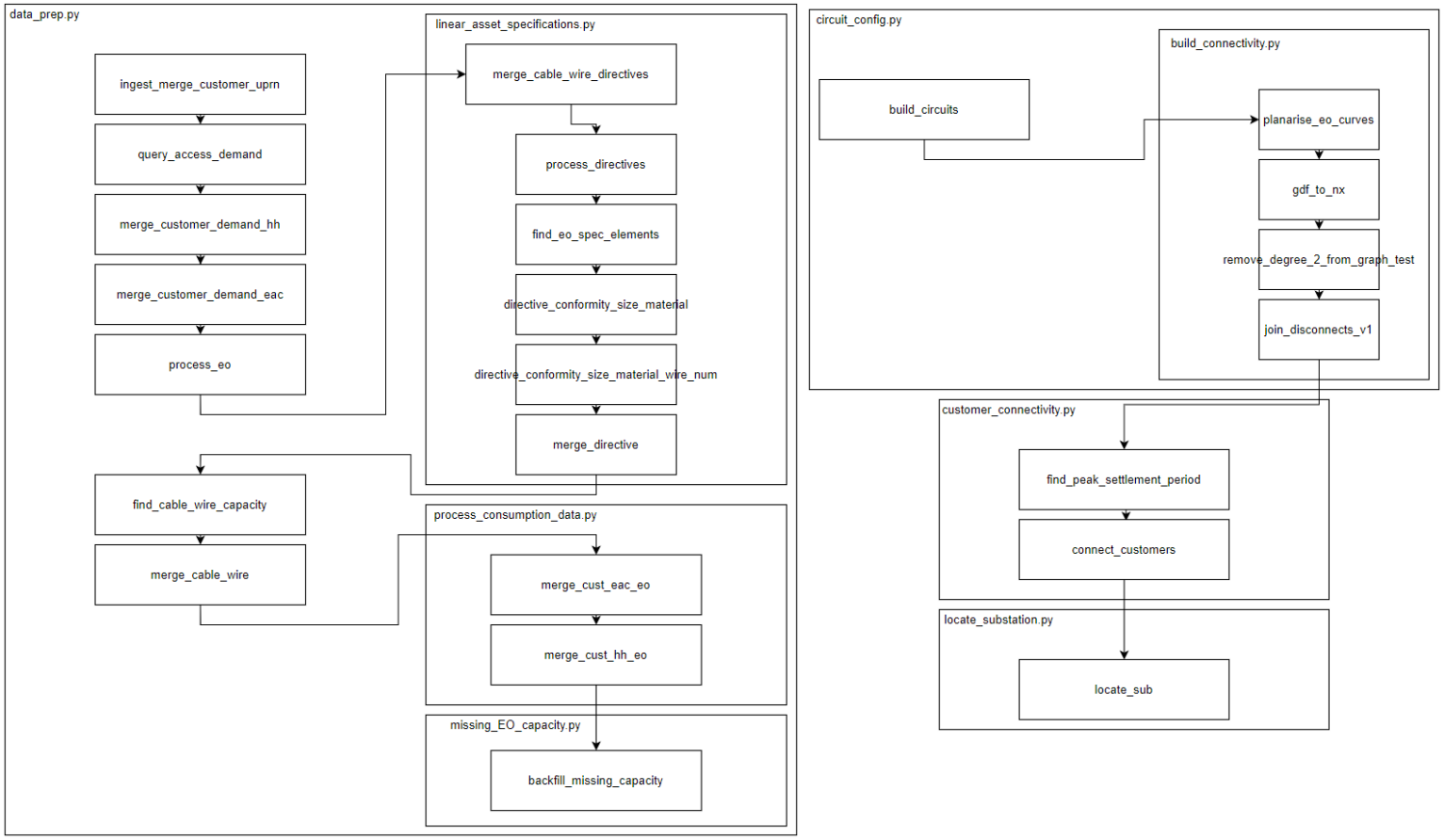

Figure 4: Left: Procedure in which functions and their respective modules which are initiated when running: the data preparation stage: wpd_seam.data_prep.main()

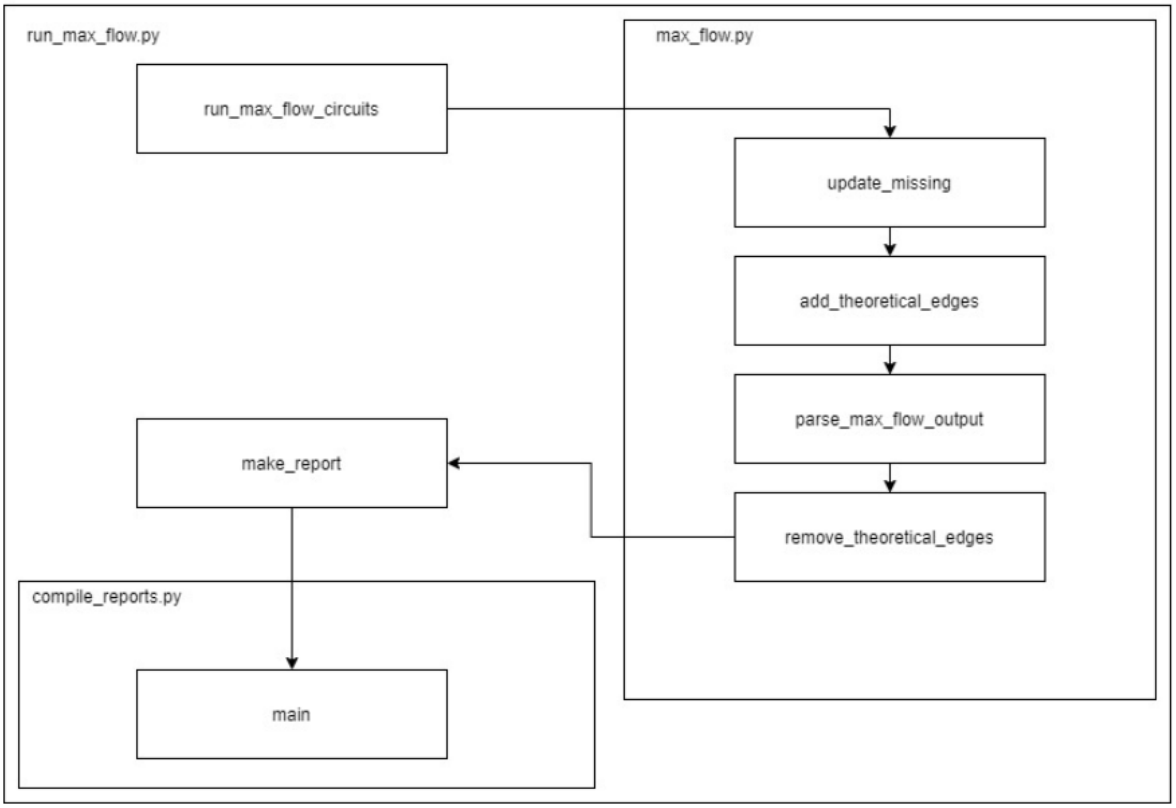

Figure 5: Procedure in which functions and their respective modules which are initiated when running: the run_max_flow stage: wpd_seam.run_max_flow.main()

3. Model 2

3.1. Description of the model

The main purpose of the spatial graph model (model 2) is to identify incorrect and missing attributes in the GIS data and to suggest correct values through the application of machine learning, which corresponds to use case group 2 from the project specification document[3]. This is achieved by exploiting the spatial relationships between the assets in addition to their intrinsic attributes, using a graph data structure and a graph neural network (GNN) based machine learning model. This allows it to learn spatial patterns of attributes in the electrical network and hence to identify attributes that are inconsistent and suggest correct values.

The model also provides a structure that can be extended with data from external sources such as CROWN, that can be used to detect inconsistencies across systems and suggest resolutions through the application of machine learning, which corresponds to use case group 3 from the project specification document[2].

For each prediction, the model also produces confidence scores that can be used to filter or rank the predictions. These are obtained from the raw outputs of the neural network. There are three trained sets of thresholds that the user can choose between to determine the required level of confidence required for a suggestion to be included in the detailed exceptions report. There is also an option to disable these thresholds and include all suggestions in the report.

The model is inductive, which means that a trained model can be applied to unseen data without retraining. Hence, the model can be trained on a subset of the network and then used to create predictions for other parts of the network. Furthermore, the prediction process is fast.

As an inductive, machine-learning-based model, there are three separate, but related, processes involved; each with its own purpose and outputs. These are as follows.

- **Training**: This process is used to obtain a new trained model that can be used to generate predictions. The model is trained against the data for a selected region taking the original data as ground truth and adding synthetic errors to form the input data. The GNN model is iteratively optimized to increase the number of output values from the model that match the true (original) values given the corrupted input data.
- **Evaluation**: This process is used to measure the ability of a pre-trained model to make correct predictions, given data with synthetic errors. As for the training process, it uses the original data for the selected region as ground truth and add synthetic errors to form the input data. It can either be applied in a transductive (i.e. same region as training, but different synthetic errors) or inductive (i.e. different region from training) manner.
- **Prediction**: This process is used to identify errors in the original data and obtain suggested corrections. It uses the pre-trained model to generate predictions using the original data for a selected region as the input data. This process outputs the list of suggested corrections that meet the necessary scoring thresholds. It can either be applied in a transductive (i.e. same region as training) or inductive (i.e. different region from training) manner.

The only differences between these processes are:

- Whether the model is loaded from a file (evaluation and prediction) or trained and saved (training)
- Whether synthetic errors are included in the input data (training and evaluation) or not (prediction)
- Whether the model evaluation output reports can be created (training and evaluation) or not (prediction)

3.2. Scope of the model

As a proof-of-concept, the scope of the model is to predict the correct values for following attributes for each asset using the original values plus the geometry of the point and line assets. These data are all obtained directly from the EO dataset.

[3] Insert reference to D01 Project Specification document

- network_type: Network type attribute
- nominal_voltage_pp: Operational voltage attribute
- spec_material: Material part of the specification description attribute
- spec_size: Size part of the specification description attribute

No external data is used except for:

- List of possible material values previously extracted from the WPD company directives.

There is one model that takes all of these as inputs and outputs predictions for all of them in parallel.

Specifically, the model only uses the following attributes from each layer in EO.

- cabinet: 'id', 'geometry'
- cable: 'id', 'geometry', 'network_type', 'nominal_voltage_pp', 'specification_description'
- conduit: none
- connector_point: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- connector_segment: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- energy_consumer: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- energy_source: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- isolating_eqpt: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- keypole: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- pole: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- power_transformer: 'id', 'geometry'
- protective_eqpt: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- service_connection: 'id', 'geometry'
- service_point: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- substation_gm: none
- substation_pm: 'id', 'geometry'
- tower: 'id', 'geometry', 'network_type', 'nominal_voltage_pp'
- wire: 'id', 'geometry', 'network_type', 'nominal_voltage_pp', 'specification_description_1'

3.3. High-level design

Objectives

As discussed above, the main objective for model 2 (spatial graph model) is to identify and correct missing and erroneous attributes values in the GIS data.

In particular, the model must be able to support the following objectives, even if they are not necessarily implemented for the PoC:

- Multiple asset types, including:
  - point, line and polygon geometries
  - electrical assets (e.g. wire, isolating equipment)
  - support assets (e.g. pole, substation)
- Include spatial relationships between assets
- Include electrical relationships between assets (where available)
- Include other relationships between assets (e.g. physical)
- Allow for missing assets, attributes and relationships
- External data (e.g. buildings with attributes)
- Incremental addition of input attributes, feature types and relationship types
- Able to predict for unseen data, but not necessarily optimized for it

Background

WPD's GIS data were migrated from the previous CAD-based system, which contained data that were accumulated over many years from multiple predecessor organisations according to different standards and procedures that were

applicable at the time, and is being updated and improved by the GIS team. Due to their origin, the GIS data have the following characteristics:

- Lots of missing assets.(most commonly service cables)
- Attributes derived from symbology used in CAD system and manual data entry. Hence, geometries are complete in 2D for all assets present in the data, but otherwise, there are lots of missing or incorrect attributes.
- Geometries of assets that are directly electrically connected may not meet exactly. For example, there may be gaps or overlaps.
- Geometries of assets that are not directly electrically connected may overlap or come very close together. For example:
    - HV wire passing over an underground LV connector point should not be part of the same circuit.
    - 2 cables and a connector point (CP) should be part of the same circuit but connected as cable to CP and CP to cable with no direct connection between the cables.
- Electrical relationships limited to circuit membership. Circuit membership is currently being inferred by GIS team using an algorithmic approach based on the geometries, and this activity is ongoing.

Selected Approach

There are two key components to this model: the spatial graph structure and the graph neural network. The spatial graph structure is a model for organising the GIS data into a graph structure based on the spatial relationships between the assets, and the graph neural network is the machine learning model that is used to predict the true properties of assets in the graph. In this PoC, the GNN is limited to predicting the true attribute values for selected attributes of the EO assets (i.e. node classification).

This is different from approaches that have been taken before, because:

- it does not require information about electrical connections,
- it supports all kinds of assets, attributes and relationships,
- it supports external geospatial data, and
- it can be constructed using only a subset of the assets, attributes and features.

Spatial graph structure

The spatial graph model contains a layer of point location nodes, with distance edges between them to create a spatial mesh, and edges between each asset or features and the location nodes that are part of it.

The model comprises:

- one node for each asset/feature
- one node for every unique point location (subject to some tolerance)
- edges from each assets/features to every location that is associated with it:
    - points: one location from Point geometry
    - lines: two or more locations that make up the LineString geometry
- reverse edges for locations to assets
- edges between “nearby” locations with “distance” attribute to create a complete mesh
- self-loop edges for all node types

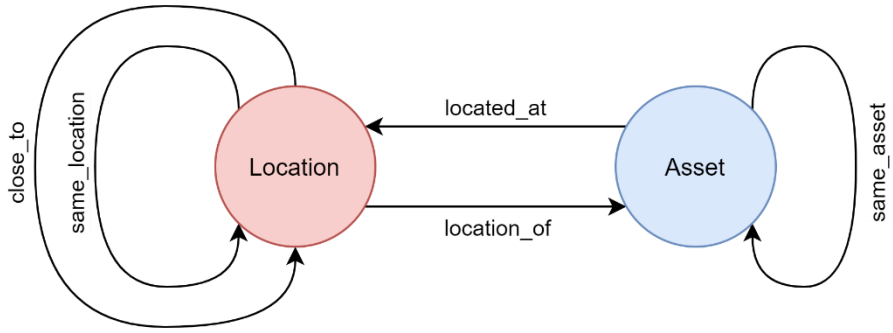

Figure 6: Spatial graph structure for SEAM PoC

This structure could be extended by adding, for example:

- edges from polygon assets to all locations that are within them
- edges between assets to denote logical relationship, such as:
  - pole-mounted assets to associated pole(s) and/or towers
  - substation to associated winding, isolating and protection equipment
- one node per circuit ID plus edges to all the assets that are part of that circuit
- edges between assets to denote direct electrical connection
- node(s) for each external feature, such as UPRN or annotation, with edge(s) to associated location node(s)

Note that this graph model is still fairly complex, even the minimal version, because it is a heterogenous graph (i.e. different node types and different edge types) with different node feature vectors for each node type, some edges with feature vectors, and both directed and undirected edges.

The main advantages of this model are as follows.

- Can be constructed using only the assets, attributes and relationships of interest.
- All assets are connected to all other assets via the location layer.
- Separate, complementary concepts of sharing location and electrical connection.
- Information about electrical connection is not mandatory, but can be included, even if it is incomplete.
- Allows other location-based data, such as UPRN and property classification, to be added.
- No edges with zero distance (or very short).
- The problem of selecting distance edges to add is reduced to creating a mesh on a point set in 2-D Euclidean space.
- Spatial relationships with linear assets take into account every point that is part of its LineString geometry, not just the ends or middle.
- Distances through the spatial mesh are only approximate (i.e. not exactly the same as the Euclidean distance between arbitrary locations), which will reduce the risk of the model overfitting on proximity.

The main disadvantages of this model are as follows.

- Curved geometries will result in lots of point locations.
- Extent of polygon geometries are not stored: only the locations of other assets that are within them.
- Order of the locations along LineString geometries is not stored.
- Exact distances along the circuits may not be available, depending on which distance edges and length attributes are included.

While a model can only predict based on patterns that are included in the training data, this graph model can support tasks such as the following:

- Node classification:
  - network type for all assets using geometry and network type attributes only
  - cable/wire specification from cable/wire assets, locations and attributes only

    - connector point joint type from cable/wire assets
- Link prediction:
    - circuit membership even when they are missing conductor assets
    - electrical connections between assets that are not exactly co-located
    - location(s) associated with an added cable given one location, circuit ID and other selected attributes

Note that only locations that are already in the spatial mesh can be used for location link prediction.

Graph neural network

The graph neural network is the machine learning component that uses the graph data structure, described above, and the observed attribute values to predict correct values for each of the attributes of interest for each of the assets of interest. The main factors that lead to the selection of a graph neural network for this model are as follows. See the section on “rejected approaches” for more details.

- Relationships and feature vectors are equally important
- Should be able to predict for unseen data
- Need to scale to arbitrarily large areas in the future
- Derived embeddings are optimised for the target output attributes

The full spatial graph model is a heterogenous graph (or heterograph), which requires the use of relational graph convolutional (R-GCN) layers (arXiv:1703.06103), and types of layers derived from them, in the neural network model. Furthermore, each asset attribute to be predicted by the neural network requires a separate "head", resulting in a multi-headed architecture.

For the purposes of the PoC, all the node attributes to be predicted/corrected are categorical ones or are converted to categorical ones by binning the values. This has two main advantages for the PoC:

- The implementation is far simpler using only classification heads, rather than a mixture of classification and regression heads.
- The creation of “confidence scores” is easier for classification outputs than for regression ones.

The architecture of the neural network model for the PoC is shown in ***Figure 7***.

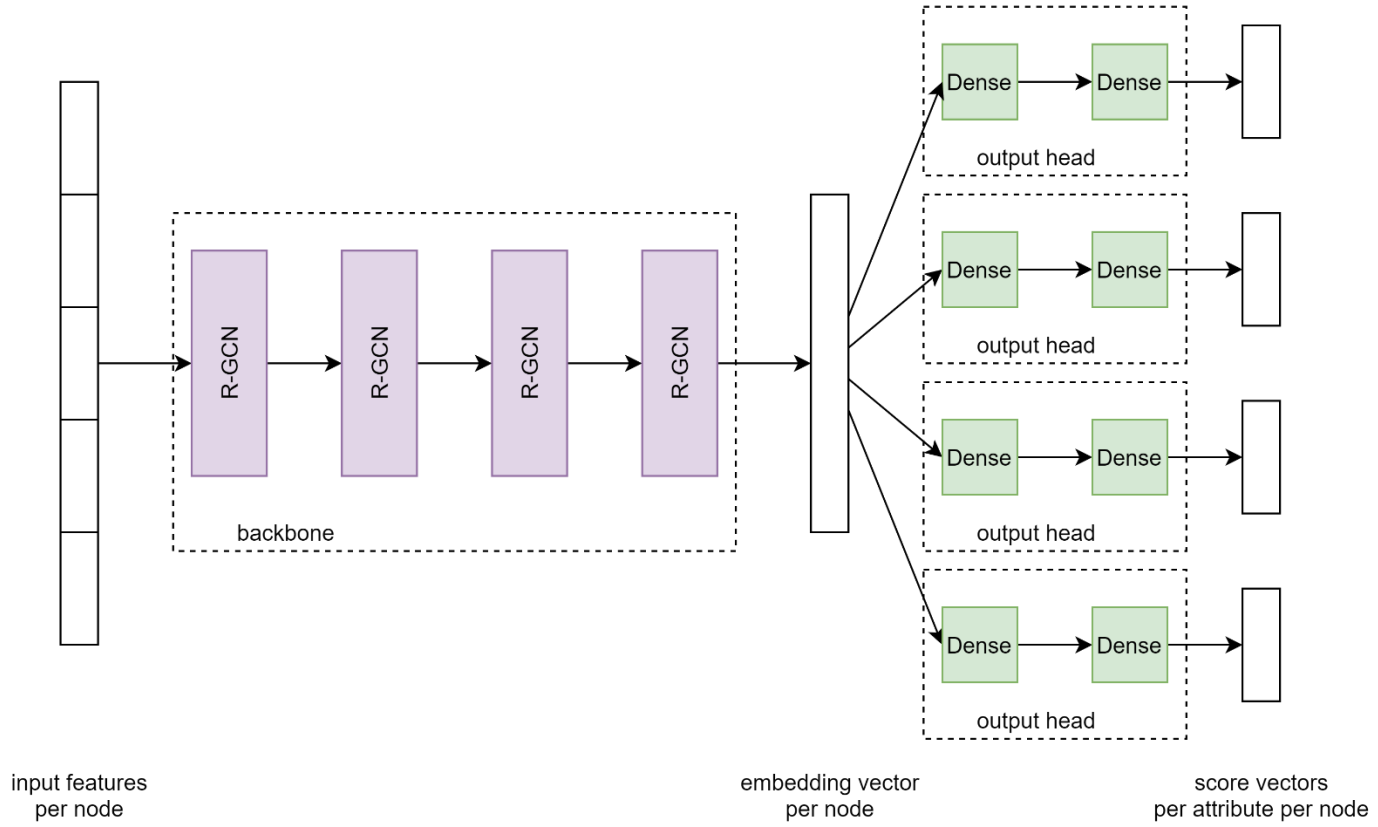

Figure 7: Architecture of the graph neural network model for SEAM PoC

The input vector to the neural network is the concatenation of the encoding of all the input attributes for each asset. Categorical attributes are encoded via a one-hot encoding, numerical attributes are encoded via a fixed scaling with a column added that is 1 when the value is not missing, and missing attributes encoded as all zeros. These inputs are used as the feature vectors for the asset nodes. The location nodes have all zeros as the feature vectors. For the location-location edges, the edge weight is set to be the inverse of the distance between the locations up to a configurable maximum value. All other edge types have no edge weights (equivalent to all fixed value).

The backbone of the neural network is comprised of used 4 weighted R-GCN layers (shown in ***Figure 8***) with sum aggregation and ReLU activation and without regularization[4]. The output of the backbone is an embedding for each node. The node embedding for each asset node of interest is then passed to the corresponding output heads for the corresponding attributes of interest. There is one output head for each output attribute and each output head is comprised of 2 dense layers with ReLU activation (except for the final layer). Dropout layers are present in the implementation but were not enabled in the current model training. The output of each head is a raw, unnormalized score per class for the corresponding attribute.

During training and evaluation, a cross-entropy loss is applied to each output head, with weights per class that are inversely proportional to the frequency of that class in the training set, in order to mitigate class imbalance problems, and then summed across the outputs to obtain a single combined loss. The loss is only evaluated for nodes where the original value is applicable and not missing. The neural network was optimized over 200 epochs using the Adam algorithm with a learning rate of 0.01 and a weight decay of 0.0005.

[4] Regularization (e.g. basis- and block-diagonal-decomposition) is not currently required, since the number of distinct edge types is small and, hence, the number of parameters in each heterogenous convolution is manageable.

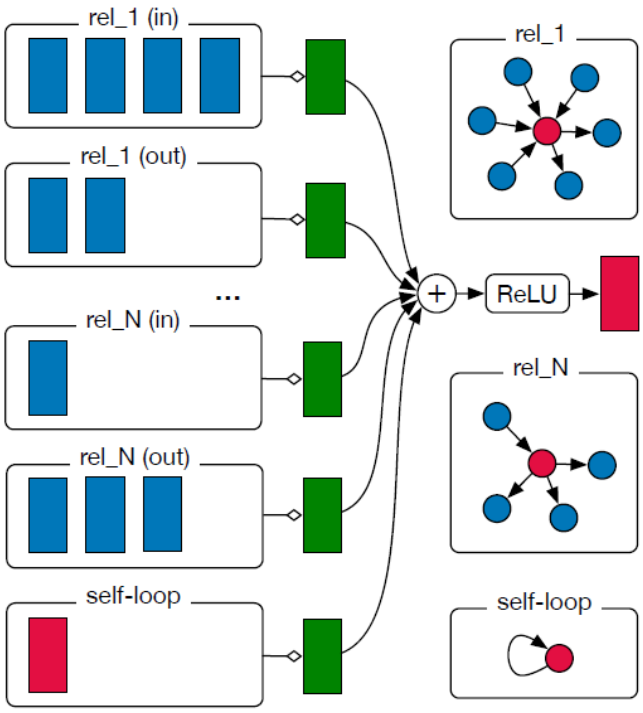

Figure 8: Diagram for computing the update of a single graph node/entity (red) in the R-GCN model from neighbouring nodes (dark blue) (from arXiv:1703.06103)

The graph neural network model is implemented using DGL 0.6.1 and PyTorch 1.8.1.

Model outputs and Confidence scores

As discussed above, the output of each head is a raw, unnormalized score per class for the corresponding attribute. From these are extracted the predicted (i.e. output) value and confidence scores.

For the PoC, exactly one output value is identified for each output attribute of interest. These are given by the index of the highest value in the scores vector for each attribute for each asset. This works well for the categorical outputs used for the PoC, since there are relatively few output classes for each attribute. A future version of SEAM could output multiple plausible values for investigation by the data steward.

Associated with this prediction, two confidence scores are extracted for each attribute for each asset: the “absolute score” is the highest value from the scores vector, and the “relative score” is the difference between the highest value and the second highest value. Each of these scores should be higher for predictions that the model is more confident in.

The purpose of the confidence scores is to allow the user to filter-out the low-confidence exceptions that are identified and investigate the high-confidence ones. In principle, this could be achieved using some combination of these two scores. However, for the PoC, it was found that the simplest approach was to use just the relative score, i.e. how much more confident the model was in the top output than the second top one.

In order to achieve the desired filtering, a score threshold must be specified. Since the scores are dependent on the trained model, the thresholds must be selected as part of the model tuning and they must be different for each output attribute. The model training selects 3 named sets of thresholds: “low”, “medium” and “high”, which the end user can choose between in the Excel UI[5]. These thresholds are tuned as follows:

1. The model is evaluated on the training fold of the training dataset

[5] The user can also select “none” to disable the thresholds.

2. For each attribute, the scores for assets where a change is detected (i.e. output is different from the input) are selected
3. For each attribute, the threshold is set at a fixed quantile of those scores

The quantiles used for the tuning are configurable via the model parameters file (See Appendix 2: Model 2 parameters file), and the default values are “low” = 0.2, “medium” = 0.5, “high” = 0.8. This means that, for example, after training, 20% of the changes identified in the training dataset had scores that were below the “low” threshold. Since the training dataset contains synthetic errors in the inputs, these quantiles refer to the synthetic errors, rather than the real ones.

Note that this threshold logic does not change the underlying model outputs: it just controls which changes appear in the detailed exceptions report and the reporting columns that relate directly to that.

Priority list of target attributes

The plan was to develop the model incrementally, as time allows; these stages are shown below, in priority order. Note that it is also a priority to explore multiple kinds of output attributes within the scope of the PoC. The current model only predicts the attributes in bold.

These stages are:

1. Voltage (node classification)

   These attributes are relatively straightforward to model, expect to find spatial patterns and common to many asset types, while still meaningful when taking a subset of asset types.

   - **Operational voltage**
   - **Network type**
   - Usage
   - Running 3 phase
   - Design voltage

   It is expected that this will not identify many (if any) real errors in the sample dataset. However, it will demonstrate the ability of the graph model to extract relevant patterns to identify and correct synthetic errors. This can easily be measured, since it can be assumed, with a high degree of confidence, that the original data match the ground truth.

2. Specification (node classification)

   These attributes are harder to model and include attributes that are known to have lots of missing data in the sample dataset, but are mostly only relevant to conductors.

   - Specification description (parts)
     - Number of wires
     - Number of cores
     - **Size**
     - **Material**
   - Number of wires
   - Armouring
   - Fluid-filled
   - PVC insulation
   - Capable 3 phase

   Since there are lots of missing values in these attributes in the original data, these model outputs are more interesting to the end users than the voltage-related ones. However, since, the information available about the underlying ground truth is less reliable, it is hard to accurately assess the model performance and the predictions will also be less reliable.

3. Circuits (link prediction)

   This attribute is important for building the electrical connectivity of the network and is known to have lots of missing values in the sample dataset.

   - Circuit identifier

   Using this model, it should be possible to link assets to circuits even in the presence of missing assets, by exploiting spatial patterns and other attributes. This would provide a complementary, data-driven approach to the current rules-based approach being developed by the mapping team.

Rejected Approaches

Initially, it was planned to use standard table-based imputation techniques, such as Multivariate Imputation by Chained Equations[6]. Some other promising approaches for data cleaning that use similar techniques to each other are Denoising Autoencoders and image restoration/inpainting. While many of these techniques are designed to fill missing data only, it was anticipated that these could be adapted to also detect and correct input values that are likely to be erroneous too.

However, the nature of the missing GIS data is such that these cannot be done for each asset independently without some representation of the local neighbourhood of that asset, and the local neighbourhoods of each asset in this situation are highly complex: they comprise many kinds of assets in different configurations on various spatial scales. Engineering spatial features to capture this would be complicated, expensive and inefficient. Instead, it is better to use graph-based techniques which are intended for precisely this purpose.

Traditional graph models for power networks are focussed on power systems analysis and network management, rather than on asset management. Hence, they tend to have the following characteristics:

- Focus on electrical properties
- Require complete electrical connectivity
- Simplify linear assets to abstract connections
- Ignore spatial relationships

Furthermore, it is difficult to just add spatial relationships to these graph models since it's hard to define a satisfactory distance relationship between the assets. For example, using the minimum distances between assets is useful for identifying assets that may be connected together, but it does not satisfy many of the requirements for a mathematical distance function, such as the triangle inequality: if cable A and B have a distance of 0 between them and cables B and C have a distance of 0 between them, then the distance between cables A and C is not necessarily 0. Alternatively, using the distance between the mid-points of the geometries makes it hard to identify which assets might be connected together; this option is also not robust to often arbitrary choices about where to end one conductor asset and start the next in the GIS.

The lack of a normal mathematical distance function means that it is also hard to decide which distance edges to include in the graph. Also, there will be lots of edges with zero or very short distances.

From all of this, it is clear that a new graph model is required that is focussed on predicting asset attributes and relationships and emphasises the spatial relationship between assets. This is the model described above.

In terms of machine learning on the graphs, there are two main kinds of approach: those that use direct encoding algorithms and graph neural networks. Direct encoding methods, such as node2vec and DeepWalk, are unsupervised techniques that calculate an embedding vector for each node that can then be used as the input to a decoder model that is specialised for the task at hand. This direct encoding technique has two significant drawbacks:

[6] Stef van Buuren, Karin Groothuis-Oudshoorn (2011). "mice: Multivariate Imputation by Chained Equations in R". Journal of Statistical Software 45: 1-67.

- It is a transductive technique, which means that when new data are added or a new graph is to be analysed, then new brand-new embeddings must be created and the decoder model must be retrained. This does not match our objectives.
- It is an unsupervised technique, which means that the embedding must capture all the information about the neighbourhood in the hope that it will be relevant to the decoder model. Our problem is complex, in that the node feature vectors are rich, and semi-supervised, in that we have some information about the target that can be exploited to create an efficient embedding.

In contrast, graph neural networks (GNNs) can avoid both these drawbacks. Hence, GNNs are selected for the ML component.

3.4. Model logic and data flow

Figure 9 shows the top-level model logic and data flow for the spatial graph model. It is takes various input parameters and files and creates various output files, as shown in the diagram.

Figure 9: Top-level model logic and data flow for spatial graph model (model 2)

The steps are as follows, with more details provided later.

1. **Initialisation**. Process input arguments, load parameter file and model file (if applicable) and prepare internal state of the model application.
2. **Load data**. Read EO asset data from file and prepare for modelling.
3. **Prepare graph data**. Extract and convert node and edge data from EO data. Generate synthetic errors (if applicable).
4. **Make graph**. Construct graph data structure and organise data for training and evaluation.
5. **Train model**. (If applicable) Train neural network model and score thresholds, and save to file.
6. **Evaluate model**. Calculate predictions for all assets in dataset and check score thresholds.
7. **Create reports**. Create detailed and summary output reports as CSV and GeoPackage (as appropriate).

When the model is run from the Excel UI or the CLI, then the console outputs are stored in an "outputs.txt" file in the reports directory, which can be read to monitor progress.

Initialisation

The initialisation processes the input arguments, loads the model parameter file and model file (if applicable) and prepares internal state of the model application. It also saves the input arguments as a JSON file.

Inputs

- Input arguments from Excel UI
- Model parameter file (optional for evaluation and prediction)
- Model file (except for training)

Outputs

- Initialised model application
- Model parameters structure
- Trained model, including sets of thresholds (evaluation and prediction)
- Application settings JSON file

Assumptions

- N/A

Processes

1. Convert and validate input parameters. Use default values for parameters that are not specified.
2. Read the model parameters file (if specified).
3. Read the model file (evaluation and prediction).
4. Create the feature generator objects from model parameters file (training only) or the model file (evaluation and prediction).
5. Initialise various internal data structures used in the spatial model processes from the data mentioned above.
6. Save the input parameters to JSON for traceability and reproducibility.

Techniques/Calculations

Model file

The model file is created as part of the model tuning stage (see below for details). It contains all the parameters that are required to restore and run the trained model.

It also contains all the parameters that are required to restart the training from where it finished, which could be used in the future to refine the model using more data rather than starting again. Note that there is no option in the Excel UI to restart the training.

Feature generators

The feature generators are internal classes that handle tasks related to the data for the individual input and output attributes for the neural network model. For example, there is functionality like extracting the relevant attribute from the EO data frames and converting it to the correct Pandas data type (including categorical), converting categories to integer levels, converting categories to one-hot encoded binary vectors, and the reverse conversions (e.g. numerical arrays to Pandas categorical). Encapsulating this functionality in one place means that conversion between user-friendly data types (e.g. Pandas data frame containing various data types) and other data types required by the model (e.g. PyTorch tensors containing a single data type) and back is easy and is guaranteed to use the same definitions, such as the feature name, category names and order, throughout the model.

This also has the functionality to define which input features are also output features from the model so that the same data structure can be used to process the model inputs and outputs.

As mentioned above, care is taken to ensure that missing data is always encoded as a level of -1 in the arrays of integer levels and as a vector of all zeros in the input arrays used by the neural network model. The functionality to add simulated errors to each attribute is also part of these classes, since the necessary actions can vary depending on the type of the attribute.

These objects need to be configured with the desired attribute definitions. For the training process, these definitions are taken from the model parameters JSON file, and the resulting objects are stored with the trained model in the model checkpoint file. For the evaluation and prediction processes, these objects are loaded from the model checkpoint file instead, in order to guarantee consistency with the trained model.

Load data

This step reads the GIS data from the input data file and prepares it for modelling.

Inputs

- EO data extract as Spatialite or Geopackage file

Outputs

- Dictionary containing processed asset data from EO

Assumptions

- Input data file is in any format that that can be read with Fiona[7].
- Bounding box parameter (if specified) overlaps with the data in the input data file.
- Each RWO is uniquely identified by its RWO id. Apart from the attributes that are specifically related to splitting the geometry into tiles and parts, all the attributes should be constant for each RWO.
- All materials of interest are specified in the model parameters file.

Processes

1. Read all layers from the input data file into a dictionary with one data frame per layer, using bounding box parameter (if specified) to filter the features to the specified area
2. Fix data types by converting known columns into relevant Pandas datatypes (e.g. boolean, categorical, numeric, datetime, etc)
3. Merge rows in each data frame to create one row per RWO id by merging the geometries of each row, discarding the attributes that are different for each part (e.g. “tilename”, “part”, “geometry_length”) and taking the only unique value for each of the other attributes.
4. Clip the geometries to the bounding box parameter (if specified)
5. Parse specification description attributes into parts

[7] Fiona is the Python interface to the vector formats in GDAL.

Techniques/Calculations

Merge RWO

The generic functionality to merge the rows per RWO is part of the GeoPandas library. It groups the features of each layer by the "id" column and, for each group, returns a feature with geometry corresponding to the union of all geometries in that group plus the only unique values[8] for the other attributes, after discarding those that are different for each part RWO. This ensures that there is only one asset node per RWO in the spatial graph.

The main advantages are that it removes the arbitrary distinction of crossing a grid square edge from the model and minimises the number of asset nodes. The disadvantage is that disconnects at the grid square edges or missing parts result in multi-geometry objects (e.g. MultiLineString) with no way for the graph model to know that there is a gap in the geometry.

Specification parsing

The specification parts attributes are extracted by parsing the specification description field of the associated asset. These are:

- cable: specification_description
- wire: specification_description_1 (note that specification_description_2 is ignored)

From these strings, the following parts of the specification are extracted:

- material: return the longest pattern matching a pre-calculated list[9] that can be found within the string, ignoring case.
- size: 2 or more digit integer or decimal number as a whole word, then converted to a number and numbers less than 1 are converted from square inches to square millimetres.

For the material, the pre-calculated list is stored in the model parameters file that is read in the initialisation step. By taking the longest match, it will deterministically match more specific material like "solidal" or "cad cu" rather than less specific ones like "al" and "cu". These patterns were extracted from the company directives[10].

For the size, the largest imperial size encountered is 0.75 square inches. Hence, the threshold for detecting imperial vs. metric sizes is set at "1".

When the relevant pattern does not match anything, it is treated as missing.

For example:

- "25 hyb" → material = "hyb", size = 25
- "0.04" → material = NA, size = 25.8064
- "2c 35 abc" → material = "abc", size = 35
- "3 x 185 1c txal epr" → material = "al", size = 185
- "4 s/c" → material = "s/c", size = NA
- "2w sv unknown from unattributed" → material = NA, size = NA

Prepare graph data

This step extracts the node and edge data from EO data for the spatial graph model described in section 3.3 and converts it into appropriate data types. It also generates synthetic errors in the input data (training and evaluation) and

[8] For each of these attributes of each RWO, the code checks that the values for each part are equal and then returns that one value.
[9] The pre-calculated list of possible
[10] WPD Company Directive STANDARD TECHNIQUE: SDB/3 and SDB/4

selects which training fold each asset will be part of (training only). These calculations are intended to be independent of the graph neural network design.

Inputs

- Dictionary containing processed asset data from EO

Outputs

- Dictionary containing node and edge data for the graph

Assumptions

- All categorical values are defined in the model parameters file.
- The breakpoints for digitising the specification size attribute are defined in the model parameters file.
- For synthetic error generation, see the detailed section below.
- For training folds generation, see the detailed section below.

Processes

1. Extract assets and asset attributes from EO data, including circuit_id and geometry
2. Extract locations from the asset geometries
3. Create position edges, i.e. edges between assets and locations
4. Create spatial mesh of location nodes, i.e. distance edges
5. Convert asset attributes to numerical arrays
6. Add simulated errors to asset attributes (evaluation and training)
7. Split asset nodes into training, validation and test folds (training only)

Techniques/Calculations

Spatial edges

There are various potential approaches for the creation of spatial edges. Ideally, this should provide a connected graph[11], since this allows messages to be passed between any two location nodes, in principle. The spatial edges should not be too dense, since this increases the size of the graph and the time to execute the model, or too sparse, since this decreases the message-passing ability of the network.

The current model uses a Delauney triangulation for the spatial edges. This has the following properties:

- Complete triangulation of the location nodes.
- Mathematical definition: maximises the minimum angle of all the triangles in the triangulation.
- Seems to strike a good balance for the number of edges per node.
- Fast to compute: quasilinear[12] time complexity. Implemented natively in most geometric libraries[13].

Alternative approaches include the following, which could be explored as part of the next phase of development.

- Nearest *k*-neighbours: Not guaranteed to produce a connected graph. Choice of *k* might need to be different in different situations, e.g. close to substation vs along a transmission line.
- All neighbours within given distance: Not guaranteed to produce a connected graph. Choice of distance might need to be different in different situations, e.g. urban vs rural.

[11] i.e. if there's a path between every pair of location nodes
[12] $O(n \log n)$ where n is the number of locations
[13] including shapely, PyGEOS and PostGIS

- Alternative geometrical constructs, like relative neighbourhood graph or Gabriel graph, tend to be more sparse than the Delauney triangulation (in fact most are subgraphs of it) so it's only worth considering these if the current model is too expensive.

The Euclidean length of each edge is stored as an edge property.

Synthetic errors

Synthetic errors are only added to the input data as part of the training and evaluation processes. The creation of synthetic errors is based on the following assumptions:

- Synthetic errors only affect asset node feature vectors.
- Only the attributes predicted by the model can have errors, i.e. attributes like geometry are always correct.
- The possible patterns of error are "all missing", "one missing" and "one corrupt".
- When an attribute is corrupt, it can take any other permitted value for that attribute, independent of the original values (except that it cannot be the same) and the other attributes.
- For the "one missing" and "one corrupt" errors, the choice of attribute affected is uniformly at random.
- Errors are uncorrelated between assets.

Hence, the synthetic error generation works by selecting a random proportion of asset nodes to apply each type of error pattern to without replacement. The default proportions are 15% "all missing", 15% "one missing" and 20% "one corrupt", which leaves 50% of assets with the original values unmodified. If the selected attribute to modify is already missing, then it is left as missing.

Training folds

Training folds are only used as part of the training process. The creation of the training folds is based on the following assumptions:

- Assets to include each fold can be chosen at random (i.e. without any spatial correlation)
- Each fold should have a similar composition of output attribute values to address issues related to the balance of the classes.

Hence, the training fold generation works by grouping the assets by the output attribute values and assigning them different groups IDs, then using a stratified train-test split based on those group IDs to split the assets between the required folds. The default proportions are 20% validation, 30% test, which leaves 50% for training.

Splitting the assets like this ensures that the folds have a similar composition also in terms of spatial distribution; although, every validation and test asset will be spatially close to some training assets. This is fine, possibly even beneficial, for a model where the objective is to apply it to the same area as was used for training. If the model is to be optimized for its application to unseen data instead, as might be the case in a future phase of development, then it would be better for the validation and test folds to be entirely separate graphs; however, then it is challenging to ensure that the compositions of those folds are sufficiently similar.

Make graph

This step constructs graph data structure described in section 3.3 from the tables created above and organises data for training and evaluation. Some of these calculations are specific to the graph neural network architecture selected.

Inputs

- Dictionary containing node and edge data for the graph

Outputs

- Spatial graph data structure

Assumptions

- N/A

Processes

1. Construct graph data structure from node and edge data frame, including reverse edges and self-loops
2. Encode input data for all nodes (e.g. one-hot encoding) using the feature generators
3. Store all relevant node and edge features in node and edge data dictionaries as PyTorch tensors
4. Calculate edge weights for distance edges (i.e. location–location)
5. Set node features for node types without data to all zeros
6. Create training, validation and test versions of the node feature tensors with all features of the asset nodes for the higher-level test sets set to “missing” and the data unmodified for the other node types (training only)

Techniques/Calculations

Spatial edge weights

The simplest way to factor the spatial edge distances into the graph neural network model is to convert them to weights, then using weighted graph convolution layers, as mentioned in section 3.3.

The objective of the edge weights is to make the neural network place higher importance on nearby locations than on ones that are further away. Hence, the weight for the edge between u and v is calculated as follows:

$$w_{u,v} = \frac{1}{\max(d_{u,v}, d_{\min})}$$

Where, u and v are location nodes, $d_{u,v}$ is the Euclidean distance between them and $d_{\min}$ is a minimum distance parameter, which limits the maximum weight that can be applied to a weight, i.e. all edges shorter than $d_{\min}$ are treated as if the distance was $d_{\min}$. The default value for $d_{\min}$ is 1 metre.

Training, validation and test feature vectors

This step is only required during the training process, when the assets are split between training folds. This is an important step to avoid data leakage between the training, validation and test datasets.

Since the validation and test assets are all part of the graph used for training and the feature vectors for those nodes are highly correlated with the labels (i.e. original values), it is necessary to mask that information during training (and validation) in order to avoid the risk that the neural network will memorise it and bias the validation and testing results. Hence, the feature vectors for asset nodes that are in the validation and test folds must be masked for training, and those in the test fold must be masked for validation. Here, “masking” means “replacing with zeros”[14], since it is not possible to remove asset nodes from the graph without fundamentally affecting the graph structure.

The masking is achieved by creating separate node feature arrays for training, validation and test, that contain only feature vectors for suitable nodes, with the values for the remaining assets set to zero.

One consequence is that the validation and test graph both contain more information, especially less missing data, than the training graph, so it is not unusual for the validation and test accuracy to exceed the training accuracy (or equivalently the loss to be lower).

Train model

This step is only run as part of the training process. It trains the graph neural network model and the score thresholds. The outputs are saved to the model file. The design of the graph neural network model, including the loss and optimizer, is described in section 3.3 above.

[14] Note that the feature generators (see above) were designed such that all zeros in the encoded vectors means a missing value.

Inputs

- Spatial graph data structure

Outputs

- Trained model, including sets of thresholds
- Model checkpoint file

Assumptions

- Spatial graph data structure contains all the data required for the model training and evaluation.
- Good model performance can be achieved without early stopping or other hyperparameter optimization.
- Parameters for threshold fitting are provided in model parameters file.

Processes

1. Configure neural network model
2. Configure loss functions
3. Configure neural network optimizer
4. Train model using training fold with validation fold for monitoring
5. Fit score thresholds using training fold (as described above)
6. Evaluate model on test fold to check observed performance during training
7. Save model checkpoint file

Techniques/Calculations

Validation and Testing

The training fold is used to update the parameters of the neural network model, via the standard process of model evaluation, loss evaluation and back-propagation.

In every training iteration, the performance on the validation fold, using the validation feature vectors, is evaluated to give an independent assessment of the performance of the current model. This is a more representative measure of the model performance than the performance using the training fold and the training feature vectors. In principle, the performance using the validation fold could be used to optimize the model hyperparameters, such as number and size of neural network layers, learning rate, number of epochs[15], etc. However, this is not implemented for this model.

The test fold is only evaluated at the end of the training. This performance should only be used as a check on the performance statistics observed during, and at the end of, the training for the training and validation folds. The model parameters should not be modified in order to maximise the test performance, otherwise it will result in data leakage between this independent test data and the training and validation data. If the model parameters are to be manually optimised, then the final validation performance values or the results of the separate evaluation process (i.e. with different data errors or on a different area) should be used instead.

Accuracy statistics

An accuracy statistic is required for monitoring the training progress, since this is more easily understood by the user than the loss. Since this model have several output heads, these must be combined to create a single accuracy score.

Since the model performance is expected to be good, the combined accuracy is defined in the most conservative way possible, i.e. all the predictions for a given asset must be correct for it to count as correct for the accuracy statistics.

[15] i.e. stopping the training when the validation accuracy stops improving.

Only attributes that are valid for that asset type and have non-missing true (i.e. original) values can be checked; these are referred to as “valid attributes”. Some assets have no valid attributes and these must be excluded from the calculation.

Hence, the combined accuracy statistic is defined as “proportion of assets with all their valid attributes predicted correctly out of those with any valid attributes”.

Model checkpoint file

The model checkpoint file which is loaded as part of the model initialisation for the evaluation and prediction processes, must be created as part of the model training process. It contains all the parameters for the trained model, including the fitted score thresholds.

The file also contains all the parameters relating to the loss functions and optimizer, which are required to resume the training later, either because training was interrupted or to refine a model for a more specific application. While this process is not implemented in this version of the model, the necessary parameters are present in the model checkpoint file.

Evaluate model

This step calculates predictions for all assets in input dataset using the trained model and checks the selected score thresholds. This is identical to the evaluation process that is part of the model training except that the whole dataset is used without splitting into folds and the scores are checked using the selected thresholds out of the ones fitted during the training.

Inputs

- Spatial graph data structure
- Trained model, including sets of thresholds
- Name of selected thresholds

Outputs

- Dictionary containing results of model evaluation

Assumptions

- Selected thresholds matches one of the sets of thresholds from the model training.
- Model evaluation can use all the input data without holding back data from validation or testing folds.

Processes

1. Calculate the raw scores per attribute for all asset nodes using the neural network model
2. Calculate predicted (output) values and confidence scores for every attribute and asset in the dataset
3. Compare the confidence scores with the selected thresholds
4. Compare with original values and create evaluation outputs (evaluation and training)

Techniques/Calculations

N/A

Create reports

This step creates detailed and summary output reports as CSV and GPKG (as appropriate).

Inputs

- Dictionary containing processed asset data from EO
- Dictionary containing node and edge data for the graph
- Spatial graph data structure
- Dictionary containing results of model evaluation

Outputs

- Model outputs report
- Exceptions report and summary
- Evaluation report and summary (training and evaluation)
- Classification report (training and evaluation)

Assumptions

- GIS applications used by WPD can read GPKG files

Processes

1. Construct model outputs data frame, including converting raw outputs back to user-friendly ones using feature generators
2. Extract exceptions report data frame and summary from model outputs data frame
3. Extract evaluation report data frame and summary from model outputs data frame (training and evaluation)
4. Create classification report from evaluation report data frame (training and evaluation)
5. Write all reports as CSV and GPKG (where applicable)

Techniques/Calculations

Reports

The spatial model produces some reports with detailed results at the level of the attributes of each asset and some summary reports that summarise the results across the entire area covered by the task. These are summarised in the table below.

The detailed results are saved as both CSV files (for use with analytical software[16]) and GeoPackage files (for use with GIS software). The CSV files have one row per attribute per asset, while the GeoPackage files have one layer per attribute, then one feature per asset. The summary results are saved as CSV only.

For the training and evaluation tasks, the evaluation report, evaluation summary and classification report are most relevant. For the prediction task, the exceptions report and exceptions summary are more relevant.

Report Name	Purpose	Description
	Detailed	
`model_outputs`	Understanding all the outputs from the model	All relevant inputs and outputs from model. All the other reports are a subset or summary of the data in this report.
`exceptions_report`	End user investigating suggested changes to the GIS data	All rows from model_outputs where output value is different from input value and score meets criteria. Excludes columns that are not interesting for the end user.
`evaluation_report`	Understanding the behaviour of the model in response to simulated errors	All rows with non-missing values in the original data. Only produced when synthetic errors are added to the input data.
	Summary	

[16] The CSV reports can be opened in Excel but be aware that Excel may incorrectly interpret some of the values. For example, it replaces “1/1” with “01-Jan” and “10-20” with “Oct-20”.

Report Name	Purpose	Description
exceptions_summary	End user understanding the distribution of identified errors across asset types and attributes	Number of rows with each error_code value (see below) for each asset type for each attribute.
evaluation_summary	Understanding the overall performance of the model on simulated errors for different asset types and attributes	Accuracy (proportion exactly correct) for each asset type and attribute for each error_code. Only produced when synthetic errors are added to the input data.
classification_report	Understanding the overall performance of the model on simulated errors for different attributes and attribute values	Classification report (see below) for each attribute. Only produced when synthetic errors are added to the input data.

Table 6: Model 2 Output Reports

Columns

These are the columns of the model outputs file. All the other reports are a subset or summary of these data.

- **Exceptions report**: all rows with “changed” = True and “score_ok” = True
- **Exceptions summary**: number of rows for each combination of “asset_type”, “attribute_name” and “error_code”
- **Evaluation report**: all rows with “true_ok” = True
- **Evaluation summary**: average of “correct” column for each combination of “asset_type”, “attribute_name” and “error_code” out of all rows with “true_ok” = True
- **Classification report**: classification report (see below) for each “attribute_name” using all rows with “true_ok” = True

The “Excluded from” column indicates when each column is not included in the outputs. For example, it may be constant or not interesting for some reports and it may not be produced for all tasks.

Column	Description	Excluded from
asset_id	Index of associated asset node in spatial graph	Exceptions report: internal
rwo_id	Real world object ID from EO	
asset_type	Asset type from EO	
circuit_id	Circuit ID from EO (if available)	
geometry	Geometry from EO, merged across tiles and parts (WKT)	
attribute_name	Name of attribute for this line	
input_value	Input value of attribute to model (category)	
output_value	Output value of attribute from model (category)	
changed	Whether output value is different from input (Boolean)	Exceptions report: always True
score_abs	Absolute score from model (maximum of scores per category)	
score_rel	Relative score from model (difference between scores for top 2 categories)	
score_ok	Whether score meets criteria (Boolean)	When thresholds is ”none”: treated as True Exceptions report: always True
error_code	Error code for output (see below)	
error_code_simple	Simplified error code for output (see below)	Exceptions report: internal
true_value	Original value of attribute in EO: treated as ground truth by the model (category)	Prediction task
true_ok	Whether original value is not missing (Boolean)	Evaluation report: always True Prediction task
input_modified	Whether input value is different from original value	Prediction task
error_code_true	Simplified error code if output were the original value (i.e. ideal error code value)	Prediction task
correct	Whether output value matches the original value (treated as ground truth)	Prediction task
error_type	What kind of error was added in the input data (see below)	Prediction task
fold	Which training fold the asset is part of (see below)	Prediction or Evaluation task

Table 7: Model 2 Output Columns

Enumerations

The enumerated columns listed above have the following possible values:

Code	Description	Meaning
error_code: Kind of error detected by the model.		
`no_error`	No error	Output value matches the input value.
`missing_value`	Missing value	Input value was missing.
`wrong_value`	Wrong value	Output value different from input value, which was also not missing.
`missing_value`	Missing value - low score	Missing value, but score for predicted value does not meet criteria. (i.e. low confidence score)
`wrong_value`	Wrong value - low score	Wrong value, but score for predicted value does not meet criteria. (i.e. low confidence score)
error_code_simple: This is like error_code but ignoring the score criteria.		
`no_error`	No error	Output value matches the input value.
`missing_value`	Missing value	Input value was missing.
`wrong_value`	Wrong value	Output value different from input value, which was also not missing.
error_type: Kind of error that was simulated in the data.		
`no_error`	No error	Input data matches original data exactly.
`missing_all`	All values missing	All output attributes were replaced with missing values in the input data.
`missing_one`	One value missing	One output attribute (at random) was replaced with a missing value in the input data. If the selected attribute is already missing, then it is not changed.
`corrupt_one`	One value corrupt	One output attribute (at random) was replaced with a random different value in the input data. If the selected attribute is already missing, then it is not changed.
fold: Which training fold the asset is part of.		
`train`	Training set	Asset labels are used for tuning the model.
`validation`	Validation set	Asset labels are used to supervise the model tuning.
`test`	Test set	Hold-out data for final model testing.

Table 8: Enumerations in Model 2 Outputs

Classification report

The classification report file is created by concatenating the multi-label classification reports for each classification output (i.e. one per attribute) from the model using all of rows that are part of the evaluation. Remember that this assumes that the original values are the ground truth.

Each of the multi-label classification reports are constructed as follows:

- There is one row for each associated attribute value, which is referred to as the “row value” below
- True positives (TP) is the number of rows where the output and true values are both equal to the row value
- False negatives (FN) is the number of rows where the true value equals the row value but the output value does not
- False positives (FP) is the number of rows where the output value equals the row value but the true value does not.
- True negatives (TN) is the number of rows where neither the output or true values are equal to the row value
- Precision is the accuracy out of the rows with output values equal to the row value = TP / (TP + FP)
- Recall is the accuracy out of the rows with true values equal to the row value = TP / (TP + FN)
- F1-score is a balanced accuracy metric, which is the harmonic mean of the precision and recall = 2 TP / (2 TP + FP + FN)
- Support is the number of rows where the true value equals the row value = TP + FN

3.5. Code structure

The main parts of the spatial model application are implemented as follows.

Top-level model application logic

- `wpd_seam.graph.main`: top-level model application logic and Excel UI entry-point interface.
- `wpd_seam.graph.__main__`: command-line interface to model application.

Data import/export and preparation

- `wpd_seam.data.access`: routines for reading and writing database-like data files, including SQLite databases and Fiona vector mapping files.
- `wpd_seam.data.eo`: data preparation routines specific to the EO data files.
- `wpd_seam.features.specification`: routines for parsing specification description strings.

Spatial graph model

The project source code contains the implementation for 4 incremental versions of the spatial graph model, with each one adding more features and functionality to the model. The code for each version of the spatial graph model is in a separate sub-module. The current version is v4 and it is the only maintained version. Earlier versions are also missing some of the final functionality, such as output reports and threshold tuning.

- `wpd_seam.graph.v4`: *nodes*: locations, assets; *edges*: distance, position; *attributes*: network type, operational voltage, specification material, specification size.

Each version of the model has the functionality divided between the following functions and classes within the corresponding sub-module.

- `prepare_features`: routines to prepare the relevant feature generators for this model.
- `SpatialGraphGenerator`: routines for constructing the graph data structure from the EO database.
- `ModelWrapper`: routines for constructing, training and evaluating the neural network model on the spatial graph data.

Common functionality shared between the different versions of the model is implemented in the following modules:

- `wpd_seam.graph.features`: routines for extracting and processing features that go into and come out of the spatial graph model.
- `wpd_seam.graph.neural`: neural network models used in the spatial graph model.
- `wpd_seam.graph.spatial`: common routines for the spatial graph model implementation.
- `wpd_seam.graph.outputs`: routines for preparing, saving and loading output reports.

4. User Interface

4.1. Description of the User Interface

The User Interface (UI) is an Excel based tool designed to enable a user to run the SEAM models without the need to directly interact with the Python code and to make adjustments to key model parameters. The UI is launched from an Excel workbook and the user navigates the tool through a series of VBA user forms to perform tasks and run the models.

The following tasks can be performed through the UI:

- Configure the Python settings to run the models locally
- Select which model is to be run
- Select the key data input files
- Set the file path for output reports
- Determine the parameters and/or thresholds to be used for a model run
- Setup multiple profile settings for each model and duplicate existing profiles
- Record a log of model runs carried out by the user

These following sections describe the design of the UI and provide user instructions for setting up and running the two SEAM models.

4.2. High-level design

The following diagram outlines at a high-level the relationship between the different user forms that are available to a user of the tool and how these can be navigated to define settings, adjust parameters, and run the models.

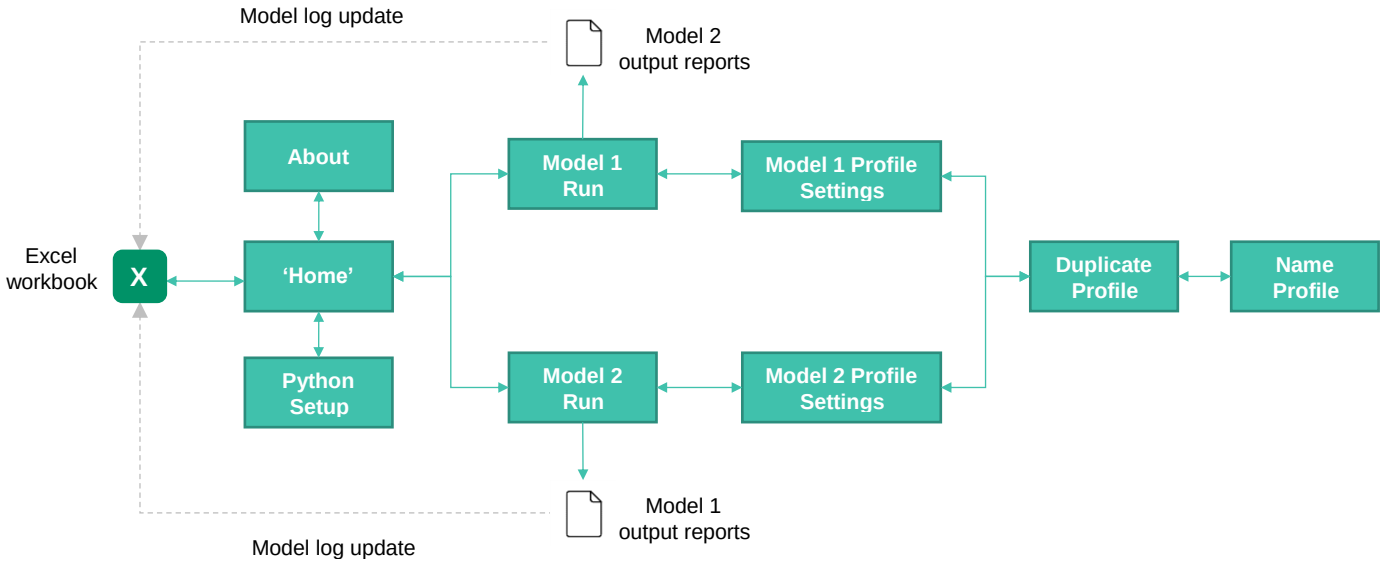

Figure 10: User Interface high-level design

When setting up the UI and SEAM models for the first time, the user must perform the following two activities:

1. Configure Python setup (see Section 4.3.3 for further information)

2. Create a profile setting with the intended input and output settings and model parameters (see Section 4.3.5 for further information)

4.3. User forms

4.3.1. Home

Description

Landing user form following launch of the model. From here the user can initialise/adjust the local Python setup and access the SEAM models.

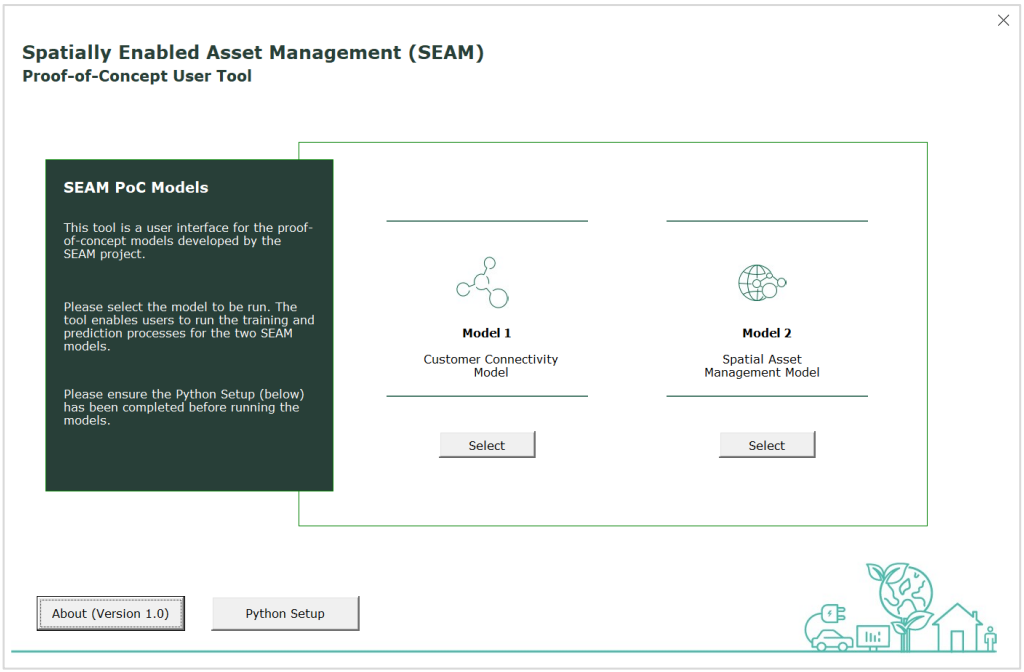

Figure 11: Screenshot of 'Home' user form

Tasks

The following tasks can be performed by user:

- Access the ‘About’ information associated with the version of the model
- Access the ‘Python Setup’ to initialise/adjust settings
- Select and launch Model 1 and Model 2

User-defined settings

There are no user-defined settings available in this user form.

4.3.2. About

Description

Presents key information associated with the version of the model being used. There is a developer option to change this information (see Section 4.5).

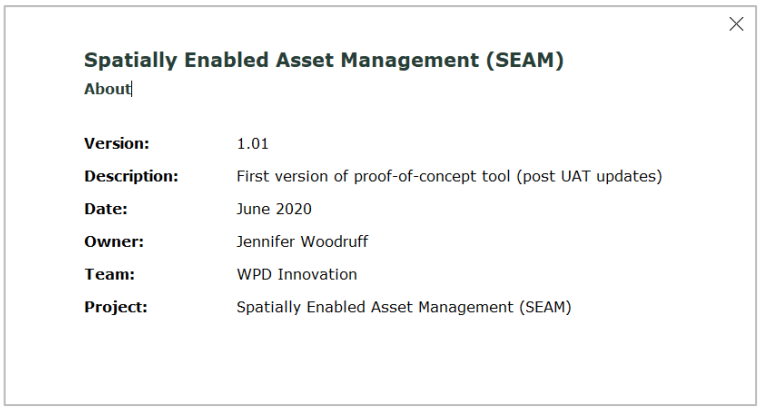

Figure 12: Screenshot of 'About' user form

Tasks
No tasks are performed within this user form. Displayed for information purposes only.

User-defined settings
There are no user-defined settings available in this user form.

4.3.3. Python Setup

Description
User form for defining the Python configuration settings. The model uses xlwings which is an open source Python library that makes it easy to call Python from Excel and vice versa.

The model uses the RunPython function in VBA to call the Python scripts that run the SEAM models and pass the variables that are defined by the user in the profile settings.

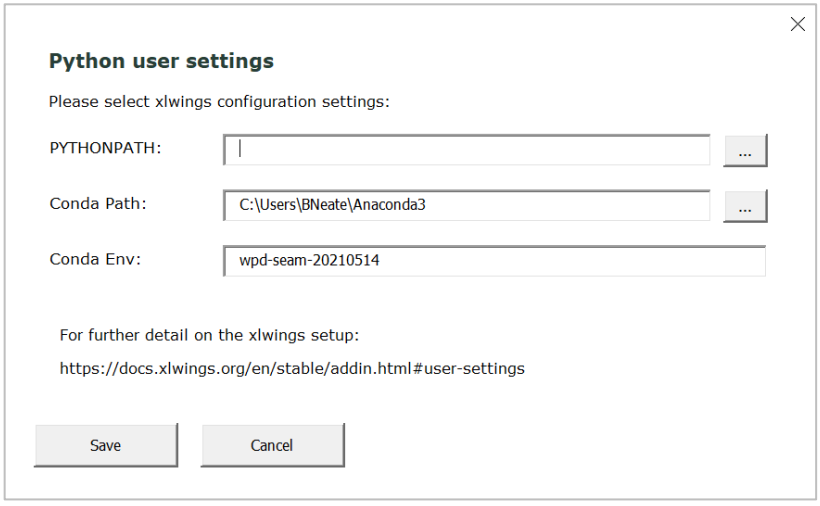

Figure 13: Screenshot of 'Python Setup' user form

Tasks
The user if presented with a user form containing the option to define the xlwings Python and conda settings (these overwrite the global settings). These need to be correctly defined for a local setup to run the models.

User-defined settings
See Appendix 1: Python setup instructions for the steps to follow for the Python setup. The solution is designed so that an environment can be installed from snapshot (e.g. for non-networked PC).

4.3.4. Run model

Description

Accessed from the ‘Home’ user form. This is used to execute a run of the model selected by the user based on a specified profile. The same user form structure is applied for each of the SEAM models.

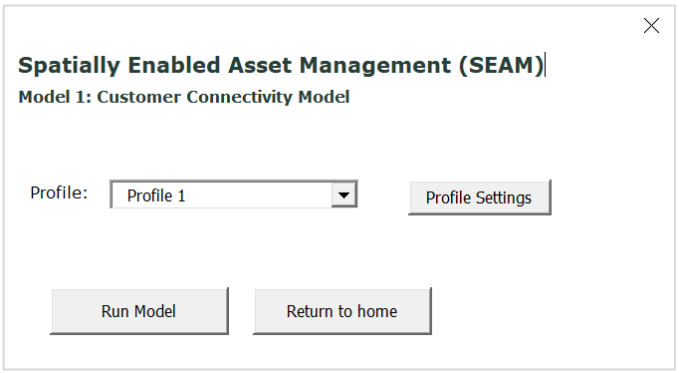

Figure 14: Screenshot of 'Run Model' user form

Tasks

The following tasks can be performed by user:

- Access the ‘Profile Settings’ user form to create or amend the model profiles
- Select the profile to be used for a model run
- Execute a model run

User-defined settings

The profile to be used for the model run is selected from the Profile dropdown menu.

4.3.5. Profile settings

Description

Accessed from the ‘Run Model’ user form. This user form is used to create, amend, and save profiles that are used to run the models. The same user form structure is applied for each of the SEAM models.

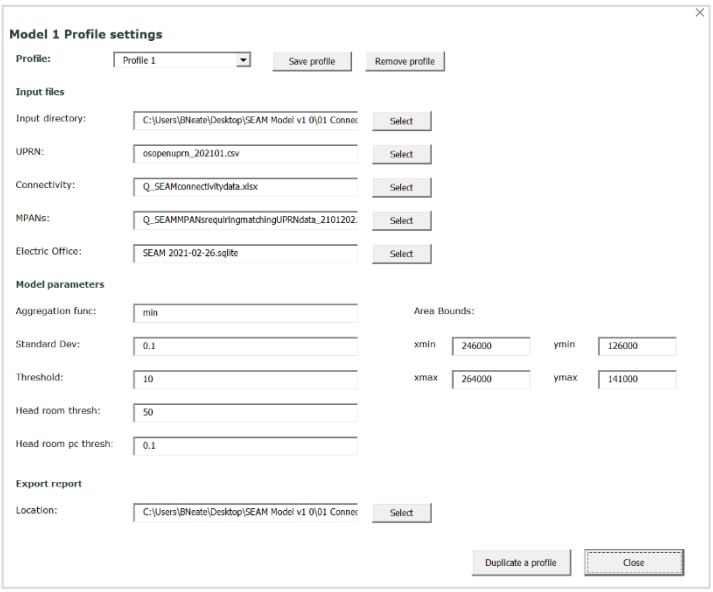

Figure 15: Screenshot of 'Profile Settings' user form for model 1

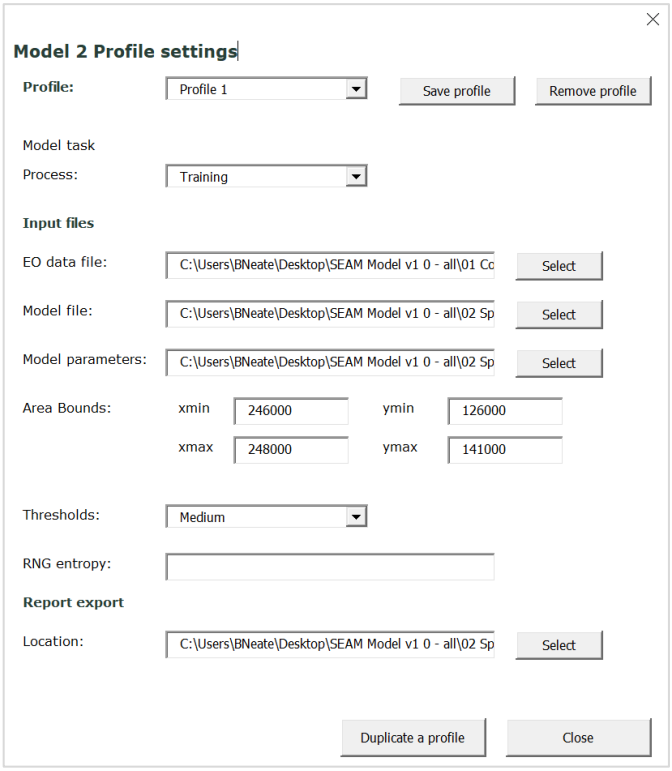

Figure 16: Screenshot of 'Profile Settings' user form for model 2

Tasks

The following tasks can be performed by user:

- Define inputs, outputs and parameters and save a new profile
- Amend inputs, outputs and parameters of existing profiles and save changes
- Remove existing profiles
- Access the 'Duplicate Profile' user form

User-defined settings

Model 1

Setting	Type	Description
Input directory	Folder path	Provides the path to the directory which contains the input data files.
UPRN	Filename	Name of the file in the input directory which contains the OS Open UPRN dataset (.csv)
Connectivity	Filename	Name of the file in the input directory which contains the CROWN customer connectivity dataset (.xlsx)
MPANS	Filename	Name of the file in the input directory which contains the CROWN customer connectivity dataset including MPANs (.xlsx)
Electric Office	Filename	Name of the file in the input directory Electric Office GIS database (.sqlite)
Aggregation func	String: min, mean, max	Determines how capacity backfilling should be applied: - Min / mean / max of circuit and area cable or wire of the same type - Min / mean / max of neighbours which backfilling synthetic cables / wires in the circuit

Standard dev	Float	Determines the % of the mean demand peak as the standard deviation when calculating the design demand at the consumer: Design demand = Mean demand + 1.28 x σ Where σ = Mean demand x Standard dev This design demand corresponds to a 90% probability of meeting the demand within the design voltage regulation – an acceptable level of risk (ACE 105).
Threshold	Float	This is the distance threshold (in metres) for the distance between the substation location and the nearest node on the circuit.
Head room thresh	Float	This is the head room threshold (kW) for flagging a wire / cable as a violation when (capacity – flow) < threshold
Head room pc thresh	Float	This is the percentage head room threshold for flagging a wire / cable as a violation when (capacity – flow) / (capacity) < Head room pc thresh
Area bounds	X, Y coordinates	Defines the minimum and maximum X and Y coordinates for the area bound to be included in the model run. For example, the training area has a bounding box with X in [248000, 264000] and Y in [126000, 141000].
Export location	Folder path	Provides the folder path to export the output reports following completion of a model run.

Table 9: Description of user profile settings for model 1

Model 2

Setting	Type	Description
Process	List: Training, Evaluation, Prediction	**Training:** This process is used to obtain a new trained model that can be used to generate predictions. The model is trained against the data for a selected region taking the original data as ground truth and adding synthetic errors to form the input data. The GNN model is iteratively optimized to increase the number of output values from the model that match the true (original) values given the corrupted input data. **Evaluation:** This process is used to measure the ability of a pre-trained model to make correct predictions, given data with synthetic errors. As for the training process, it uses the original data for the selected region as ground truth and adds synthetic errors to form the input data. It can either be applied in a transductive (i.e. same region as training, but different synthetic errors) or inductive (i.e. different region from training) manner. **Prediction:** This process is used to identify errors in the original data and obtain suggested corrections. It uses the pre-trained model to generate predictions using the original data for a selected region as the input data. This process outputs the list of suggested corrections that meet the necessary scoring thresholds. It can either be applied in a transductive (i.e. same region as training) or inductive (i.e. different region from training) manner.
EO data file	File path	Provides file path to the Electric Office dataset (SQLite).
Model file	File path	Note this file is an output for the training process and an input for the evaluation and prediction processes. Since it is

		created by the training, it does not need to exist when running that process and, if it does exist, then it will be overwritten. If it does not exist, then manually add the file extension to the path in the user form text box.
Model parameters	File path	Provides file path to read the model parameters. These are parameters used by the model that are not defined in the UI (e.g. threshold quantile settings). They are configurable by making changes directly in the file.
Area bounds	X, Y coordinates	Defines the minimum and maximum X and Y coordinates for the area bound to be included in the model run. For example, the training area has a bounding box with X in [248000, 264000] and Y in [126000, 141000].
Thresholds	List: Low, Medium, High	Determines the threshold for filtering the output reports based on confidence levels. The quantiles used for the tuning are configurable via the model parameters file, and the default values are “low” = 0.2, “medium” = 0.5, “high” = 0.8. This means that, for example, after training, 20% of the changes identified in the training dataset had scores that were below the “low” threshold. The user can also select “none” to disable the thresholds
RNG entropy	Numeric	Random seed sequence used for simulation of errors during the training and evaluation processes. To recreate the random result, the seed sequence can be input here – the value is included in the output text file following a model run. If it’s left blank then a new random value is created.
Export location	Folder path	Provides the folder path to export the output reports following completion of a model run.

Table 10: Description of user profile settings for model 2

4.3.6. Duplicate profile

Description

Accessed from the ‘Profile Settings’ user form. This is used to duplicate an existing profile and save under a new profile name.

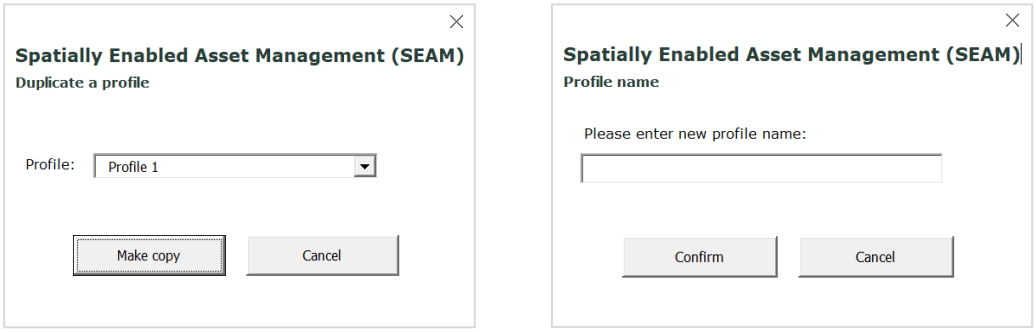

Figure 17: Screenshots of 'Duplicate Profile' user forms

Tasks

The following tasks can be performed by user:

- Select an existing profile to duplicate
- Create and save the duplicate with a new profile name

User-defined settings

The user selects the profiles to be duplicated from the dropdown menu on the initial user form and the new profile name is entered on the subsequent user form.

4.4. Model log

Following the completion of a model run a log is maintained in Excel workbook in the 'Log' worksheet. The following details are stored in the log:

Field	Model 1	Model 2
Model	The model that was run (Model 1 or Model 2)	
User	The username from System Environment Variables	
Settings	Profile name	
Report	Folder path for output reports	
Timestamp	DD/MM/YYY HH:MM:SS of model run	
Parameter (1-5)	Aggregation function; Standard deviation; Threshold; Headroom threshold; Headroom pc threshold	N/A – blank
X (min, max); Y (min, max)	Area bounds	
RNG entropy	N/A	Random value
Input file (1-4)	UPRN; Connectivity; MPANs; Electric Office	Electric Office; Model file; Model parameters

Table 11: Model log details

4.5. Developer options

There are a series of hidden sheets in the UI workbook which are not intended to be accessed by users of the model but can be amended for development purposes. The following table outlines the purpose of the sheets and how they can be used for future development:

Sheet	Description
Model 1	This sheet is used to store information on the profile settings in Model 1 and referenced when a run is recorded in the log. This is automatically updated when changes are made in the UI.
Model 2	As above for Model 2.
xlwings.conf	This sheet contains the Python setup information defined during configuration process in the UI. The format and variables are structured to be read by the xlwings VBA module.
About	This sheet contains the information presented in the 'About' user form. This can be edited to change the information displayed in the UI.

Table 12: Summary of UI hidden worksheets

Appendix 1: Python setup instructions

USER SETUP

Install environment from snapshot (e.g. for non-networked PC)

See https://conda.github.io/conda-pack/#commandline-usage.

Note that conda must be installed first.

1. Run `conda info` and find the "envs directories" settings. Choose one of these as the location for the environment. For example: `C:\Users\<username>\.conda\envs`.
2. Unzip shapshot archive into a new directory in one of the env directories; you may need to create the envs directory. For example, if the envs directory is C:\Users\<username>\.conda\envs, then the contents of the archive should be extracted into C:\Users\<username>\.conda\envs\<name>, where "`<name>`" will be the name of the conda environment below.
3. Open a command prompt
4. Run `conda env` list and should see the name and location of the environment you extracted
5. Run `conda` activate `<name>`
6. Run `conda-unpack`
7. Close and reopen the command prompt
8. Run `conda` activate <name>
9. Run `xlwings addin` install to install xlwings addin, if it's not already installed
10. Run python `-m wpd_seam.xlwings.settings` to get xlwings settings

Updating the project code package

Once a working conda environment is obtained, the project code can be updated using the Python wheel provided without creating a whole new environment. This can be done offline, provided there are no new dependencies.

1. Open a command prompt
2. Run `conda` activate `<name>`
3. Run python `-m pip install wpd_seam-<version>.whl`

Setup User Interface for initial model run

The zip file contains the User Interface and input files needed to run the models.

Extract UI and input files:

1. Unzip the folder into its own directory. This does not have to be a specific location but will be where the input files will need to be stored and referenced from the UI.
2. The UI ('Spatially Enabled Asset Management User Tool v1 0.xlsm') is in the first level of the directory. You can open and run the tool from here or move it to the location you want to access the UI.

UI Python setup

Click the 'Python Setup' button on the home user form. The required settings are given by the "xlwings settings" step above. Alternatively, you should set the conda path to the location where the conda distribution is installed and set the conda env to the name of the conda environment above. Once complete press 'Save'.

TROUBLESHOOTING

Cannot find conda command

This means that the conda executable is not on your path.

There are three alternatives:

1. Run `conda init` in a command prompt to update the configuration of your command prompt (this may require admin access). See `conda init --help` for details.
2. Your conda distribution may include a pre-configured command prompt, which can be used in place of the "Command Prompt" above. Check the Start Menu for "Anaconda Prompt", "Miniconda Prompt", "Miniforge Prompt" or similar.
3. Replace references to conda above with the full path to the conda.bat file, until an environment is activated. On Windows, this is usually `<conda install path>\condabin\conda.bat`, where "`<conda install path>`" is the directory where the conda distribution is installed. For example, on some systems, this is `C:\ProgramData\Anaconda3\condabin\conda.bat`.

Appendix 2: Model 2 parameters file

The contents of the default model parameters file for model 2 are as follows. This file defines various parameters used by the spatial model, especially the levels for the categorical variables and parameters relating to the structure and training of the neural network.

spatial_model.json


```
{
  "categories": {
    "network_type": [
      "LV",
      "MV",
      "HV"
    ],
    "nominal_voltage_pp": [
      "110",
      "230",
      "400",
      "11000",
      "33000",
      "132000"
    ],
    "spec_material": [
      "aaac",
      "abc",
      "acsr",
      "al",
      "c/c",
      "cad cu",
      "consac",
      "cu",
      "hdc",
      "hyb",
      "s/c",
      "sac",
      "solidal",
      "wcon"
    ],
    "spec_size": [
      0,
      10,
      20,
      30,
      60,
      90,
      140,
      280,
      550,
      900
    ]
  },
  "training_kwargs": {
    "f_val": 0.2,
```

```
    "f_test": 0.3
  },
  "model_kwargs": {
    "n_hidden": 12,
    "conv_layers": 4
  },
  "optimizer_kwargs": {
    "lr": 0.01,
    "weight_decay": 0.0005
  },
  "num_epochs": 200,
  "thresholds_kwargs": {
    "quantiles": {
      "low": 0.2,
      "medium": 0.5,
      "high": 0.8
    },
    "include_missing": false
  }
}
```


Table 13: Default model parameters file for model 2

Glossary

Abbreviation	Term
Artificial Intelligence (AI)	The training of computer systems with human intelligence traits like learning, problem solving, and decision making.
Business-as-Usual (BaU)	The normal execution of standard functional operations within an organisation.
Command Line Interface (CLI)	An alternative to the User Interface as a mechanism to run the models, particularly suitable for repetitive running of the model via a batch file or similar.
Comma-separated values (CSV)	An open tabular data interchange format
Computer-Aided Drawing (CAD)	Creation of computer models defined by geometrical parameters
CROWN	WPD enterprise asset management system. Holds data about assets which includes data defining the assets, condition data and defect data. It also records inspection and maintenance activities on the assets as 'events'.
Data Cleanse	The action of identifying and then removing or amending any data within a database that is incorrect or incomplete.
Electric Office (EO)	WPD's geospatial system which displays the network layout at all voltages
Geospatial Information System (GIS)	A data system capable of capturing, storing, analysing, and displaying geographically referenced information.
OGC GeoPackage (GPKG)	An open geospatial database format using SQLite
Graph Convolution Network (GCN)	Neural network architecture for machine learning on graphs.
Graph Neural Network (GNN)	A class of machine learning / deep learning methods designed to perform inference on data described by graphs.
Integrated Network Model (INM)	WPD's combined dataset for 11kV and above that merges data from CROWN, GIS and PowerOn.
JavaScript Object Notation (JSON)	An open data interchange format.
Linestring	A data structure for representing lines (curved and straight) within the well-known text markup language for processing vector geometry objects.
Machine Learning (ML)	A subset of AI, the study and application of algorithms that improve automatically through experience.
Meter Point Administration Number (MPAN)	A unique 21-digit reference number used in the UK that identifies each electricity supply point.
Planarise	To create multiple line features by splitting the feature where they intersect.
PowerOn	WPD's distribution management system used for system operations.
Proof of concept (PoC)	An exercise or demonstration to verify that concepts or theories have the potential for real-world application.
Python	An open-source general-purpose programming language.
QGIS	A free and open-source cross-platform desktop geographic information system (GIS) application that supports viewing, editing, and analysis of geospatial data
Real World Object (RWO)	Unique identifier of an object in Electric Office.

Relational Graph Convolution Network (R-GCN)	Heterogenous version of GCN
SQLite	An open SQL database engine and file format
Unique Property Reference Number (UPRN)	A unique number (1-12 digits in length) created by the Ordnance Survey for every addressable location in the UK.
User Interface (UI)	The means by which the user will interact with the model.
Well-known text (WKT)	A text markup language for representing vector geometry objects.

WPD INNOVATION

Transforming the electricity network

Western Power Distribution (East Midlands) plc, No2366923
Western Power Distribution (West Midlands) plc, No3600574
Western Power Distribution (South West) plc, No2366894
Western Power Distribution (South Wales) plc, No2366985

Registered in England and Wales
Registered Office: Avonbank, Feeder Road, Bristol BS2 0TB

wpdinnovation@westernpower.co.uk
www.westernpower.co.uk/innovation