Building towards an MVP

Our goal for the MVP is the port a simplified version of the training pipeline built by Marat to the Databricks ecosystem. This excludes certain capabilities that are unnecessary for an MVP, and overcomes certain limitations that were prohibitive for an MVP. This document describes the progression of steps that were taken to build the MVP. In general, these iterations can be considered additive, so a model change described for a given iteration will be included in all subsequent iterations.

Scope:

Features:

- Building Features: All features that are plausibly related to any end use
- Weather Features: Supports all 7 hourly weather features contained in ResStock, but only uses `temp_air`, `wind_speed`, `ghi` as well as an additional `weekend` flag.

Targets: `electricity`, `methane gas`, `fuel oil`, `propane` aggregated over all end uses

Target Timescale: Annual

Upgrades: 0 (baseline), 1 (basic enclosure), 3 (min hp), 4 (max hp), 6 (hpwh), 9 (1+4+6+induction range+hp dryer), 11.05 (medium hp), 13.01 (medium hp + basic enclosure)

Universe: Occupied homes without shared HVAC or water heating systems, including single family homes (attached and detached), mobile homes, and multi-family homes with <5 units.

Iterations

This describes the steps that were taken to build the MVP.

Notes on reported metrics: 70/20/10 train/val/test split with metrics reported on the test set. However for bucketed comparison, we are reporting training error because this paradigm doesn't naturally apply to the bucketed model. Also, for the larger models, we report the test metrics on a 100K sample of the test set to limit computational expense.

0. Non-DataBricks initial model

https://github.com/rewiringamerica/surrogate_modeling/pull/1

https://github.com/rewiringamerica/surrogate_modeling/pull/2

Best model/experiment: Not stored.

Training time: ~1h to train on 9K training samples using 1 GPU/85GB memory. An additional ~3h to build and cache features.

Summary: Marat (google fellow) implemented a first version of the surrogate model without using databricks.

Features:

- Building Features: 15 features relating to HVAC, most concentrated on insulation
- Weather Features: Supports all 7 hourly weather features contained in ResStock, but only uses `temp_air`, `wind_speed`, `ghi` .

Targets: `heating` and `cooling` aggregated over all fuel types

Target Timescale: theoretically any

Upgrades: 0, 1, 3, 4, 5

Universe: Non-vacant single-family attached homes, <8000 sqft, ≤ 10 occupants. Trained only on ~5% of the available data (10K homes) due to I/O bottlenecks

Architecture: The general flavor of the model architecture is described in this page which details the architecture for MVP. The original architecture deviated in two ways: only two dense layers in the building metadata branch; the combine trunk consisted of 4 dense layers with 16 neurons each, ending in a final output layer of size 2.

Parameters:

- *Batch size*: 6
- *Epochs*: 100, with patience is set to 5 tracking training loss.
- *Initialization*: All trainable layers use the default Xavier intialization.
- *Activation*: All trainable layers use a leaky ReLU activation function.
- *Optimizer*: Adam
- *Loss*: Mean absolute error

All other parameters use keras defaults.

Results: ~13% and 17% Median APE (ish?) for heating and cooling respectively over baseline and 4 upgrades. Metrics were not evaluated on savings and were not stored anywhere.

---

1. Initial training pipeline with feature store/mlflow

https://github.com/rewiringamerica/surrogate_modeling/pull/5

Best model/experiment: sf_detatched_hvac_baseline@v1

Training time: 3.5h on ~220K training samples using CPU/85 GB memory. Time to build features is negligible, so not reported from this point forward.

Summary: The first step was to simply port over a simplified version of the training pipeline built by Marat to Databricks, in particular, integrating with the feature store and mlflow. This first version predicts HVAC baseline consumption (upgrade = 0 only)by end use (`heating`, `cooling`) at an annual timescale, and is trained on all buildings in the same universe.

Features: The building metadata feature set was expanded from the 15 implemented and used by Marat to include all 32 features that are plausibly related to HVAC energy consumption in a SF detached home.

Architecture: Since we added more features, many of which are one-hot encoded, we added 2 larger initial dense layers to the building metadata branch to accommodate the higher dimensionality of the data.

Parameters: We also changed the patience to track the validation loss, since this is a more appropriate choice for preventing overfitting.

Implementation details: To implement this, we follow this example for building a feature store, defining a custom mlflow PyFunc model, and logging and scoring a model. Full implementation details are out of scope for this document, but the stack for tracking, logging, and registering the models is tensorflow > keras > mlflow (pyfunc) > Databricks FeatureStoreClient, where each is a wrapper around the previous class.

Two important implementation details that mitigated some I/O bottlnecks:

1. Storing each weather feature as an 8670-length spark array type
2. Bring the building metadata features and targets (indexed by building, upgrade) and weather features (indexed by weather station) into memory separately, and only join these at run time when generating the data for the batch. Note that this solution will not scale when if we move to hourly prediction.

Inference Post-Processing*:* Since the final layer is a leaky ReLU meaning that the model can predict negative values, so predictions are clipped at 0 in post-processing.

Results: 10.5%, 13.7%, and 8.2% Median Absolute Percentage Error (APE) for cooling, heating, and combined HVAC (heating + cooling) respectively. This is a substantial improvement over out current bucketed method which is 31.8% and 38% Median APE for cooling and heating respectively. We also compare against the performance of a simple feed forward network, which generally performs much worse across the board, particularly when looking at mean error. The feed forward network does have outperform on predicting Room AC cooling, which the CNN seems to be struggling with. Based on these results, we elect to not continue supporting or benchmarking against the feedforward network from this point forward. Complete metrics with heating and cooling error are broken down by heating and cooling type respectively.

2. Adding HVAC upgrades and evaluating savings predictions

https://github.com/rewiringamerica/surrogate_modeling/pull/6

Best model/experiment: sf_detatched_hvac@v5

Training time: 4h on ~800K training samples using CPU/170 GB memory

Summary:

Upgrades: We next added 3 HVAC upgrades 1 (basic insulation), 3 (min efficiency HP), 4 (max efficiency HP), effectively quadrupling the size of the training set. These were added by copying the baseline features, and updating a handful of them according to the upgrade description in the EUSS documentation, deferring to the https://github.com/NREL/resstock/blob/run/euss/EUSS-project-file_2018_550k.yml when the docs were ambiguous.

Results: 14.1%, 12.1%, and 9.1% Median Absolute Percentage Error (APE) on HVAC savings for upgrades 1, 3, and 4 respectively, and 7.9% median APE on baseline HVAC consumption. These results somewhat alleviate the concern that subtracting two predictions to calculate the final quantity of interest would lead to compounding errors. The median errors only slightly higher than baseline, although the mean APE is close to ~45% for upgrades 1 and 3 which is quite high but still a marked improvement over bucketed, which is ~200% mean APE for these upgrades. While the bucketed model does actually outperform the CNN for homes without baseline heating in terms of APE on upgrades 0 and 1, the absolute error is still quite small. The model seems to be struggling Upgrade 1 (basic weatherization), with the model performing about 5 percentage points worse at predicting savings then baseline, but again, by comparing to the absolute savings, it seems likely mostly a reflection of the smaller absolute savings with this upgrade, although this could also be due to the weather embeddings not being large enough or the apply logic not fully capturing the changes to the energy plus inputs.

Note that the evaluation procedure was updated to include savings error, which is calculated by subtracting the building's baseline prediction from the prediction for the given upgrade. The evaluation was also modified to calculate the error over total HVAC consumption (summing over heating and cooling), which differs from the previous evaluation which broke reported heating and cooling error separately. This latter change was enacted in order to make these results more directly comparable to iterations ≥4, at which point heating and cooling will not be predicted separately.

2A. Expanding to single-family attached homes

https://github.com/rewiringamerica/surrogate_modeling/pull/6

Best model/experiment: sf_hvac@v2

Training time: 7.5h on ~900K training samples using CPU/75 GB memory

Summary:

Universe: Universe is expanded to include attached homes, large homes (>8000 sqft, 4.1% of SF BEMs), and large household sizes (>10 occupants, 0.1% of SF BEMs), which were previously excluded due to concerns about training on outliers.

Features: Three more building features relevant for attached homes are added (attached flag, number of units, middle unit flag). Additionally, an omitted feature was added which indicates whether heating is ducted— prior to this we only had a flag for whether or not the unit had ducts, but often units with ducts still have ductless heating. This brings the total number of features to 36. Note that shared cooling and heating are both represented as their own categories of heating and cooling appliances.

Results: Not a huge difference from before, particularly to the median scores. The mean scores changed in both directions for various upgrades/fuel types, but no clear signal that adding either attached homes or outlier homes in terms of size poses a problem for training. The model doesn't appear to perform worse on homes with shared heating or cooling.

By breaking out the metric by attached and detached homes (shown across all non-baseline upgrades), we see that the model is performing slightly worse on attached homes, and interestingly, it is performing worst on predicting savings on attached homes without shared heating/cooling. It does bear mentioning that the number of building models with shared heating/cooling in the test set is quite small (~200).

Is attached	Has shared HVAC	Median APE - Savings	Mean APE - Savings	Median Abs Error - Savings	Mean Abs Error - Savings	Median APE	Mean APE
True	False	15.1	96.3	549	1065	9.3	24.6
True	True	12.2	39.7	391	1368	9.7	13.9
False	False	11.6	33.9	857	1500	8	12.1

3. Predicting by fuel type

https://github.com/rewiringamerica/surrogate_modeling/pull/7

Best model/experiment: sf_hvac_by_fuel@09b4b724a85c4aaa969447cffd884bda

Training time: 4h on ~900K training samples using 1 GPU/85 GB memory. We realized that we were able to train the model with the GPU, and then run inference using a CPU cluster on a GPU trained model. From this point onward, all models are trained with a GPU, which speeds up training by >2x (~180s/epoch vs ~500s/epoch).

Summary:

Targets: We shifted our model's focus from predicting HVAC consumption by end-use (`heating`, `cooling`) to predicting it by fuel type (`electricity`, `methane gas`, `fuel oil`, `propane`). These new targets introduce a significant challenge: propane and fuel oil, which are not used in most households, have zero consumption values in the vast majority of samples. This results in a sparse dataset where zero values dominate, complicating the training of our deep learning model. The **purple** loss curve depicted below highlights this challenge, showing how training can stagnate with a high loss due to inadequate training signals from these zero-value targets. The **blue** loss curve shows a particular lucky run in which there was a training break through, causing the loss to drop precipitously from a near constant value for first propane and then fuel oil. In this case, the model was able to find a path through a more favorable region of the loss landscape, allowing it to escape local minima where the gradient information was insufficient, but such outcomes were rare, and generally, the model just ceased to learn as it failed to receive sufficient gradient feedback. Overall, we need a more robust and dependable training strategy as this sparsity issue is expected to worsen as we incorporate more electric and extends to cover all end uses, both of which will be even more likely to have zero delivered fuel consumption.

Initial unstable and stagnated training (blue and purple loss) due to sparsity of fuel oil and propane targets. Solved using custom masked MAE loss function (red and orange).

Parameters & Architecture: Tried various parameter and architecture changes to solve the sparsity issue ✅ := included in final model ❌ := not included in final model

- *Leaky ReLU Activation*: *Already implemented* ✅ The most obvious solution to mitigating the dying ReLU problem, where this activation function allows a small, positive gradient when the unit is inactive, theoretically maintaining some gradient flow. However, this activation was already being used, offering no further improvement for the current model iteration.
- *Linear Activation in Final Layer: Worse performance* ❌ Introducing a linear activation in the final layer was intended to help help the gradient flow, in particular, preventing vanishing gradients. However, this change exacerbated the learning stagnation, presumably due to the model's inability to restrict output to smaller negative values, which is problematic given that our targets are non-negative.
- *He Initialization*: *Slight improvement* ✅ Specifically designed for layers using ReLU activation functions, He initialization sets the initial weights to values that support maintaining equal variance in activations throughout the network. Note that we were previously using the default Xavier initialization, which has this property when using a linear activation. This new initialization yielded a very slight improvement by facilitating better gradient propagation at the start of training, and we decided to retain even given the very modest gains it since it is the more statistically appropriate choice for a a layer with a non-linear activation like ReLU.
- *Increasing Batch Size*: *Slight improvement* ✅. We increased the batch size from 64 to 256 to increase the probability of including non-zero targets within each batch. This proved moderately effective at providing a more representative gradient estimate during training, thereby improving the model's ability to learn from a more diverse set of data points.
- *Custom Loss Function - Zero-Masked MAE*: *Substantial improvement* ✅ We implemented a masked mean absolute error (MAE) where predictions for the given fuel type are set to zero if the true target is 0. This means that the model gets no training signal unless the fuel is actually present for the given sample. The idea is that a similar transformation will be applied in post-processing, where we simply set a fuel consumption to 0 if the home has no appliance with that given fuel. By incorporating this into the loss, we prevent the model from wasting computational resources on learning patterns that are known a priori and are therefore predictively irrelevant. Overall, the custom loss led to a substantial improvement in model stability and learning, as evidenced by the improved blue loss curve. Note that the increased batch size was likely instrumental in the success of this new loss, without which there would have likely been no training signal in many batches since the summed loss would be 0. The **red** loss curve above shows our shows a training run using the custom loss function along with He initialization and increase batch size. We may refine this custom loss function further in future iterations by modifying the masking mechanism (e.g, only include in loss if non-zero) or applying adaptive sampling methods to balance the representation of different fuels within training batches, particularly if prediction on delivered fuels deteriorates as we add more fully electric samples.

- *Batch Normalization*: *Worse performance* ❌ While batch normalization is generally beneficial for stabilizing neural network training by normalizing layer inputs and reducing the probability of vanishing gradients, it introduced significant noise into our model's training process. This noise overwhelmed the benefits, complicating the learning process rather than facilitating it. The **orange** curve above exhibiting a solid training loss and noisy validation loss is from a training run where batch norm layers were added in addition to the custom loss function, He initialization, and increased batch size. We may also revisit batchnorm in future iterations, paired with an increased patience to prevent the training from quitting early due to the noisy val loss.
- *Increased Patience: Moderate Improvement* ✅ We increasing the patience to 10 epochs since the training was a little bit more noisy on the new prediction task, and training was quitting before the electricity loss had time to flatten out (see **red** training loss from run with lower patience)

Implementation details: Unfortunately, actually implementing the most promising of the solutions unleashed a giant can of worms described here for a future data engineer to make sense of Issue: Logging Model with Custom Loss. TL;DR: we cannot log a model with a custom loss function using the built in DataBricks FeatureEngineeringClient log_model wrapper function, which means we have to unwrap one layer of our giant layer cake of dependencies here, and use MLflow (upgraded to v2.13.0 to allow for caching the script containing the custom loss fn) for logging instead. Unfortunately, this results in us no longer being able to actually register the model for serving, since we are required to log a schema signature to register a model to the unity catalog with MLflow, which inexplicably slows down inference > 100x. It also prevents us from using some of the the more convenient integrations with the feature tables, such as automatically pulling features based on keys at inference time. For now, we have decided to just skip model registration, and just call the model based on the run ID.

Inference Post-Processing*:* Values are also set to 0 in post processing if the given fuel target is not used by any appliance in the modeled building.

Results: Summed over all fuels, we get 14.2%, 11.9%, and 8.6% Median Absolute Percentage Error (APE) on HVAC savings for upgrades 1, 3, and 4 respectively, and 6.5% median APE on baseline HVAC consumption. The medians are actually lower than before, although mean MAE generally went up across all upgrades. By summing over the fuels, we are not fully capturing the error since errors in different directions on different fuels will get smoothed out, but this likely isn't that huge of an issue since most homes will not have more than 2 non-zero fuels. Looking at results by fuel type, we see that the median APE is 11% for fuel oil, and ~15% for all other fuels. The MAPE for electricity is quite a bit higher than the rest (79.4) while the mean absolute error is much lower (387kWh) because a lot of households have only trace electricity used by appliances. It is worth mentioning that while we set non-existent fuels to 0 in post-processing, for the purposes of evaluation, we set these to null so that our evaluation errors are not deflated by these zeros.

4. Predicting total consumption

Best model/experiment: mvp@d1ab327ba46644a3a917484b13d10dc5

Training time: 10.6h on ~1.3M training samples using 1 GPU/85 GB memory.

Summary:

Features:

Added 35 features relating to all end uses described in . This brings the total to 71 building metadata features, including 16 categorical features, resulting i an input vector of dimension 135 to the building metadata branch.

A few other small changes were made to existing features:

- There were a few features (e.g, setpoint, vintage) where "centering" (e.g, subtracting the mean) was being applied in transformation. This was removed in favor of applying centering consistently on all features via an initial batchnorm layer in training. In the future we may want to add a normalization layer that is not batchnorm, which would be less computationally expensive, although more annoying to implement .
- Climate temperature zone (i.e 1-7) was converted from categorical to numeric in order to capture the inherent ordering of these zones.
- Previously roof insulation and ceiling insulation were grouped into one features since only one is ever non-null. However, this requires more careful treatment since there are other groups of insulation features with the same

phenomenon, and it is unclear how to then apply upgrade logic to these combined features. For now, we have split these back into a two features, and left this as future work .

Targets: We are still predicting by fuel type (`electricity` , `methane gas` , `fuel oil`), but expanding to predict total consumption rather than just HVAC consumption. We experimented with breaking out fuels by end use too with mixed results, but decided to push this to future work .

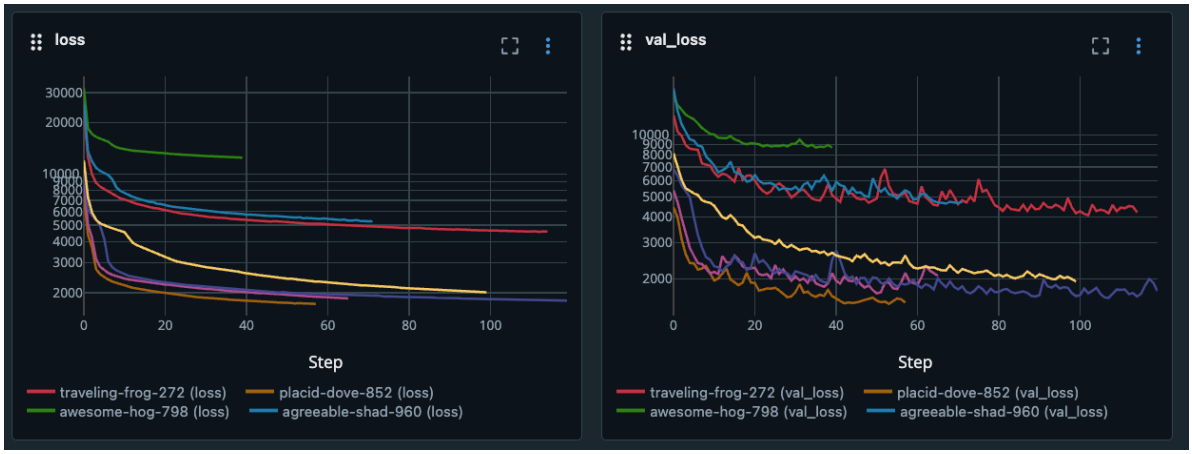
Upgrades:

We added HPWH upgrade (6) and whole whole upgrade (9), resulting in a total of 6 upgrade including baseline as described in , and a total of 1.9M (buidling, upgrade) samples. We also experimented with adding upgrades for heat pump dryer and induction alone outside of the whole home upgrade, but ultimately this was unsuccessful because the energy consumption savings are so small compared to the total energy consumption. The results were awful (~100% Median APE), and did not get anywhere close to beating the bucketing benchmark. This is partially because the benchmark is actually quite good here, since the variance in these end uses is so low (only like 10 possible values for range consumption). As mentioned above, we experimented with breaking out by end use to help ameliorate this issue, but ultimately decided this was out of scope for the MVP.

We also realized that the `applicability` columns in ResStock, which idenfies if the given upgrade was actually applied, can be used to verify whether our upgrade metadata apply logic is applied correctly. This led to finding a few small inconsistencies in the previous upgrade logic implementation, which is now corrected.

Universe: We removed the handful of homes that had other fuels for end uses other than heating. We also removed the handful of attached homes that have shared heating and cooling only heat pumps, since it s super unclear how to extract efficiency and how apply upgrade logic— note that these do not get upgraded in ResStock for some reason.

Parameters & Architecture: Tried various parameter and architecture to improve performance ✅ := included in final model ❌ := not included in final model



Training Run Progression: Yellow: first try predicting total consumption, without upgrades 6&9. Purple: adding batchnorm layers. Pink: adding upgrades 6 & 9. Brown: widening chokepoint. Blue: switching to masked MAE rather than zeroed out MAE, without widened chokepoint. Green: try adding dropout. Red (final): remove dropout and add widened chokepoint.

- *Increased Epochs & Patience: Moderate Improvement* ✅Increased to 200 and patience to 15. We added this because we did not see evidence of overfitting in the first 100 epochs, with the validation loss was still trending downward.
- *Batch Normalization*: *Substantial improvement* ✅ We added batchnorm layers to the two branches of the model, so after the dense layers in the building metadata branch and after the convolutional layers in the weather branch. We also added this directly after the initialization on both branches, thus performing the initial feature normalization described above, which should always be done even if we choose not to use batchnorm layers. We chose to apply batchnorm prior to activation—while there is some debate in the literature on whether it should be added before or after the activation, the original batchnorm paper applied this before the activation to reduce covariate shift, so this is still more the standard. We could experiment with putting this after activation in later

epics. Interestingly, adding layers in the combine trunk (not including the last layer) seemed to completely kill training, so we did not include them there. One hypothesis for why this is happening is that while batchnorm is fairly standard in classification tasks, where targets are invariant to arbitrary scaling and shifting, in regression tasks we are predicting absolute values, and thus applying this transformation closer to the targets may adversely effect prediction. See improvement in **purple** loss compared to **yellow.**

- *Custom Loss Function Masked MAE*: *Moderate improvement* ✅ We modified the masking mechanism of the custom loss function to actually mask out the (sample, fuel) predictions if the true value is 0, meaning that the mean is taken over a varying set of elements on each batch, which in effect allows the model to place more emphasis on optimizing predictions for delivered fuels when they are present. Previously, we simply set these to 0 which resulted in lower mean error in batches with a higher proportion of sparse targets. This new masking mechanism seemed to result in lower error for delivered fuel targets, and in general lower mean percentage errors, even when median percentage errors were similar. The **blue,** green, and red losses (with the latter two including additional param changes) are from runs using this new loss function— while this may look much higher, this is simply because the loss is not getting deflated by 0s.
- *Dropout: Worse performance* ❌ With the increased epochs, and the increased number of building metadata features, many of which are probably not that relevant, we did start to see some evidence of overfitting. To mitigate this, we tried adding dropout layers (p=0.2) to the building metadata branch, but a saw drastic performance hit in validation loss. This may be related to the same thing we saw with adding batchnorm to later layers, which is that this operation may not be that well suited for regression tasks. We can instead experiment with increased regularization in later epics. See **green** loss compared to blue.
- *Widening "chokepoint:": Moderate Improvement* ✅ We increased the size of the building metadata embedding from 8 to 16, thus widening the "chokepoint" where the weather and building metadata embeddings are concatenated to size 24. This also involved modifying the fourth layer in the building branch from 8 to 16, and did not involve removing any layers, since we previously had jumped from 32 down to 8. The motivation for this was that accuracy might improve by having more building information carried through and combined with the weather data, given the increased number of building metadata features. See improvement in **red** loss (final MVP model) compared to **blue**, or the **orange** compared to **pink** for the version with the previous implementation of masked loss. Interestingly, when experimenting with widing the chokepoint when splitting out by end use, we noticed in particular this drastically improved the performance on predicting the "other" end use, likely because of the large number of building features that influence this.

Results: We get 14.6%, 11.5%, 8.5%, 24.3%, and 6.9% Median Absolute Percentage Error (APE) on total energy for upgrades 1, 3, 4, 6, and 9 respectively, and 4.1% median APE on baseline consumption. The metrics haven't changed much for the HVAC upgrades, although it is worth nothing that since we are now predicting the total energy consumption, the absolute error in kWh has increased, but commensurately with denominator. However, for baseline energy consumption, the absolute error actually barely increased, resulting in a 2.5 percentage point improvement. We also added the weighted mean absolute percentage error (wMAPE) which can be thought of as relative absolute error relative in the context of the magnitude of the predictions. Compared to median and in particular mean APE, this will somewhat dampen the effect of the model's relatively poorer performance on smaller values which may not actually matter that much. The least impressive performance is on the HPWH, where the median error is ~24% and the mean is close to 100%. However, this is still a 2-4fold improvement over the bucketed model. The model is still also struggling on homes without heating, which is the only cross-section of the housing stock where the bucketed model beats the surrogate model. One important note is that we changed the implementation for calculating the bucketed metrics. Previously, the "true" values that the bucketed predictions were compared to were the true savings *only over the upgraded end use* since this is the way that model is set up. However, when examining some inconsistencies with the HPWH error, we realized that this is problematic, because this is not actually the true savings value, which should be considered over all end uses. In the case of water heaters, this makes a substantial difference due to the 1.0 interaction factor with the heating system: mean national savings are ~3200kWh when just considering at the water heater end use, but are ~2200kWh when considering all end uses. The inability to calculate savings over all end uses is a fundamental limitation of the bucketed model which should be captured in the evaluation. Thus, we switched to comparing the building-level bucketed savings predictions directly to the actual total savings predictions. This update has the added benefit of making the metrics even more comparable by calculating metrics over the same test set, rather than over all data points for the bucketed model. Note that for the bucketed baseline errors we don't have predictions over all end uses, so we calculate the sum over all end use buckets (which does not

include other) and compare it to the true consumption excluding “other” end uses. Finally, due to this same issue with buckets predicting by end use, we cannot compare absolute consumption errors, and so these have been removed from the evaluation table for the bucketed model.

5. Modifying Home Type Universe to Match GGRF

Best model/experiment: new_home_types@231c367d5a1a42f49d5b8ded2808c757

Training time: 8h on ~1.5M training samples using 1 GPU/85 GB memory.

Summary:

Universe: In order to align more closely with the homes types that we will be serving for GGRF, we modified the universe of homes we are training on in the following ways:

- Exclude homes with shared HVAC or shared WH (-2,600). Note the previously we were accidently including homes with shared water heaters but without an indicator feature for flagging whether or not it was shared, which likely has a negative impact on performance.
- Include mobile homes (+ 35,000)
- Include MF with <5 units without shared HVAC or shared WH (+ 20,000)

The reason we elected to include MF with <5 units is that ResStock actually does not represent any SF attached homes with <5 units, even though this is obviously a home type that does exist (e.g, duplex). We decided that the only home types that were out of scope are those with shared HVAC or horizontally stacked MF units with more than 5 units.

Features:

Added two new features to capture these new home types: `is_mobile_home` (bool) and `unit_level_in_building` (categorical) which described the vertical position of a multi-family unit within a building.

Results: We get 17%, 13.3%, 10.8%, 23.5%, and 9.9% Median Absolute Percentage Error (APE) on total energy savings for upgrades 1, 3, 4, 6, and 9 respectively, and 5.9% median APE on baseline consumption. For most upgrades, this amounts to about a 2 percentage point increase in median APE, with the exception of water heating (6), which went down slightly, likely because of the removal of the mistakenly included attached homes with shared water heating. Note that this is not exactly an apples to apples comparison with the last method because in addition to adding ~50K new homes, this is also completely different training/dev/test partition. Further, the metrics for the bucketed method are also less comparable, since that method does not provide predictions for mobile homes or multi-family.

If we look at the breakdown of test error by home type across all upgrades, we see that indeed the new home types, mobile and MF, do have slightly higher median percentage errors. Interesting, mobile homes have MUCH higher mean error, whereas MF has a much lower mean than median error, implying that they have long right and left tails respectively. We should probably dig further into this breakdown by upgrade id, and see if we can understand what is happening here, or if perhaps there are other important features that we have omitted.

home_type	median_absolute_error	median_APE_savings	MAPE_savings
SF Attached	680	13.1	50.1
SF	1032	13.2	45
MF	650	15.1	36.3
Mobile	550	14.5	144.9

Finally, we noticed then in the previous training run try, we mistakenly forgot to exclude homes with shared water heating, so had to rerun this one. However, the training ran for many more epochs, with the result of lower overall error, but much higher error for homes with no heating in baseline. See results here.

6. Adding Medium Efficiency Heat Pump from RAStock

Best model/experiment: add_med_eff_hps@81d5620d30b54bd3a279d0c152462874

Training time: 4.2 on ~2.0M training samples using 1 GPU/85 GB memory (implemented weather embeddings in between iter 5 and 6 which sped up training by ~40%)

Summary:

Upgrades:

We added a Medium Efficiency Heat Pump upgrade (11.05) and a Medium Efficiency Heat Pump with Basic Enclosure upgrade (13.01), resulting in a total of 8 upgrades, and a total of 2.9M(building, upgrade) samples.

Features:

Added a new feature since we now have multiple methodologies for heat pump sizing: `heat_pump_sizing_methodology` (str) which is `HERS` in the new upgrades being added, and `ACCA` for all other heat pumps. Note that we also implemented a bug fix Fix sumo upgrade feature bugs, where the fuel type for the heat pump water heater was not updated correctly to `Electricity` and the features `has_heat_pump_dryer`, `has_induction_range` were accidentally not included, meaning that the differences in the whole home upgrade were not fully reflected in the features. This resulted in higher error across the board, in particular for the directly related upgrades: hpwh (6) and whole home (9).

Results: We get 14.5%, 11.9%, 8.9%, 23.1%, 6.5%, 10.7%, 11.7% Median Absolute Percentage Error (APE) on total energy savings for upgrades 1, 3, 4, 6, 9, 11.05, 13.01 respectively, and 4.4% median APE on baseline consumption. These are relatively consistent results on the old upgrades (most of the improvement was due to the bugfix), and the performance on the new upgrades aligns roughly with what we would expect: ~11% on the medium efficiency heat pump which is in between the performance of the min and max hp, and 12% on the mean eff hp + insulation, which is higher than the heat pump only upgrades and lower than insulation only. The trend of worse performance on electric resistance homes continues, likely since these don't get the assistance of getting a fuel zeroed out in the upgrade.