BRIEFING B2 UNCLASSIFIED

Software Pipeline of SOL Aquifer

How SOL Aquifer turns codified physics into training data and instant predictions, demonstrated on the HVAC Manual J load calculation

SOL Engineering Services

PUBLISHED 2026

READING TIME 7 min

Overview

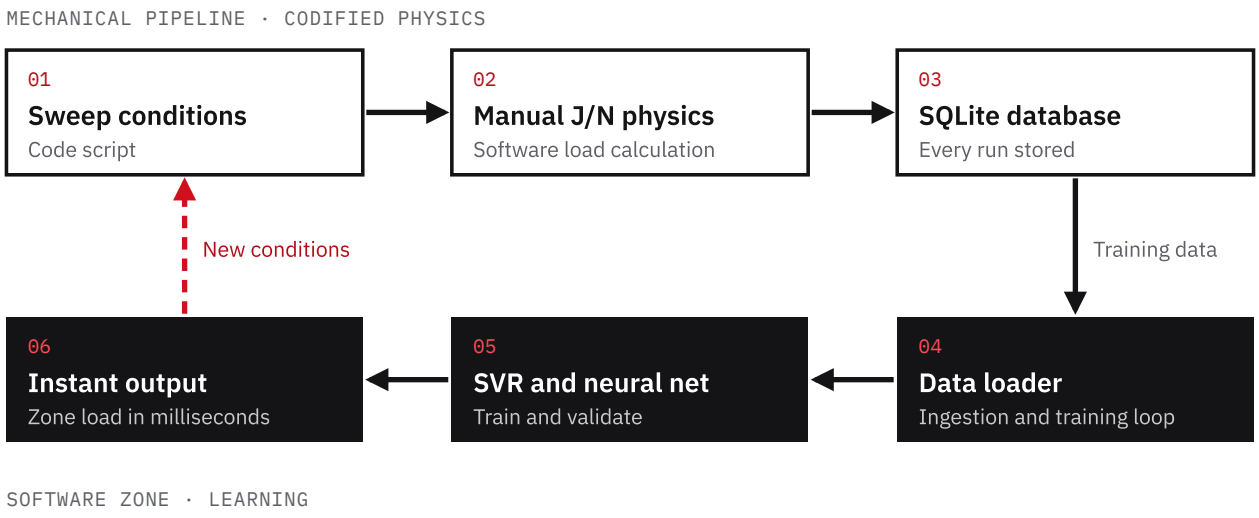
SOL Aquifer™ is the software pipeline at the center of SOL's AI framework. It turns codified physics into training data, trains machine-learning models on that data, and serves the trained models as instant predictors. The same pipeline is meant to design the CF-JET cold plate; before that, SOL proved it end to end on a problem whose physics is well documented: the heating, ventilation and air-conditioning (HVAC) cooling load of a building zone, calculated with the industry-standard Manual J procedure.

The HVAC run is the "mechanical pipeline" half of SOL's Code-to-Architecture loop made concrete. Codified physics and scripted sweeps are the optional target: a code script sweeps the conditions, a physics engine calculates the result for each, and the learning half of the pipeline replaces days of manual calculation with a model that answers in real time. This page walks through each stage, the tools behind it, and what the run shows.

1. The Pipeline at a Glance

Figure 1 shows the loop. The top row is the mechanical pipeline, the codified physics; the bottom row is the software zone, where the models learn. Each step in the bottom row is the software equivalent of a manual engineering task: the sweep script replaces choosing cases by hand, the physics engine replaces the hand calculation, and the trained model replaces repeating that calculation for every new question.

Figure 1. The SOL Aquifer loop from scripted sweep to instant prediction



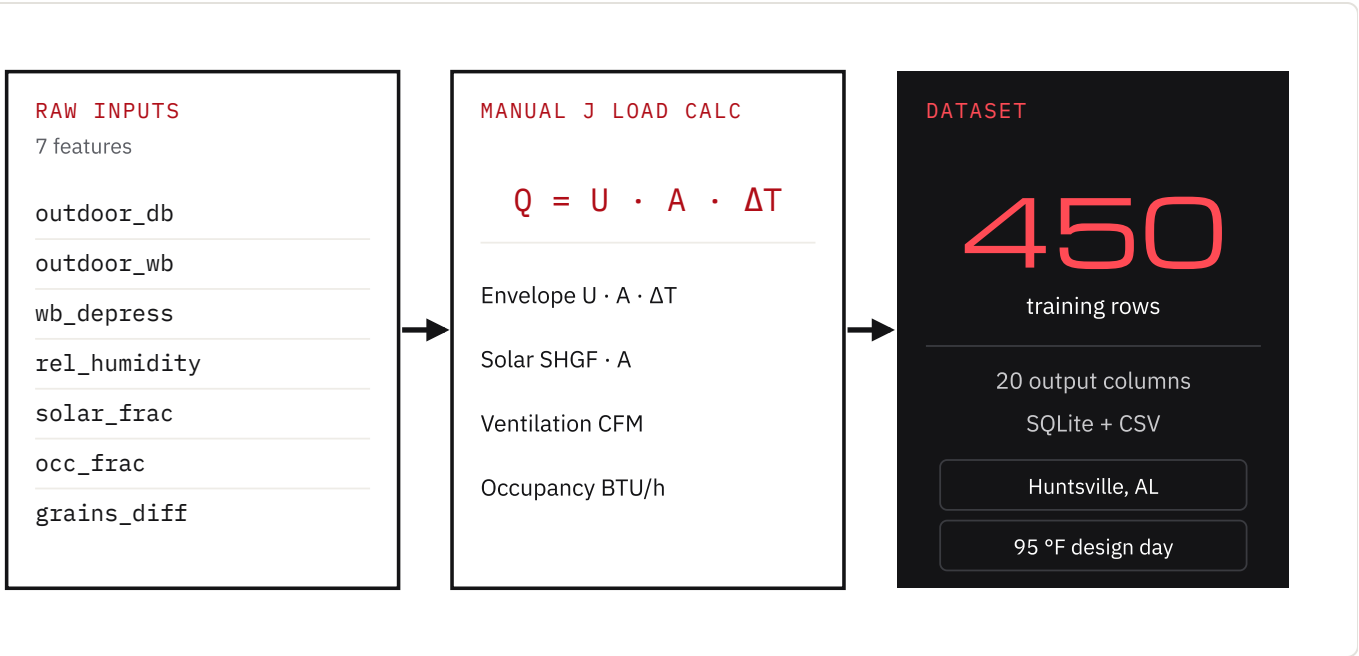
Schematic. Top row: physics that is written down and run as code. Bottom row: the models that learn from it.

01	**Sweep conditions.** A code script generates the combinations of operating conditions to study, the software equivalent of choosing test cases by hand.
02	**Manual J/N physics.** A software implementation of the ACCA Manual J (residential) and Manual N (commercial) load procedures calculates the cooling load for each combination.
03	**SQLite database.** Every result is stored in a lightweight database, organized for efficient retrieval and kept for the life of the project.
04	**Data loader and training loop.** Rows are pulled from the database, prepared and fed to the models in batches.
05	**SVR and neural network.** Two kinds of model learn the relationship between conditions and load, and are validated against each other.
06	**Instant output.** The trained model returns the zone load for new conditions in real time; questions it cannot answer confidently become new sweep conditions.

2. Generating the Training Data

The pipeline creates its own training data by systematically varying temperature, humidity, occupancy and solar inputs. Each scenario is processed with the Manual J/N calculations to build a comprehensive, labeled dataset that captures the full range of zone load behavior (Figure 2). An automated environmental sweep script populates the SQLite database continuously, so the structured, data-driven pipeline replaces manual calculation workflows entirely.

Figure 2. Seven inputs, one physics calculation, 450 labeled rows



Schematic of the demonstration run.

The seven input features describe the outdoor air, the sun and the people in the zone:

- **outdoor_db** and **outdoor_wb**: outdoor dry-bulb and wet-bulb temperatures, the two standard measures of air temperature and moisture.
- **wb_depress**: wet-bulb depression, the difference between the two, which indicates how dry the air is.
- **rel_humidity**: relative humidity.
- **solar_frac**: the fraction of peak solar gain at that moment.
- **occ_frac**: the fraction of full occupancy.
- **grains_diff**: the difference in moisture content between outdoor and indoor air, in grains of water per pound of dry air, which drives the latent (dehumidification) load.

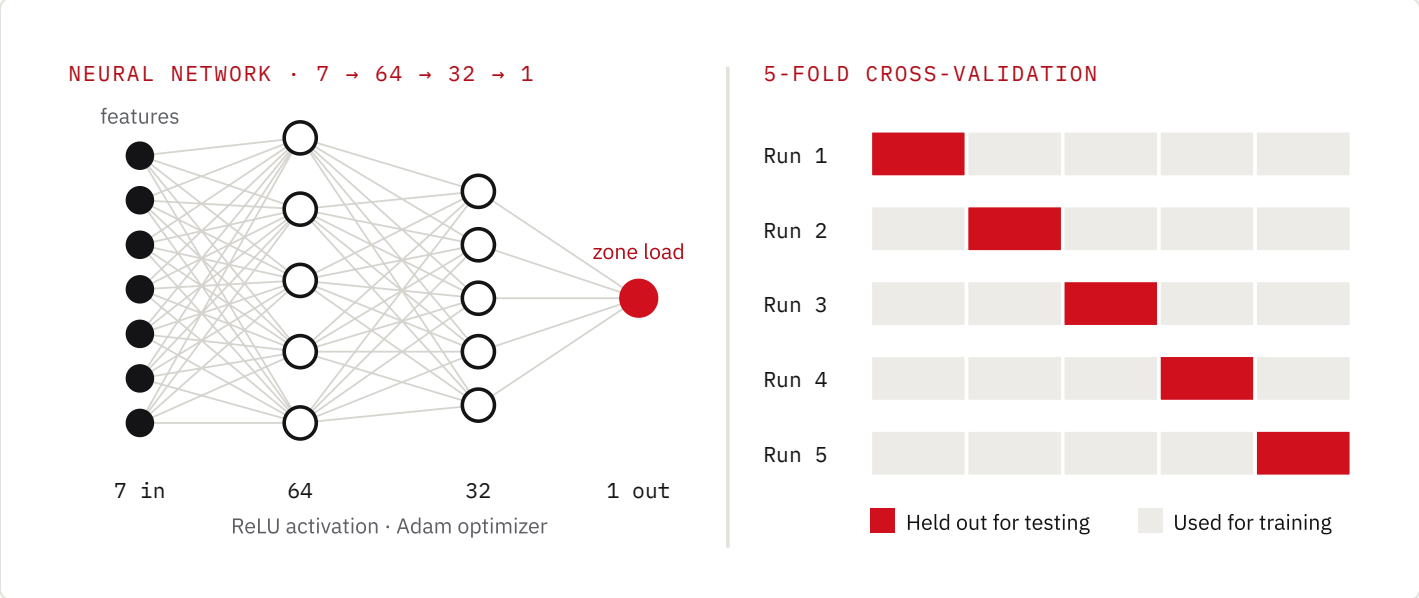
Manual J adds up the zone's cooling load from its parts: heat through the envelope ($U \cdot A \cdot \Delta T$, the heat-transfer coefficient times area times temperature difference), solar gain through windows (solar heat gain factor times area), outdoor air brought in for ventilation (by airflow in CFM), and the heat people give off (in BTU per hour). The demonstration swept these conditions for Huntsville, Alabama, on a 95 °F design day and stored 450 rows with 20 output columns, in SQLite and as CSV.

3. Training and Validating the Models

Support vector regression (SVR) and neural network models are trained on the stored simulation data to learn the complex relationships in HVAC load. The data moves from the database to the models through an extract, transform and load (ELT) process, or a PyTorch DataLoader that batches and shuffles rows from the dataset for each training epoch:

01	**Ingestion.** Pull rows from SQLite into a DataFrame.
02	**Preprocessing.** Scale and normalize the features (StandardScaler) so that no input dominates because of its units.
03	**Train/test split.** Divide the rows into five folds for cross-validation.
04	**Training loop.** (1) Forward pass, (2) compute the loss as mean squared error (MSE), (3) backpropagate, (4) update the weights with the Adam optimizer.
05	**Evaluation.** Root-mean-square error (RMSE) on the held-out fold.

Figure 3. The neural network and five-fold cross-validation



Schematic. The input layer shows all 7 features; the hidden layers show a sample of their 64 and 32 neurons.

The two models take different approaches:

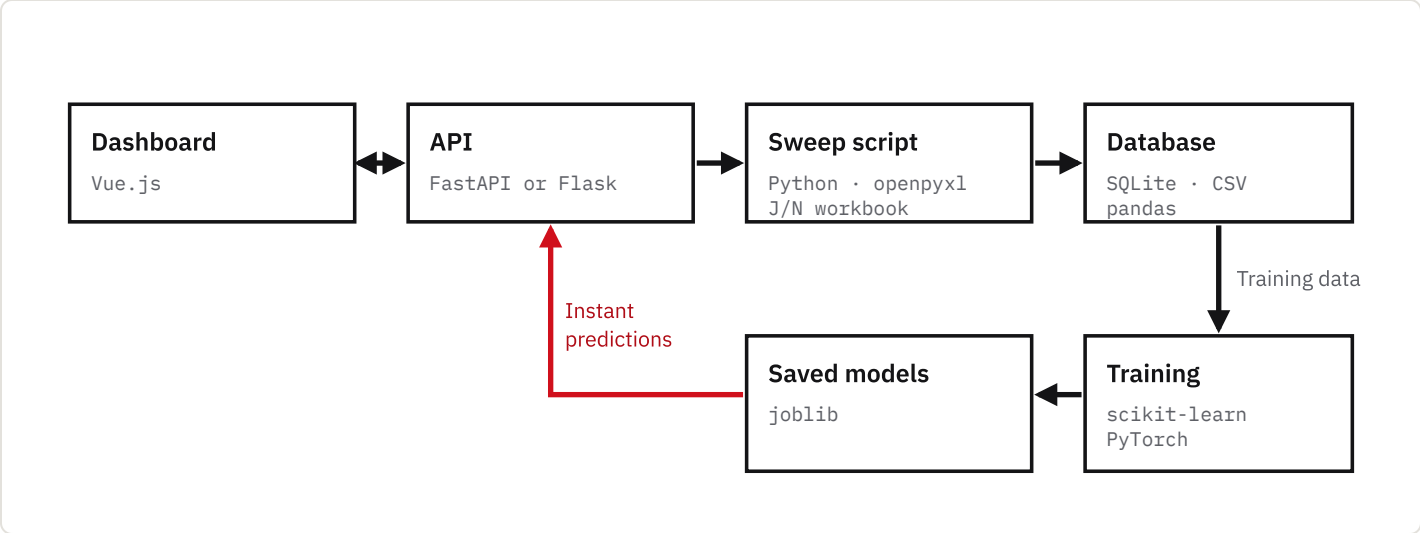
- **Support vector regression** uses a radial basis function (RBF) kernel with three settings: C, the penalty for errors; ε, the margin within which errors are ignored; and γ, how far each training point's influence reaches. SVR works well on small, smooth datasets like this one, but its training time grows roughly with the square of the number of rows, $O(N^2)$, so it suits hundreds or thousands of rows rather than millions.
- **The neural network** takes the 7 inputs through two hidden layers of 64 and 32 neurons to a single output, the zone load. Each neuron uses a rectified linear unit (ReLU) activation, and the Adam optimizer adjusts the weights. It scales to much larger datasets than SVR.

Five-fold cross-validation (Figure 3) trains each model five times, each time holding out a different fifth of the data for testing, and compares the RMSE across all five runs. Every row is tested exactly once, and the comparison shows which model generalizes better rather than which one happened to suit a single split. Once trained, the models deliver instantaneous zone load predictions, replacing days of manual computation with real-time, accurate forecasting for engineering design decisions.

4. Data Pipeline and Tech Stack

The pipeline runs on widely used, open tools, shown in Figure 4 and listed by stage in Table 1. A web dashboard asks the API for predictions; the API runs sweeps that fill the database, the training stage builds models from it, and the saved models answer the dashboard instantly.

Figure 4. How the parts connect



Schematic. Black arrows: data and training. Red arrow: predictions from the saved models.

Table 1. Data pipeline and tech stack

Stage	Role	Tool	What it does
1. Environmental sweep: Manual J/N physics, synthetic scenario generator	Data foundation; framework to sweep conditions across many combinations	FastAPI or Flask	Back-end API; provides the Vue.js endpoints and makes the sweep calls
		openpyxl	Reads and writes the Manual J/N Excel workbook
2. SQLite database	Stores all data from training and testing runs; serverless architecture	pandas	Loads, cleans, filters and sorts the data
		Node.js	JavaScript runtime that handles application logic
		CSV	Comma-separated values: stores tabular data in a simple, structured format
		sqlite3	SQLite support built into Python
3. Neural network and support vector regression	Training loop; extract, transform and load	MLPRegressor	Feed-forward multi-layer perceptron (MLP) neural network in scikit-learn
		joblib	Saves trained models and loads them back
		scikit-learn	Python library that trains and runs the models: neural networks, SVMs, StandardScaler and more
		pandas	Runs grouped operations on large datasets
		ELT, PyTorch DataLoader	Ingestion, training loop and evaluation
		ReLU	Rectified linear unit, the activation function for the network's neurons

Stage	Role	Tool	What it does
		Adam	Adaptive moment estimation, the optimization algorithm for the neural network
		RBF kernel, C, ε	Settings of the support vector regression (SVR) model
		GridSearchCV	scikit-learn tool that searches for the best hyperparameters, used here for SVR
4. Dashboard	Queries the model for instantaneous predictions	Vue.js	JavaScript framework for the user interface, with reactive data binding

Tools as listed for the demonstration run.

5. From HVAC to CF-JET

The HVAC run validates the pipeline itself: data generation, storage, training, cross-validation and instant prediction all work end to end on physics that engineers already trust. The same structure carries over to the cold plate. Correlations and simulations for jet impingement replace the Manual J calculation; the sweep varies design variables such as jet diameter, jet pitch and flow rate instead of weather; and the trained surrogate predicts thermal resistance and pressure drop in milliseconds instead of zone load.

Two things change with the harder problem. Physics simulations of a cold plate cost far more than a Manual J calculation, so the pipeline has to learn from fewer, more expensive runs; and a design decision needs to know how far to trust each prediction. Those are the subjects of the companion research papers Engineering Design Under Sparse Data, Surrogate Modeling, Multi-Fidelity Learning, and Optimization for Thermal-Fluid Design and Uncertainty-Aware Neuro-Symbolic AI for Engineering Decisions, and of the Forest-to-Desert Framework, which this pipeline feeds.

6. Key Takeaways

- SOL Aquifer turns codified physics into labeled training data automatically, so models are trained on clean, consistent data from the start.

- The HVAC demonstration ran the whole loop: 7 inputs, Manual J/N physics, 450 stored rows, SVR and neural network models, five-fold cross-validation and instant predictions.
- The pipeline is built on open, widely used tools: Python, scikit-learn, PyTorch, SQLite, FastAPI or Flask, and Vue.js.
- The same stages apply to the CF-JET cold plate, where the open questions are learning from sparse, expensive data and knowing how far to trust each prediction.